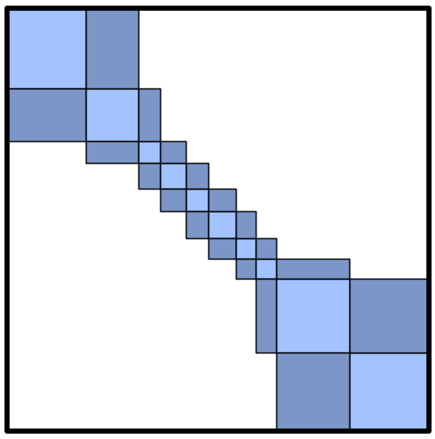
Block Tridiagonal Matrices in Electronic Structure Calculations



Dan Erik Petersen

Department of Computer Science

Copenhagen University

A thesis submitted for the degree of

Doctor of Philosophy

April 30, 2008

Revised August 7, 2008

System: $\LaTeX\,2_\varepsilon$

Typeface: Palatino 12pt

Submitted: April 30, 2008

Revised: August 7, 2008

For truth and beauty.

Summary

This thesis focuses on some of the numerical aspects of the treatment of the electronic structure problem, in particular that of determining the ground state electronic density for the non–equilibrium Green's function formulation of two–probe systems and the calculation of transmission in the Landauer–Büttiker ballistic transport regime. These calculations concentrate on determining the so–called Green's function matrix, or portions thereof, which is the inverse of a block tridiagonal general complex matrix.

To this end, a sequential algorithm based on Gaussian elimination named Sweeps is developed and compared to standard Gaussian elimination, where it is shown to be qualitatively quicker for the task of determining the block tridiagonal portion of the Green's function matrix. The Sweep algorithm is then parallelized via a straightforward approach in order to enable moderate speedup and memory distribution.

The well known block cyclic reduction algorithm first developed by Gene Golub is then presented and analyzed for further expanding our parallel options, and finally a new hybrid method that combines block cyclic reduction and a form of Schur complement calculation is introduced.

The parallel algorithms are then benchmarked and the new hybrid method is shown to possess promising speedup characteristics for common cases of problems that need to be modeled.

Resumé

Denne afhandling fokuserer på nogle af de numeriske aspekter ved løsningen af elektronstruktur problemet, specielt udregningen af grundtilstandselektrontætheden i ikke–ligevægts Green's funktion formuleringen af to–elektrode systemer samt beregningen af transmission af elektroner i Landauer–Büttiker regimet. Disse udregninger koncentrerer sig hovedsageligt om bestemmelsen af en Green's funktion matrix, eller dele deraf, som er en inversion af en blok tridiagonal generel matrix med komplekse elementer.

Til dette formål udvikles en "Sweeps" algoritme baseret på Gauss eliminering. Denne sammenlignes med standard Gauss eliminering, og det vises at Sweeps er kvalitativt hurtigere til beregningen af den blok tridiagonale del af et Green's funktion matrix. Denne nye algoritme paralleliseres naivt for at udvinde et moderat speedup og udnytte den kollektive hukommelse til rådighed i et netværk af samarbejdende maskiner.

Den velkendte blok cyklisk reduktions algoritme, oprindeligt beskrevet af Gene Golub, præsenteres og analyseres for at udvide vores repertoire af parallele algoritmer. Derefter udvikles en ny mere effektiv hybrid metode, der kombinerer blok cyklisk reduktion med en form for Schur komplement beregning.

De parallele algoritmer bliver derefter målt på under kørsel i et parallelt miljø, og den nye hybride metode vises at have fordelagtig speedup for typiske problemstillinger inden for modellering af elektronstrukturer af to–elektrode systemer.

Acknowledgements

My acknowledgements go to many beyond those listed here — but especially those listed here — for making the past 3 years of life and work a time I have enjoyed, am eternally grateful for, and will fondly remember going forward into the great unknown.

To Stig Skelboe for giving me the independence and freedom in the work I did, while at the same time guiding and protecting me from the pitfalls and perils that can strike a Ph.D student.

To Per Christian Hansen for having the infectious youthful exuberance in all he does that inspires to keep juggling matrices and vectors, and see what comes out.

To Kurt Stokbro for the many fruitful hours of work which brought about some great quantum simulation code and his family, for being open and warm to me while visiting Stanford. Kurt proved one can easily discuss Hartree–Fock theory and configuration interaction while waiting in line at the tax office.

To Hansu–san, my Ph.D. partner–in–crime, for the countless hours spent coding, discussing, grumbling and laughing. We never did find Scarlett in Tokyo, but we did experience just about everything else Japan could surprise us with.

To my brother Allan, whom I look up to[1] and can always make me laugh. In hindsight, my first experience with quantum mechanics was his room; always in some state of quantum superposition, and if lucky, upon observation, would collapse to stark orderliness rather than remain a dense entangled soup of indistinguishable particles.

To my parents, Elena and Erik, for all the love and support — os quiero muchísimo más que palabras pueden expresar.

To Pernille, my truth and beauty, for her strength, unwavering support, and unconditional love. I can’t imagine life without you.

Østerbro, April 2008

Dan Erik Petersen

[1]He’s taller.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the beginning the universe was created. This has made a lot of people very angry and has been widely regarded as a bad move.

Douglas Adams – The Hitchhiker's Guide To The Galaxy

1.1 Life, the Universe and Everything

42

Forty–two. That might be the first thought for some when posed with the Question of what the meaning of life, the universe, and everything is. This is because in a fictional universe, this really was the answer to that Question, and the result of millions of years of computing power, and indeed, why the planet Earth came to be. It was in order to compute a more precise form of the Question in order to better understand the Answer. Earth was not where the computer was; it *was* the computer, complete with useful biological computing elements. It took 10 million years to compute the Question, and unfortunately, minutes from that final return statement, Earth was destroyed in order to make way for a hyperspatial express route.

This thesis, however, strives not to satisfy this unsolved problem, but to try to do what every scientist has done since history remembers: break the problem down into smaller[1], more manageable questions, and attempt to answer them one by one in the great hope of achieving a greater understanding of,

[1]Maybe *much* smaller.

indeed, life, the universe, and everything — or at least why a transistor might behave the way it does — or how a molecule of water might bend. Oh, and to do all this in less than 10 million years.

1.2 Overview

This thesis is divided into the following chapters and appendices, which will be described here briefly. Chapter 1, the current chapter and not as sober and serious as the rest, serves as the introduction.

Chapter 2 provides a brief theoretical background of electronic structure theory and relevant computational methods that give rise to the particular kind of linear algebraic matrix structure which this thesis has focused its efforts on. The chapter ends with a notational guide to the mathematics and programmatic constructs used later on.

Chapter 3 introduces a series of sequential algorithms that lay the foundation for two different strategies for inverting our special matrix, namely standard Gaussian elimination and Sweeps. Furthermore, these are developed in two variants that either calculate the full inverse, or only a specific portion of it. A complexity analysis is carried out on all methods in order to characterize them qualitatively.

Chapter 4 focuses on parallel algorithms for the inversion problem, where we first start with a short overview of what we assume is our parallel computing environment. Three algorithms are presented, where the first is a direct parallelization of the sequential Sweeps method. Block cyclic reduction is introduced and treated, and finally a Hybrid method combining block cyclic reduction with a form of Schur complement calculation is presented. Again, complexity analysis is carried out for each method.

Chapter 5 moves on to benchmark the parallel algorithms and determine their execution characteristics in a practical setting. Speedup predictions are carried out and compared to execution times, and remarks are made as to their validity and what we can carry forward.

Chapter 6 concludes the thesis, with a brief overview of the significant conclusions drawn, and with a view towards further work.

A series of three articles is appended to the thesis, of which two have been accepted for publication in the Journal of Computational Physics [1] and Physical Review B [2]. A third has been submitted to Physical Review B, and is available from arXiv [3]. A fourth article [4] concerning the hybrid method is currently in preparation in cooperation with Eric Darve and Song Li from Stanford University, Kurt Stokbro and Stig Skelboe from the University of Copenhagen and Per Christian Hansen from the Technical University of Denmark.

Though there may be a degree of overlap between the body of this thesis and [1], the methods presented in Chapters 3 and 4 are designed to return a different result, serving a different purpose in electronic structure calculations. The thesis body looks to determine the so–called ground state density of a system, while [1] uses similar methods to calculate transmission characteristics for systems that have already had their ground state densities determined.

Chapter 2

Theory

Thirty-one years ago [1949], Dick Feynman told me about his "sum over histories" version of quantum mechanics. "The electron does anything it likes," he said. "It just goes in any direction at any speed, forward or backward in time, however it likes, and then you add up the amplitudes and it gives you the wave-function." I said to him, "You're crazy." But he wasn't.

Freeman Dyson

2.1 Theoretical Background

Since the discovery of the electron and the rise of quantum mechanics and particle physics, it has been shown that many natural phenomena of interest can be well explained by how electrons behave in the vast environment of other electrons, nuclei, electromagnetic fields and other fundamental forces that the universe exhibits.

The challenge addressed by this thesis stems ultimately from the problem of describing and understanding how matter behaves. Quantum mechanics and statistical mechanics have come a long way in providing theoretical models of electron behavior that account well for the behavior of matter under a large variety of conditions, and equations have been devised and studied to this effect.

However, we are not interested in pursuing a complete description of matter that can take account for any condition imaginable in the universe. Our goal, being one of understanding the behavior of matter in our everyday lives, allows us to limit ourselves to equations devised to describe the more mundane behavior of matter, in comparison with the extreme conditions the universe can exhibit. We can thus omit relativistic effects, quantum electrodynamics and electromagnetic fields in the model of our system.

Relativistic effects[1] for atoms with an atomic number less than about 25 (Manganese) can be neglected, since it is only for heavier nuclei that *core*[2] electrons move at relativistic velocities. Ultimately, these effects can be included via so–called relativistic effective core potentials in conjunction with the fundamental equation Eq. (2.1) we are about to present, but this will not be addressed in this work.

The effects of quantum electrodynamics is not accounted for, as we will be investigating the properties of matter that arise without invoking the conditions that require describing the strong nuclear force explicitly, while magnetic effects can be included as *Zeeman* terms in Eq. (2.1). Thus it is from the fundamental equations presented in the following section from which we take our starting point.

2.1.1 Basic Equations

The many–body hamiltonian operator that governs the behavior of a system of interacting electrons and nuclei in atomic units takes the form

$$\begin{aligned}\hat{H} = &-\frac{1}{2}\sum_i \nabla_i^2 + \sum_{i,I}\frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2}\sum_{i\neq j}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \\ &-\sum_I \frac{1}{2M_I}\nabla_I^2 + \frac{1}{2}\sum_{I\neq J}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|},\end{aligned} \tag{2.1}$$

where summations over i and j correspond to electrons, and summations over I and J correspond to nuclei. The kinetic energy terms employ the spatial differential operator ∇ where the Ith nuclei, with atomic number Z_I, has a mass ratio of M_I to that of an electron. The three dimensional vector positions of the ith electron and Ith nuclei are denoted as $\mathbf{r}_i$ and $\mathbf{R}_I$, respectively.

Examining Eq. (2.1), we can identify the five terms it is composed of as being, in order, the kinetic energy of the electrons, the attractive electron–nuclei interaction, the electron–electron repulsive interaction, the kinetic energy of

[1]Relativistic effects are properties or behavior of matter that is only well–described by incorporating the theory of relativity. This becomes necessary for high–energy physics.

[2]Core electrons are electrons occupying inner orbital shells and are thus tightly bound to the positive nucleus and shielded from outside effects by weaker bound valence electrons in outer orbital shells.

the nuclei, and the nuclei–nuclei repulsive interaction. This can be written down in a compact form in the following manner:

$$\hat{H} = \hat{T}_e(\mathbf{r}) + \hat{V}_{eN}(\mathbf{r}, \mathbf{R}) + \hat{V}_{ee}(\mathbf{r}) + \hat{T}_N(\mathbf{R}) + \hat{V}_{NN}(\mathbf{R}) \tag{2.2}$$

where the various kinetic $\hat{T}$ and potential $\hat{V}$ operators depend on the positions of the electron ($_e$) and nuclei ($_N$) positions $\mathbf{r}$ and $\mathbf{R}$, respectively.

2.1.2 Born–Oppenheimer Approximation

One of the first reasonable steps we can take in order to make Eq. (2.1) more tractable to solve is to employ what is known as the *Born–Oppenheimer*, or *adiabatic*, approximation.

The motivation behind this approximation is two–fold. The first, is that Eq. (2.2) would be much more tractable to solve if it were *separable*, but the electron–nuclei interaction term $\hat{V}_{eN}(\mathbf{r}, \mathbf{R})$ prevents this as it depends explicitly on both the positions of the electrons $\mathbf{r}$ as well as the positions of the nuclei $\mathbf{R}$.

The second motivating factor comes from the observation that the mass of an electron is negligible in the face of the atomic masses in the system, i.e. that $M_I \gg 1$. Thus the nuclei, being orders of magnitude heavier compared to the light, agile electrons, can be assumed to remain stationary from the point of view of an electron. As the spatial configuration of nuclei might change, we assume electrons will instantly find themselves adjusted to the new spatial configuration of nuclei.

Thus we fix the spatial configuration of nuclei $\mathbf{R}$ to some value $\mathbf{R}_0$, as seen in Fig. 2.1, effectively parameterizing our equations. In this way, we have simplified our earlier problem in subsection 2.1.1 to that of considering a system of moving electrons interacting with stationary nuclei as well as amongst themselves. Furthermore, in fixing the nuclei positions, the potential energy from the nuclei–nuclei interaction becomes constant and is in the following expression of the hamiltonian contained in the term E_{NN}. We are also able to effectively disregard the kinetic energy $\hat{T}_N(\mathbf{R}_0)$ of the nuclei, and omit it from our new parametrized hamiltonian:

$$\hat{H} = \hat{T}_e(\mathbf{r}) + \hat{V}_{eN}(\mathbf{r}; \mathbf{R}_0) + \hat{V}_{ee}(\mathbf{r}) + E_{NN} = \hat{H}_e + E_{NN}. \tag{2.3}$$

The new term $\hat{H}_e$ is known as the electronic hamiltonian, as it describes the motion of electrons in a fixed environment of atomic nuclei. It can be further broken down into having terms of *internal* and *external* character by writing

$$\hat{H}_e = \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{ext}}(\mathbf{r}; \mathbf{R}_0) + \hat{V}_{\text{int}}(\mathbf{r}) \tag{2.4}$$

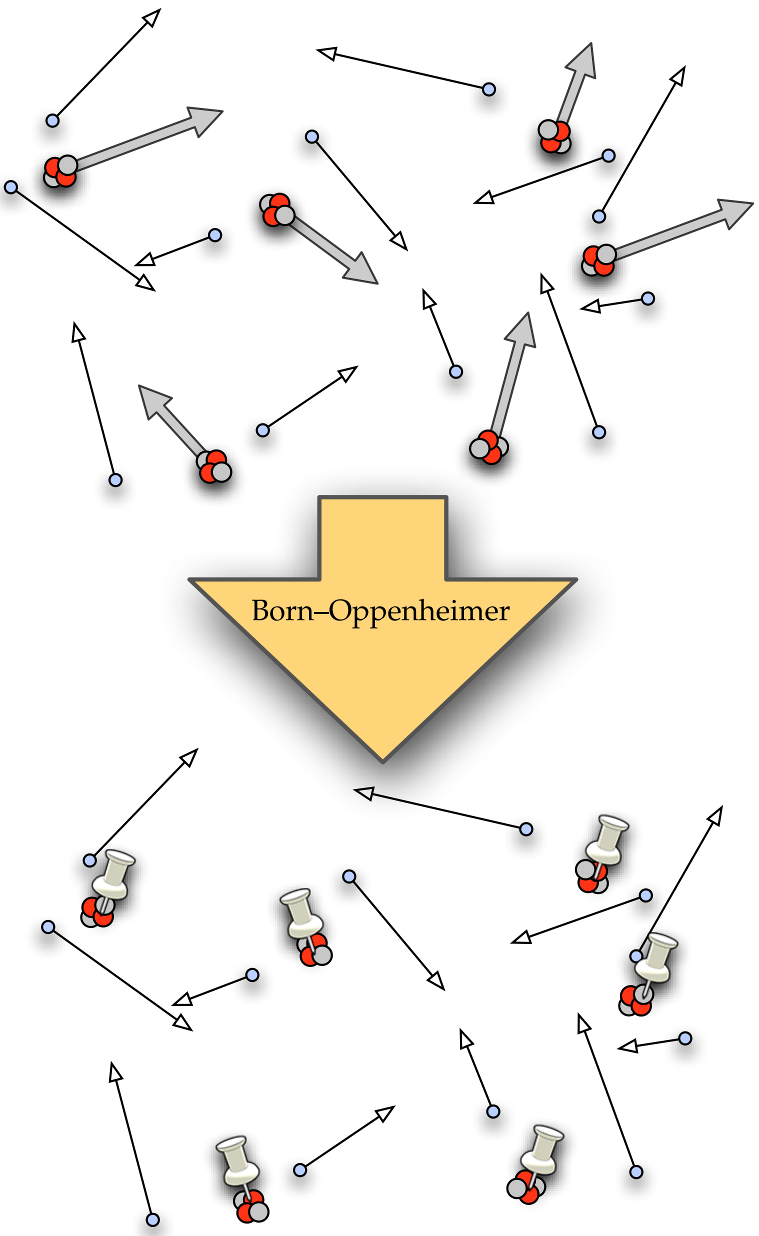


Figure 2.1: Visualizing the Born–Oppenheimer approximation, where all nuclei are fixed in space to a set of positions $\mathbf{R}_0$, while the electrons are still free to move. Any change in the spatial configuration $\mathbf{R}_0$ of nuclei is assumed to happen slowly enough for the electrons to adiabatically adapt to.

where we have classified the action of nuclei upon the electrons via an external potential $\hat{V}_{\text{ext}}$, and the electron–electron interaction as that of an internal potential $\hat{V}_{\text{int}}$.

2.1.3 The Schrödinger Equation

Having a hamiltonian in Eq. (2.3) that describes the energy in our system of electrons helps us, but only brings us so far. We need a formulation of an actual problem we can try to solve, and ideally, this problem is a connection to how the real world behaves, and the solution to which tells us something about the behavior of electrons in the real world. This is where the non–relativistic, time–independent Schrödinger equation comes to our aid.

$$\hat{H}_e \Psi = \varepsilon_e \Psi \tag{2.5}$$

This equation, a so–called eigenproblem and one of the fundamental postulates of quantum mechanics, describes how the electronic hamiltonian operator $\hat{H}_e$ for a system of electrons is related to its stationary solutions. These stationary solutions turn out to be the eigensolutions to the above equation, where each eigenfunction solution Ψ is known as a *many–body electronic wavefunction*, and the associated eigenvalue ε_e represents the energy associated with the eigenfunction.

The electronic wavefunction Ψ that enters into Eq. (2.5) describes the state of all electrons in the system, and is a function of the set of the spatial locations of each electron $\{\mathbf{r}_i\}$. Furthermore, it is also affected by the set of nuclei positions $\{\mathbf{R}_0\}$ parametrically. The interpretation of Ψ can be likened to that of a probability amplitude dictating the distribution of the electrons in space. Finally, the electronic energy associated with each solution wavefunction also depends implicitly on nuclei positions:

$$\Psi = \Psi(\{\mathbf{r}_i\}; \{\mathbf{R}_0\}), \qquad \varepsilon_e = \varepsilon_e(\{\mathbf{R}_0\}) \tag{2.6}$$

From this point on, we assume implicitly the parametric dependence of the wavefunction and energies on the positions of the nuclei $\{\mathbf{R}_I\}$, and we omit this from our expressions.

2.1.4 The Pauli Exclusion Principle

From Eq. (2.3) we have a hamiltonian operator which dictates the behavior for a given system of N electrons moving about a fixed arrangement of atomic nuclei. This set of N electrons are described by their locations in space by the set of their spatial coordinates $\{\mathbf{r}_i\}$. In order to fully describe the electrons, however, we have to include information on a fundamental property all electrons possess: their *spin*[1] state. To incorporate this property, the three dimensional coordinates that make up the spatial coordinate $\mathbf{r}$ of an electron is augmented with a fourth degree of freedom, the spin coordinate ω, and we can then write

$$\mathbf{x} = \{\mathbf{r}, \omega\}. \tag{2.7}$$

We are now able to fully describe the electrons in the many–body wavefunction in terms of the augmented coordinates $\mathbf{x}$. This change, by itself, does not affect us explicitly since the electronic hamiltonian $\hat{H}_e$ from Eq. (2.3) that we apply in the Schrödinger equation Eq. (2.5) is not spin–dependent. This change is useful for another reason, since we will use it to enforce a property that electrons obey, known as the Pauli exclusion principle. This principle imposes the condition that no two electrons may possess the same quantum state at the same point in space. In other words, they may not occupy both the same position in space *and* have the same spin state.

This can be enforced by constructing the wave–function of the electrons Ψ such that it is *anti–symmetric*[2]. Thus an interchange of any two electrons in the system would change the sign of the wavefunction in the following manner:

$$\Psi(\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_N) = -\Psi(\mathbf{x}_1, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_N). \tag{2.8}$$

2.1.5 The Many–Body Wavefunction

The many–body wavefunction can be interpreted as a probability amplitude that describes the likelihood of finding an electron in a certain position in space. A main property of the many–body wavefunction that we assume, is the property that the wave function is *normalized*, i.e. for a wave function describing N electrons we have

$$\int |\Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)|^2 \, \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \ldots \mathrm{d}\mathbf{r}_N = 1, \tag{2.9}$$

and thus N electrons are described to exist in the system with unit probability. The *single–particle* density in our N electron system is defined as

$$n(\mathbf{r}) = N \iint \ldots \int |\Psi(\mathbf{r}, \mathbf{r}_2, \ldots, \mathbf{r}_N)|^2 \, \mathrm{d}\mathbf{r}_2 \mathrm{d}\mathbf{r}_3 \ldots \mathrm{d}\mathbf{r}_N \tag{2.10}$$

[1]Spin is a property of intrinsic angular momentum assigned to each electron.

[2]An anti–symmetric function obeys $f(x, y) = -f(y, x)$ on the interchange of any two variables.

which yields the intuitive result that integrating the single–particle density over all of space will yield the total number of particles in the system, i.e.

$$\int n(\mathbf{r})\ \mathrm{d}\mathbf{r} = N. \tag{2.11}$$

2.2 Density Functional Theory

Determining the ground state wave–function for the Schrödinger Eq. (2.5) in terms of the many–body electron wave function Eq. (2.6) is only really tractable for simple systems composed of relatively few electrons. Full configuration interaction (FCI) methods, for example, even when adapted for massively parallel calculations [5, 6] still prove unable to handle large systems of molecules due to the exponential increase in the number of variables as the number of electron configurations in the search for the ground state wave function increases.

One approach that transforms the many–body Schrödinger wave equation into a far more tractable one–electron equation for numerical solution is that provided via the *density functional theory* formalism, or DFT for short. A central property of DFT is that it recasts the basic variable of our equations from being the ground state electronic wave function $\Psi_0(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ to that of the ground state electron density $n_0(\mathbf{r})$, where

$$n_0(\mathbf{r}) = \iint \ldots \int |\Psi_0(\mathbf{r}, \mathbf{r}_2, \ldots, \mathbf{r}_N)|^2\ \mathrm{d}\mathbf{r}_2 \mathrm{d}\mathbf{r}_3 \ldots \mathrm{d}\mathbf{r}_N \tag{2.12}$$

and which effectively reduces the $3N$ degrees of freedom to minimize over down to just 3 for an N–electron system.

By construction, basic DFT is a theory of the *ground state* condition of electrons in a system, whereas FCI can handle excited states. However, many useful properties of systems can be determined from the ground state.

One difficulty with DFT is that, although it is in principle an *exact* theory of the ground state of a system, it is limited by the lack of knowing the exact form the so–called exchange–correlation functional. Thus we employ approximations to this functional, which can have varying levels of sophistication and numerical overhead in calculation, while still providing acceptable results. The results will then be exact ground state densities, but for a system of electrons for which the approximated exchange–correlation functional would apply. The trick is then to have the approximated exchange–correlation functional reflect the behavior of the true functional as much as possible, but this is not the focus of this thesis. We will, however, give a brief overview of basic DFT, starting with the fundamental work by Hohenberg and Kohn.

2.2.1 Hohenberg–Kohn Theorem

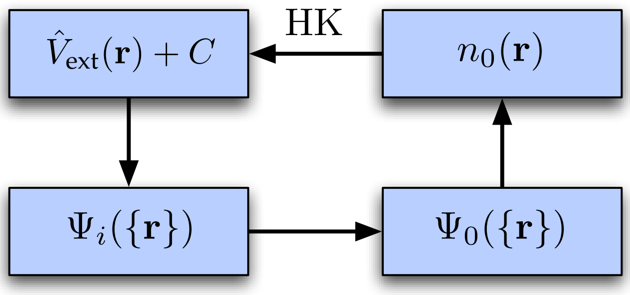

$$\begin{tikzcd}
\hat{V}_{\text{ext}}(\mathbf{r}) + C \arrow[d] & n_0(\mathbf{r}) \arrow[l, "\text{HK}"'] \\
\Psi_i(\{\mathbf{r}\}) \arrow[r] & \Psi_0(\{\mathbf{r}\}) \arrow[u]
\end{tikzcd}$$

Figure 2.2: Visualizing the Hohenberg–Kohn implications, where C denotes a constant.

The seminal paper by Hohenberg and Kohn [7] laid the foundation of all the modern methods based on DFT. The paper addressed the issue of determining the ground state properties of an electronic system with the electron hamiltonian from Eq. (2.4) in a variational manner using the electron *density* of a system rather than the electron wave function. Two theories by Hohenberg and Kohn (HK), presented and proven in [7], are presented briefly here.

HK Theorem 1. *For any system of multiple interacting particles in an external potential $\hat{V}_{ext}(\mathbf{r})$, the potential $\hat{V}_{ext}(\mathbf{r})$ is determined uniquely, except for a constant C, by the ground state particle density $n_0(\mathbf{r})$.*

HK Corollary 1. *Since the hamiltonian is thus fully determined, except for a constant shift of the energy, it follows that the many–body wavefunctions for all states, both ground and excited, are determined.*

The first theorem by Hohenberg and Kohn tells us that if we are able to find the correct ground state density of the system $n_0(\mathbf{r})$, then we have uniquely determined all other properties of the many–body system.

HK Theorem 2. *A* universal functional *for the energy $E[n]$ in terms of the density $n(\mathbf{r})$ can be defined, valid for any external potential $\hat{V}_{ext}(\mathbf{r})$. For any particular $\hat{V}_{ext}(\mathbf{r})$, the exact ground state energy of the system is the global minimum value of this functional, and the density $n(\mathbf{r})$ that minimizes the functional is the exact ground state density $n_0(\mathbf{r})$.*

HK Corollary 2. *The functional $E[n]$ alone is sufficient to determine the exact ground state energy and density. In general, excited states of the electrons must be determined by other means.*

This second theorem lends us an approach in order to determine the desired ground state density. The theorem proves that for any external potential we know that a functional $E[n]$ exists whose global minimum is the exact ground state energy. Furthermore, the actual density that minimizes $E[n]$ will be the ground state density $n_0(\mathbf{r})$. However, this is where the help stops. The HK theorems tell us nothing of what shape or form this exact functional has.

An immediate issue is that the work put forth by Hohenberg and Kohn in [7] only treats the case for many–body systems with a non–degenerate[1] ground state energy and at zero temperature. Mermin [8] extended the work to handle nonzero temperatures, however, and the work of Levy [9, 10] and Lieb [11, 12, 13] provided improvements and clarifications, and importantly, incorporated degenerate ground states into the realm of DFT. An approach and proof that incorporates degenerate ground states follows in lieu of a proof for the non–degenerate case here.

2.2.2 Extension to the Degenerate Case

We consider N electrons in an external potential $\hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)$ as defined in the electronic hamiltonian $\hat{H}_e$ from Eq. (2.4) and proceed as done in [9, 14].

For all *N–representable*[2] densities $n(\mathbf{r})$, i.e. the densities constructible from an arbitrary antisymmetric wave function $\Psi(\mathbf{r}) = \Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$, we can define the Levy–Lieb functional

$$F[n] = \min_{\Psi \rightsquigarrow n(\mathbf{r})} \langle \Psi(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi(\mathbf{r}) \rangle, \tag{2.13}$$

where the minimum is taken over all wavefunctions $\Psi(\mathbf{r})$ that construct the density $n(\mathbf{r})$ (cf. Eq. (2.10)). The functional F is universal in the sense that it mentions nothing of the system we are dealing with nor of the external potential $\hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)$, i.e. it is independent of $\mathbf{R}_0$.

[1]A non–degenerate energy level of a system is one for which there exists only *one* electron configuration, otherwise the energy level is said to be degenerate if multiple configurations exist.

[2]The original paper [7] by Hohenberg and Kohn work with *V–representable* densities, i.e. the densities that can be constructed for an arbitrary external potential $\hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)$.

If we name E_0 and $n_0(\mathbf{r})$ to be the ground state energy and density, respectively, then the two basic theorems of DFT are

$$E[n] \equiv \int \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)n(\mathbf{r})\ \mathrm{d}\mathbf{r} + F[n] \geq E_0 \tag{2.14}$$

for all N–representable $n(\mathbf{r})$, and

$$\int \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)n_0(\mathbf{r})\ \mathrm{d}\mathbf{r} + F[n_0] = E_0. \tag{2.15}$$

In order to prove the variational principle in Eq. (2.14), we introduce the notation $\Psi^n_{\text{min}}(\mathbf{r})$ for an electronic wave function that minimizes Eq. (2.13) such that

$$F[n] = \langle \Psi^n_{\text{min}} | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^n_{\text{min}} \rangle. \tag{2.16}$$

Writing $\hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0) = \sum_i \hat{V}_{\text{ext}}(\mathbf{r}_i;\mathbf{R}_0)$, we have

$$\int \hat{V}_{\text{ext}}(\mathbf{r})n(\mathbf{r})\ \mathrm{d}\mathbf{r} + F[n] = \langle \Psi^n_{\text{min}} | \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0) + \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^n_{\text{min}} \rangle \geq E_0, \tag{2.17}$$

due to the ground state being the lowest energy level. This proves the inequality in Eq. (2.14). Using the property of the ground state once again, we have

$$\begin{aligned} E_0 &= \langle \Psi_0(\mathbf{r}) | \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0) + \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi_0(\mathbf{r}) \rangle \\ &\leq \langle \Psi^{n_0}_{\text{min}}(\mathbf{r}) | \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0) + \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^{n_0}_{\text{min}}(\mathbf{r}) \rangle. \end{aligned} \tag{2.18}$$

Subtracting the external potential from both, we obtain

$$\langle \Psi_0(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi_0(\mathbf{r}) \rangle \leq \langle \Psi^{n_0}_{\text{min}}(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^{n_0}_{\text{min}}(\mathbf{r}) \rangle. \tag{2.19}$$

On the other hand, the definition of $\Psi^{n_0}_{\text{min}}$ tells us

$$\langle \Psi_0(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi_0(\mathbf{r}) \rangle \geq \langle \Psi^{n_0}_{\text{min}}(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^{n_0}_{\text{min}}(\mathbf{r}) \rangle, \tag{2.20}$$

which is only possible if both sides of the expression are equal, i.e.

$$\langle \Psi_0(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi_0(\mathbf{r}) \rangle = \langle \Psi^{n_0}_{\text{min}}(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^{n_0}_{\text{min}}(\mathbf{r}) \rangle. \tag{2.21}$$

This leaves us with

$$\begin{aligned} E_0 &= \int \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)n_0(\mathbf{r})\ \mathrm{d}\mathbf{r} + \langle \Psi_0(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi_0(\mathbf{r}) \rangle \\ &= \int \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)n_0(\mathbf{r})\ \mathrm{d}\mathbf{r} + \langle \Psi^{n_0}_{\text{min}}(\mathbf{r}) | \hat{T}_e(\mathbf{r}) + \hat{V}_{\text{int}}(\mathbf{r}) | \Psi^{n_0}_{\text{min}}(\mathbf{r}) \rangle \\ &= \int \hat{V}_{\text{ext}}(\mathbf{r};\mathbf{R}_0)n_0(\mathbf{r})\ \mathrm{d}\mathbf{r} + F[n_0], \end{aligned} \tag{2.22}$$

and thus we have proven Eq. (2.15).

We can also see how Eq. (2.21) tells us that DFT can handle systems with both degenerate and non–degenerate ground states. If the system at hand has a non–degenerate ground state, then $\Psi_0(\mathbf{r}) = \Psi^{n_0}_{\text{min}}$, otherwise $\Psi^{n_0}_{\text{min}}$ corresponds to one of the ground states, and the others can still be found. Therefore the ground state density $n_0(\mathbf{r})$ can determine all ground state wave functions (or wave function, in the non–degenerate case), from which further ground state properties can be calculated. This means that all ground state properties are essentially functionals of the ground state density. This lets us work with the density $n(\mathbf{r})$ as a basic variable, rather than the electronic wave function $\Psi(\mathbf{r})$.

The relations derived here leave us with a methodology for determining ground state properties of a many–body system. Nothing tells us, however, how the function F in Eq. (2.13) is explicitly formed. If we can construct a satisfying approximation to the true functional $F[n]$, for Eq. (2.14), we can then perform a minimization of this equation, and can accordingly obtain approximations to both the true ground state energy E_0 and the true ground state density $n_0(\mathbf{r})$.

2.2.3 Kohn–Sham Approach

Many methods attempt to tackle the many–body DFT approach from the previous section, relying on approximations to $F[n]$ via the Thomas–Fermi approach [14]. These methods suffer from the necessity to approximate the kinetic energy term $\hat{T}_e(\mathbf{r})$. Unfortunately, this term contributes significantly to the total energy, and as such, the Thomas–Fermi approach predicts a lack of shell structure in atoms and the absence of chemical bonding in molecules and solids, which is behavior clearly not observed in nature.

Based on this fallacy of the Thomas–Fermi approach in predicting behavior of examined systems, it was not until the famous paper by Kohn and Sham [15] that the task of devising good approximations to the energy functional $E[n]$ was satisfied.

The philosophy of the Kohn–Sham (KS) approach is to attempt to replace the problem of a many–body system obeying the hamiltonian Eq. (2.1) with a different auxiliary system that will have the same solution, but be easier to solve. This approach is essentially an *ansatz*[1], since nothing is known about how to determine this auxiliary system.

[1]An ansatz is an assumed form for a mathematical statement that is not based on any underlying theory or principle.

$$\begin{tikzcd}
\hat{V}_{\text{ext}}(\mathbf{r}) + C \arrow[d] & n_0(\mathbf{r}) \arrow[l, "\text{HK}"'] \arrow[r, leftrightarrow, "\text{KS}"] & n_0(\mathbf{r}) \arrow[r, "\text{HK}_0"] & \hat{V}_{\text{KS}}(\mathbf{r}) \arrow[d] \\
\Psi_i(\{\mathbf{r}\}) \arrow[r] & \Psi_0(\{\mathbf{r}\}) \arrow[u] & \psi_i(\mathbf{r}) \arrow[u] & \psi_{i=1\ldots N}(\mathbf{r}) \arrow[l]
\end{tikzcd}$$

Figure 2.3: Visualizing the KS ansatz that bridges the true multi–particle wavefunction system on the left with the auxiliary system on the right based on single–particle wavefunctions.

Essentially, the KS ansatz is the assumption that the ground state density of the fully interacting multi–body system is equal to that of some other, system where the electrons do not interact. This reduces the complexity of treating a system of N mutually interacting electrons to that of treating N individual non–interacting electrons, which is far more tractable to solve, and is readily handled by numerical computation.

The KS ansatz is based on the following two assumptions:

KS Ansatz 1. *The precise ground state density of a system can be represented*[1] *by the ground state density of an* auxiliary *system whose electrons do not interact.*

This first assumption is visualized in Fig. 2.3 which connects the ground state density $n_0(\mathbf{r})$ found for a non–interacting system with the true ground state density of a fully interacting many–body system.

KS Ansatz 2. *The auxiliary hamiltonian is formed such that it contains the regular kinetic energy operator $\hat{T} = -\frac{1}{2}\nabla$, but the potential is replaced by an* effective *potential $\hat{V}_{eff}$.*

An extremely useful simplification that can be made in the second assumption is that of using an effective potential $\hat{V}_{\text{eff}}$ that is *local*. That is to say, an electron at point $\mathbf{r}$ will only 'feel' its local neighborhood[2]. To see how the single–electron KS approach ties in that of many–electron HK theory, we start with the KS energy functional.

[1]HK theory [7] was based on a functional that is defined for all densities $n(\mathbf{r})$ that can be generated from an external potential $\hat{V}_{\text{ext}}(\mathbf{r})$, i.e. is V–representable. This is in contrast to the N–representable work by Levy and Lieb [9, 10, 11, 12, 13], and the functional F (cf. Eq. (2.13)) which is defined for any density $n(\mathbf{r})$ derivable from a many–body wavefunction $\Psi(\mathbf{r})$ composed of N electrons.

[2]This is related to the concept of *nearsightedness* as first described by Walter Kohn [16, 17].

The energy functional from the KS approach is

$$E_{\text{KS}}[n] = \hat{T}_0[n] + \int n(\mathbf{r}) \left[\hat{V}_{\text{ext}}(\mathbf{r}) + \frac{1}{2}\hat{V}_{\text{Hartree}}(\mathbf{r}) \right] \, \mathrm{d}\mathbf{r} + E_{\text{xc}}[n] \tag{2.23}$$

where $\hat{T}_0$ is the kinetic energy of a system with density n and lacking electron–electron interactions. The term $\hat{V}_{\text{Hartree}}$ is the classical Coulomb potential for electrons, also known as the *Hartree* potential. Finally, E_{XC} is the so–called *exchange–correlation* energy. The first term and integral can be calculated *exactly*, while the last term, the exchange–correlation functional, incorporates both the exchange and correlation energies, as well as the 'remainder' of electron kinetic energy and anything else that might be lacking in order for the energy functional E_{KS} to be the true energy functional E from Eq. (2.14). It is only this term for which we need to construct a satisfying, approximate functional, since the form of the true functional is not known.

One of the simplest exchange–correlation functionals applied in DFT is the local–density approximation (LDA) functional, first used and developed by Kohn and Sham [15]. This functional calculates the exchange–correlation energy of a point in a system to being the same as that of a point in a homogenous electron gas of the same density. The form and strategy in devising and constructing satisfactory E_{XC} functionals is not central to the topic of this thesis, but vast number of them exist and have been thoroughly studied [18, 19], and can be used relatively interchangeably.

The advantage of the KS method over the Thomas–Fermi approach dealing with the many–body system becomes clear when viewed in the light of how each term in Eq. (2.23) contributes to the total ground density energy E_0. It turns out that the kinetic energy term $\hat{T}_0$ for the non–interacting electrons can account for a large part of the full kinetic energy term $\hat{T}_e$ for the many–body interacting system [14]. Thus only a relatively small part of the energy contributed to the functional E_{KS} comes from E_{XC}, and thus our calculated ground state properties become relatively well approximated despite a rough estimate of the exchange–correlation energy.

If we apply the variational principle [14] to Eq. (2.23), we get

$$\frac{\delta E_{\text{KS}}[n]}{\delta n(\mathbf{r})} = \frac{\delta \hat{T}_0}{\delta n(\mathbf{r})} + \hat{V}_{\text{ext}}(\mathbf{r}) + \hat{V}_{\text{Hartree}}(\mathbf{r}) + \frac{\delta E_{\text{XC}}[n]}{\delta n(\mathbf{r})} = \mu \tag{2.24}$$

where μ is the Lagrange multiplier associated with the constraint of keeping the number of electrons in the system constant. We can now compare this with a similar equation, but where we *neglect* electron–electron interactions, getting

$$\frac{\delta E_{\text{KS}}[n]}{\delta n(\mathbf{r})} = \frac{\delta \hat{T}_0}{\delta n(\mathbf{r})} + \hat{V}_{\text{eff}}(\mathbf{r}) = \mu, \tag{2.25}$$

where we have $\hat{V}_{\text{eff}}$ as an *effective* potential that not only incorporates the nuclei and external effects, but also the so–called effective potential of the other electrons, although no explicit interaction is given. The equations Eq. (2.24) and Eq. (2.25) are identical, provided

$$\hat{V}_{\text{eff}}(\mathbf{r}) = \hat{V}_{\text{ext}}(\mathbf{r}) + \hat{V}_{\text{Hartree}}(\mathbf{r}) + \frac{\delta E_{\text{XC}}[n]}{\delta n(\mathbf{r})}, \tag{2.26}$$

whose solution we can find by solving a set of single–particle Schrödinger equations for noninteracting particles for each electron in the system

$$\left[\hat{T} + \hat{V}_{\text{eff}}(\mathbf{r})\right] \psi_i(\mathbf{r}) = \varepsilon_i \psi_i(\mathbf{r}), \quad i = 1, 2, \ldots, N \tag{2.27}$$

with the density being constructed up from each electron's fictitious[1] wave function $\psi_i(\mathbf{r})$ like this

$$n(\mathbf{r}) = \sum_{i=1}^{N} |\psi_i(\mathbf{r})|^2. \tag{2.28}$$

Thus we can present an overview of the KS single–particle equations, for which we solve for the ground state density $n_0(\mathbf{r})$, in Eqs. (2.29)–(2.31).

$$\begin{aligned}
\hat{V}_{\text{eff}}(\mathbf{r}) &= \hat{V}_{\text{ext}}(\mathbf{r}) + \hat{V}_{\text{Hartree}}(\mathbf{r}) + \hat{V}_{\text{XC}}(\mathbf{r}) && (2.29)\\
\hat{H}_{\text{KS}}(\mathbf{r}) &= \hat{T}(\mathbf{r}) + \hat{V}_{\text{eff}}(\mathbf{r}) && (2.30)\\
\hat{H}_{\text{KS}}(\mathbf{r})\psi_i(\mathbf{r}) &= \varepsilon_i \psi_i(\mathbf{r}) && (2.31)
\end{aligned}$$

One unanswered issue is that the effective potential from Eq. (2.29) depends on the average location of the electrons in the system, and this is not known a priori. As we will see in the following section, determining this is done via a self–consistent approach, using an initial guess of the density $n(\mathbf{r})$.

2.3 Solving The Kohn–Sham Equations

2.3.1 Basis Sets

In order treat the problem of dealing with the Schrödinger equation and its associated wavefunction on a computer, the wavefunction itself is expanded

[1]The wave function is fictitious in the sense that, individually, they are not related to any true physical description of the system, however, the consideration of them all together still yields the physical result we are looking for.

into components of a *basis set* $\{\phi_n\}$ comprised of a total of n basis functions.

$$\psi(\mathbf{r}) = \sum_{j=1}^{n} c_j \phi_j(\mathbf{r}) \tag{2.32}$$

The original wavefunction can then described as a column vector of coefficients with respect to the set of n basis functions like this:

$$\psi \equiv \mathbf{c} = \{c_1, c_2, \ldots, c_n\}^T, \tag{2.33}$$

where the superscript (^{T}) denotes transposition.

In principle, for an arbitrary wave function $\psi(\mathbf{r})$ to be able to be described exactly by a basis set expansion, a complete set of basis functions $\{\phi_\infty\}$ would be needed, and thus an infinite number of coefficients c_i would describe $\psi(\mathbf{r})$. Since this is not practical for implementation, a finite basis set is utilized, and the wave function would be described exactly only in terms of the subspace that the finite basis set would define.

If we look at how this affects the Schrödinger equation, either in Eq. (2.5) or in Eq. (2.31), we substitute Eq. (2.32) into either one and obtain

$$\sum_{j=1}^{n} c_j \hat{H} \phi_j(\mathbf{r}) = \varepsilon \sum_{j=1}^{n} c_j \phi_j(\mathbf{r}). \tag{2.34}$$

We multiply both sides by $\phi_i^*(\mathbf{r})$, where the superscript (*) denotes complex conjugation, getting

$$\sum_{j=1}^{n} c_j \phi_i^*(\mathbf{r}) \hat{H} \phi_j(\mathbf{r}) = \varepsilon \sum_{j=1}^{n} c_j \phi_i^*(\mathbf{r}) \phi_j(\mathbf{r}). \tag{2.35}$$

We then integrate both sides over $\mathbf{r}$ to get

$$\int \sum_{j=1}^{n} c_j \phi_i^*(\mathbf{r}) \hat{H} \phi_j(\mathbf{r}) \, \mathrm{d}\mathbf{r} = \int \varepsilon \sum_{j=1}^{n} c_j \phi_i^*(\mathbf{r}) \phi_j(\mathbf{r}) \, \mathrm{d}\mathbf{r}. \tag{2.36}$$

We can then move the integrals into the sums and obtain

$$\sum_{j=1}^{n} c_j \int \phi_i^*(\mathbf{r}) \hat{H} \phi_j(\mathbf{r}) \, \mathrm{d}\mathbf{r} = \varepsilon \sum_{j=1}^{n} c_j \int \phi_i^*(\mathbf{r}) \phi_j(\mathbf{r}) \, \mathrm{d}\mathbf{r}, \tag{2.37}$$

which we rewrite as[1]

$$\sum_{j=1}^{n} \mathbf{H}_{ij} c_j = \varepsilon \sum_{j=1}^{n} \mathbf{S}_{ij} c_j, \tag{2.38}$$

[1] Hence, the coefficients $\mathbf{c}$ are determined by the Galerkin method, ensuring the residual to be orthogonal to the space spanned by the n basis functions.

where the *hamiltonian* matrix $\mathbf{H}$'s elements are

$$\mathbf{H}_{ij} = \int \phi_i^*(\mathbf{r})\hat{H}\phi_j(\mathbf{r})\,\mathrm{d}\mathbf{r} \tag{2.39}$$

and the *overlap* matrix $\mathbf{S}$'s elements

$$\mathbf{S}_{ij} = \int \phi_i^*(\mathbf{r})\phi_j(\mathbf{r})\,\mathrm{d}\mathbf{r}. \tag{2.40}$$

The main result is that Eq. (2.38) shows us that we have transformed our operator–based Schrödinger equation to the following matrix–based generalized eigenvalue problem:

$$\mathbf{Hc} = \varepsilon\mathbf{Sc}. \tag{2.41}$$

From Eq. (2.40), we can see that if the chosen basis set $\{\phi\}$ is *orthogonal*, the overlap matrix $\mathbf{S}$ reduces to the identity matrix $\mathbf{I}$, and Eq. (2.41) reduces to the regular eigenvalue problem $\mathbf{Hc} = \varepsilon\mathbf{c}$.

We have a great deal of freedom with regard to choice of basis set, but the more similar the basis functions ϕ_j are to the wave function ψ we want to reconstruct, the fewer basis functions are needed for an accurate result, and thus the vector $\mathbf{c}$ is shorter. This has a beneficial effect on the resulting size of the matrix problem Eq. (2.41) while at the same time potentially causing the basis set to be non–orthogonal, and leaving us with a generalized eigenproblem to solve, rather than a regular one.

Another important choice that can be made with respect to basis sets, is the use of basis sets with *compact support*. Effectively, the constituent basis functions in $\{\phi\}$ are constructed to be *localized*, such that they are non–zero in a finite volume of space, and zero outside this local zone. This property will lead to matrices $\mathbf{H}$ and $\mathbf{S}$ that are *sparse*[1].

This localization of basis functions leads to the concept of *principal layers* in the elongated linear systems we will be studying later on when introducing the topic of two–probe systems. A set of principal layers is essentially minimally sized slices through an elongated system where each slice only interacts with its nearest neighbor slice, as seen in Fig. 2.4.

In real systems, electrons everywhere feel every other electron, irrespective of distance, however, the use of localized basis functions and the rise of

[1]A sparse matrix is one in which a large fraction of its elements are zero. This fraction may vary according to definition, but a good guideline for defining a matrix to be sparse is when it has enough non–zeros for them to be exploited for significant savings in time or storage under algorithmic manipulation.

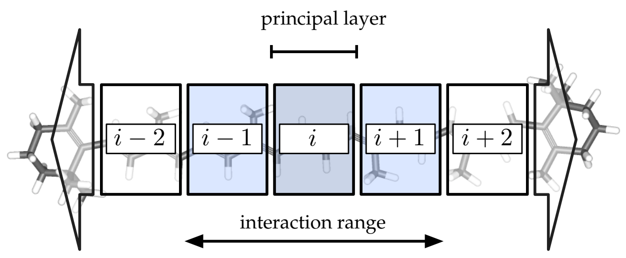


Figure 2.4: A set of principal layers in an elongated, linear system described with basis functions with compact support are minimal sized layers arranged in such a way that any layer i will only interact with its nearest neighbor layers $i-1$ and $i+1$ due to the limited range of the basis functions.

principal layers is an accepted approximation of true behavior, again due to Kohn's description of nearsightedness [16, 17]. This effect of nearsightedness will manifest itself clearly in the structure of the Hamiltonian, which we now go on to discuss.

2.3.2 Calculating the Hamiltonian

Starting with the generalized eigenproblem Eq. (2.41), and a set of basis functions $\{\phi\}$, we can easily compute the overlap matrix **S** by Eq. (2.40). However, the hamiltonian matrix **H** itself proves to be more difficult.

Looking at Eq. (2.39), we take $\hat{H}$ to be the Kohn–Sham hamiltonian from Eq. (2.30). The KS hamiltonian is composed of a regular kinetic energy term $\hat{T}(\mathbf{r})$ and the effective potential $\hat{V}_{\text{eff}}$. Of these two terms, the effective potential is the more difficult one to calculate, as it depends on the electron density $n(\mathbf{r})$ that we are trying to seek (cf. Eq. (2.26)). This problem can be overcome by solving for the effective potential using a *self–consistent* procedure. This is visualized in Fig. 2.5.

The self–consistent procedure in determining the density $n(\mathbf{r})$ that minimizes Eq. (2.23) begins with an initial guess of the density $n_{\text{guess}}(\mathbf{r})$. From this guess, we construct the effective potential $\hat{V}_{\text{eff}}(\mathbf{r})$ from equation Eq. (2.26).

The effective potential is then used in the single–electron Schrödinger equation given in Eq. (2.27), from which we can obtain single–electron wave function solutions $\psi_i(\mathbf{r})$. These solutions are then used in Eq. (2.28) in order to reconstruct the density $n'(\mathbf{r})$ in this non–interacting system. We then compare this reconstructed density $n'(\mathbf{r})$ to the original input density $n(\mathbf{r})$ and accept it as a converged density $n_{\text{scf}}(\mathbf{r})$ if they are equal to within a given tolerance.

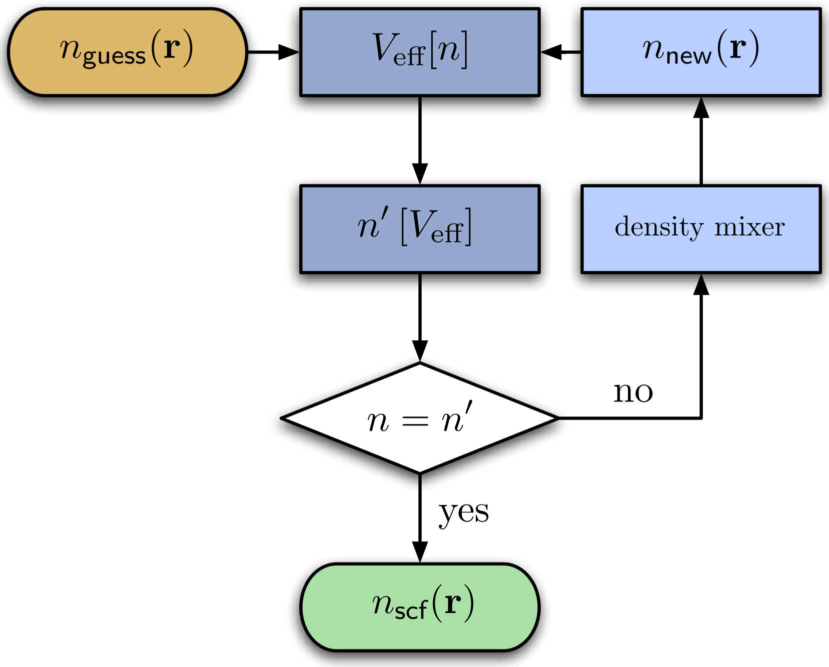

$n_{\text{guess}}(\mathbf{r})$
$V_{\text{eff}}[n]$
$n_{\text{new}}(\mathbf{r})$
$n'\,[V_{\text{eff}}]$
density mixer
$n = n'$
no
yes
$n_{\text{scf}}(\mathbf{r})$

Figure 2.5: The flow of the self–consistent procedure involved in calculating the electron density $n(\mathbf{r})$.

On the other hand, if the density $n'(\mathbf{r})$ is not considered to have converged, a new density is constructed via a density *mixing* strategy [20] in order to create a new density $n_{\text{new}}(\mathbf{r})$ that then enters the procedure from the top.

The procedure is then iterated a number of times until either convergence of the density is attained, or an upper limit of iterations is reached. If convergence is reached, a density $n_{\text{scf}}(\mathbf{r})$ is then returned which is the correct ground state density for Eq. (2.23).

2.3.3 Calculating the Effective Potential

The effective potential (cf. Eq. (2.29)) consists of three terms, each of which is described here. The external potential that each electron feels arises due to the other nuclei, and we sum up the electrostatic potential caused by each nuclei as

$$\hat{V}_{\text{ext}}(\mathbf{r}) = \sum_I \frac{Z_I}{|\mathbf{r} - \mathbf{R}_I|}. \tag{2.42}$$

This term does not depend on the density, and as such, can be calculated a single time and reused throughout the self–consistent procedure.

The Coulomb potential, or Hartree potential, is calculated by solving the Poisson equation

$$\nabla^2 \hat{V}_{\text{Hartree}}(\mathbf{r}) = -4\pi n(\mathbf{r}), \tag{2.43}$$

and tells us what the electrostatic potential is in the system given by the average location of the electrons. It has to be solved with appropriate boundary conditions, which are dictated by the type of system we are studying. In the case of isolated molecules, the boundary condition becomes one where the potential asymptotically goes to zero the further we are from the molecule. In the case of infinite bulk crystal systems, the boundary conditions become periodic. A third type of system, the *two–probe* system [21], will be treated in the next section.

Finally, the exchange–correlation term accounts for all other effects, and is here represented by a function that depends on the density and its higher derivatives.

$$\hat{V}_{\text{XC}}(\mathbf{r}) = f(n(\mathbf{r}), \nabla n(\mathbf{r}), \nabla^2 n(\mathbf{r})) \tag{2.44}$$

The exact form of f will not be described further.

2.4 Electron Transport

We now have both theory and a computational strategy in how to solve for the ground state density on a computer. However, we still have no formalism in describing and treating the case of electron flow that can happen between materials when experiencing a finite difference in their chemical potentials. We now introduce some theory and concepts that will help us do so, and set the stage for the numerical techniques presented later in this thesis.

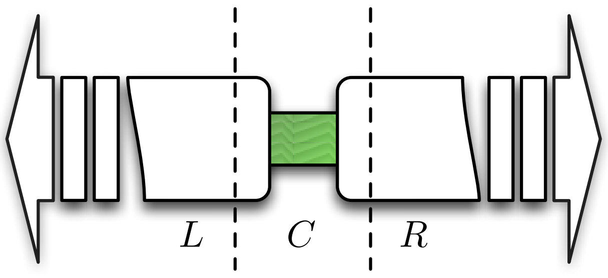


Figure 2.6: The two–probe system is composed of three distinct regions: the left (L) semi–infinite electrode, the right (R) semi–infinite electrode, and a finite central (C) region, that may contain some portion of the electrodes in order to ease the transition between central region properties and those of the electrodes.

Ultimately, we will be describing the *two–probe* system, which is illustrated in Fig. 2.6. One can imagine two electrodes, or probes, with a certain voltage potential difference between them, i.e. bias, connecting to a device or molecule from either side. On the nanoscopic scales for which we are seeking to understand transmission, these electrodes can be well described as being two semi–infinite electrodes that carry charge to and from the system and have their own chemical and bias potentials. This two–probe model can fit many situations of interest where we are interested in the electron transport taking place across a molecule, device, or an interface.

But why do we need to resort to all this trouble in describing electron transmission? We take our departure from classical physics, which has treated the phenomenon of electrical behavior on the macroscopic scale since the discovery of electricity.

2.4.1 Classical Transmission

From classical physics we have that the conductance of a macroscopic conductor is well–described by Ohm's law [22]:

$$G = \sigma \frac{A}{L}. \tag{2.45}$$

Ohm's law relates the conductance G to a single material property of the conductor, namely its conductivity σ. Furthermore, two geometric properties of the conductor affect conductance and they are the conductor's cross–sectional area A with respect to the direction of electric current, and its length L in this direction.

However, as technology has advanced, we find ourselves dealing with conductors of ever decreasing dimensions, such as those encountered in the field of integrated circuit design, where it is common to discuss conductors measured in *nanometers*[1], and at scales where individual atoms become resolved. It has been shown that the Ohmic[2] behavior of smaller and smaller conductors breaks down in the face of quantum mechanical effects experienced at these length scales. We now present a view of the world at the nanoscopic scale for the two probe system in Fig. 2.6.

2.4.2 The Landauer–Büttiker Picture

The description that we turn to for describing the behaviour exhibited by tiny devices or molecules under study, is known as the Landauer–Büttiker (LB) approach to quantum transport [23, 24].

One of the important concepts in the LB picture of things is that of *reservoirs*. From Fig. 2.6 and [23], we see that we are considering two semi–infinite electrodes, or reservoirs of electrons, which are joined by a tiny constriction consisting of a small device or molecule that we wish to study the conductance of.

The left and right reservoirs can be characterized by their electrochemical potentials μ_L and μ_R, respectively, with a resulting *bias* potential V_{bias} being interpreted as their difference and applied across the central device or molecule. This is visualized in Fig. 2.7.

$$\mu_L - \mu_R = V_{\text{bias}} \tag{2.46}$$

[1]one nanometer = 1 nm = 10^{-9} m

[2]by *Ohmic*, it is implied that the conductor adheres to Ohm's law given in Eq. (2.45)

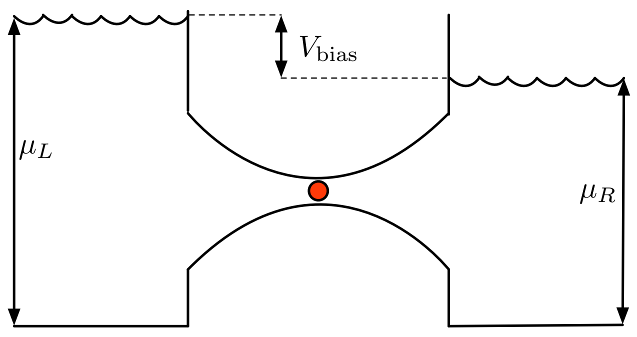


Figure 2.7: The Landauer–Büttiker picture of a device or molecule, represented by the red circle, attached between two semi–infinite reservoirs with differing chemical potentials. The difference in chemical potentials is illustrated as V_{bias}.

If there is no net bias potential, the system is said to be in *equilibrium*, and no current flows, although the presence of the reservoirs may still affect the electron density of the central device or molecule. If the electrochemical potentials are different, such that a net *bias* voltage exists, then the system is said to be in *non–equilibrium*, and a current will flow. The reservoirs, being infinite in size, will never empty or fill, and thus a steady–state of net electron transport will take place over the central device or molecule. An important condition in the LB picture is that there be no inelastic scattering of electrons within the central region, such as electron–phonon interaction

The reservoirs themselves are said to be in local equilibrium, in the sense that even though the system as a whole might not be in equilibrium, the reservoirs are seen to be in isolation. This is because the current flowing in them is distributed among a large number of states, and thus the current density itself is negligible. This is why we can regard the reservoirs as being unchanging and the simple measure of electrochemical potential for the reservoirs suffices as a basic property.

Another condition in the LB approach is that the wide, semi–infinite reservoirs connect *smoothly* to the central region, in the sense that electrons can exit the central region without reflection. Although Landauer insisted that this be the case when applying his formulas for conductance, the linear response of the central region to an applied external field is free of this geometric assumption and yields the same expressions for conductance.

2.4.3 Non–Equilibrium Green's Function Formalism

The attraction of the non–equilibrium Green's function (NEGF) formalism for dealing with the non–equilibrium case is that it is usually much easier to calculate the Green's function than to solve the eigenvalue problem presented by the Schrödinger equation. Furthermore, many properties of the system being studied can still be derived from the Green's function.

First, we define the Green's function:

$$(\varepsilon \mathbf{S} - \mathbf{H})\mathbf{G}(\varepsilon) = \mathbf{I} \tag{2.47}$$

where the overlap matrix $\mathbf{S}$ arises instead of the identity matrix $\mathbf{I}$ since we may have employed a non–orthogonal basis set in the expansion of our wave functions ψ_i.

The Green's function matrix relates how a point disturbance in the system, being a perturbation of one of the coefficients $c_{\mu i}$ in the expansion of the wave function, affects all other coefficients, and thus the whole of the rest of the system.

There are two types of Green's functions: the *retarded* and *advanced* Green's functions, which we name $\mathbf{G}^r$ and $\mathbf{G}^a$. The difference lies in where the disturbance that the Green's function represents originates from. In the case where the point disturbance spreads from a single point in the system until the rest of the system is affected, then we are dealing with the retarded Green's function. If the disturbance originates in the whole rest of the system which then collects itself at a single point, then we have the advanced Green's function.

It can be likened to throwing a rock into a pond. The rock hitting the water will create a point disturbance, which will then spread outwards in the form of circular waves until the whole rest of the system, i.e. the pond, is affected. This would be the retarded Green's function. However, in the mathematical formulation, an equally correct solution exists, which is that the circular waves *originate* from all around in the system, only to converge concentrically on a single point, much as if time was running backwards. This would correspond to the advanced Green's function.

It is the retarded Green's function which we desire, and in order for the solution of Eq. (2.47) to yield the correct result, we perturb the energy ε slightly in order to ensure this:

$$((\varepsilon + \hat{\imath}\delta^+)\mathbf{S} - \mathbf{H})\mathbf{G}^r(\varepsilon) = \mathbf{I} \tag{2.48}$$

$$((\varepsilon - \hat{\imath}\delta^+)\mathbf{S} - \mathbf{H})\mathbf{G}^a(\varepsilon) = \mathbf{I} \tag{2.49}$$

where δ^+ is an infinitesimal positive value, and $\hat{\imath}$ is the complex unit. From now on, we consider ε to include this perturbation when necessary.

2.4.4 Making the Infinite Finite

So far, we have been dealing with a system of infinite size. To be exact, we have been looking at placing a molecule or device in a finite–sized central region C, and connecting it to two semi–infinite electrodes on either side, where either electrode extends to infinity on its corresponding side of the device (cf. Fig. 2.6). This leads to the following infinite hamiltonian,

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}^L & \mathbf{V}^{LC} & \\ \mathbf{V}^{CL} & \mathbf{H}^C & \mathbf{V}^{CR} \\ & \mathbf{V}^{RC} & \mathbf{H}^R \end{pmatrix}. \tag{2.50}$$

where

$$\mathbf{H}^L = \begin{pmatrix} \ddots & \ddots & & & \\ \ddots & \mathbf{h}^L_{33} & \mathbf{v}^L_{32} & & \\ & \mathbf{v}^L_{23} & \mathbf{h}^L_{22} & \mathbf{v}^L_{21} & \\ & & \mathbf{v}^L_{12} & \mathbf{h}^L_{11} & \mathbf{v}^L_{10} \\ & & & \mathbf{v}^L_{01} & \mathbf{h}^L_{00} \end{pmatrix}, \tag{2.51}$$

$$\mathbf{H}^R = \begin{pmatrix} \mathbf{h}^R_{00} & \mathbf{v}^R_{01} & & & \\ \mathbf{v}^R_{10} & \mathbf{h}^R_{11} & \mathbf{v}^R_{12} & & \\ & \mathbf{v}^R_{21} & \mathbf{h}^R_{22} & \mathbf{v}^R_{23} & \\ & & \mathbf{v}^R_{32} & \mathbf{h}^R_{33} & \ddots \\ & & & \ddots & \ddots \end{pmatrix}, \tag{2.52}$$

and generally $\mathbf{h}^L_{ii} = \mathbf{h}^L_{jj}$ and $\mathbf{h}^R_{ii} = \mathbf{h}^R_{jj}$ for any $i, j \geq 0$ since we regard the electrodes to be an infinite, repeating bulk structure. Furthermore, $\mathbf{v}^L_{ij} = \mathbf{v}^L_{kl}$ and $\mathbf{v}^R_{ij} = \mathbf{v}^R_{kl}$ for any $i = j \pm 1 \geq 0 \wedge k = l \pm 1 \geq 0$.

With regard to the coupling of the central region to the reservoirs,

$$\mathbf{V}^{LC} = \begin{pmatrix} & & \\ & & \\ & & \\ \mathbf{v}^L_{0c} & & \end{pmatrix}, \text{ and } \mathbf{V}^{CL} = \begin{pmatrix} & & \mathbf{v}^L_{c0} \\ & & \\ & & \\ & & \end{pmatrix}. \tag{2.53}$$

Likewise with the coupling to the right reservoir, we have

$$\mathbf{V}^{CR} = \begin{pmatrix} & & \\ & & \\ & & \\ \mathbf{v}^R_{0c} & & \end{pmatrix}, \text{ and } \mathbf{V}^{RC} = \begin{pmatrix} & & \mathbf{v}^R_{c0} \\ & & \\ & & \\ & & \end{pmatrix}. \tag{2.54}$$

In order to solve the infinitely sized matrix equation Eq. (2.41) involving the infinite hamiltonian Eq. (2.50), we need to transform it, and thus its related discretized hamiltonian, into a finite system. This is done by transforming the infinitely extended electrode portions of the matrix $\mathbf{H}$, namely $\mathbf{H}^L$ and $\mathbf{H}^R$, into an equivalent pair of finite–sized *self–energy* contribution matrices $\mathbf{\Sigma}^L$ and $\mathbf{\Sigma}^R$ for the left and right electrodes, respectively.

Various methods exist in order to do this, but in this thesis, the self–energies have been developed via two different methodologies : the iterative scheme [25], and via scattering states [2][1]. Not only do these methods yield a finite hamiltonian for us to consider, they perform this transformation theoretically without approximation. The self–energies $\mathbf{\Sigma}^{L,R}$ returned are of equal size to $\mathbf{H}^C$, but are only nonzero in the corners, which will be detailed in Sec. 2.6.

Thus our original infinite sized hamiltonian will yield quantitatively and qualitatively identical results to our finite system:

$$\mathbf{H} \equiv \mathbf{H}^C + \mathbf{\Sigma}^L + \mathbf{\Sigma}^R \tag{2.55}$$

2.5 Computational Approach

The full computational approach we consider in this thesis for the determination of ground state conditions and transmission in nanoscopic systems is that of NEGF via a self–consistent field approach [21], much in the same spirit as described earlier for calculating the effective potential $\hat{V}_{\text{eff}}(\mathbf{r})$.

We present a rough program flow in Fig. 2.8, where the algorithm takes as its starting point the light green boxes at the top, where we deal with determining the characteristics of the semi–infinite electrodes and their effect on the central region. From a calculation of the effective potential for the bulk material that represent the right and left electrodes, we then proceed to calculate the self–energies the electrodes represent, and from here we can move towards the actual self–consistent loop in the NEGF routine.

Here we begin with an educated guess as to what the electron density in the system is, $n_{\text{guess}}(\mathbf{r})$, and calculate the elements of the hamiltonian matrix $\mathbf{H}_{ij}$. This step implicitly also involves calculating the effective potential.

Once we have the hamiltonian $\mathbf{H}$, we calculate the (retarded) Green's function $\mathbf{G}(\varepsilon)$ for a certain energy ε that may or may not have the imaginary perturbation as detailed in Eq. (2.49). From this Green's function matrix, we can then construct a new density based on $n'(\mathbf{G})$, which we can compare to our original

[1]Included as appendix B.

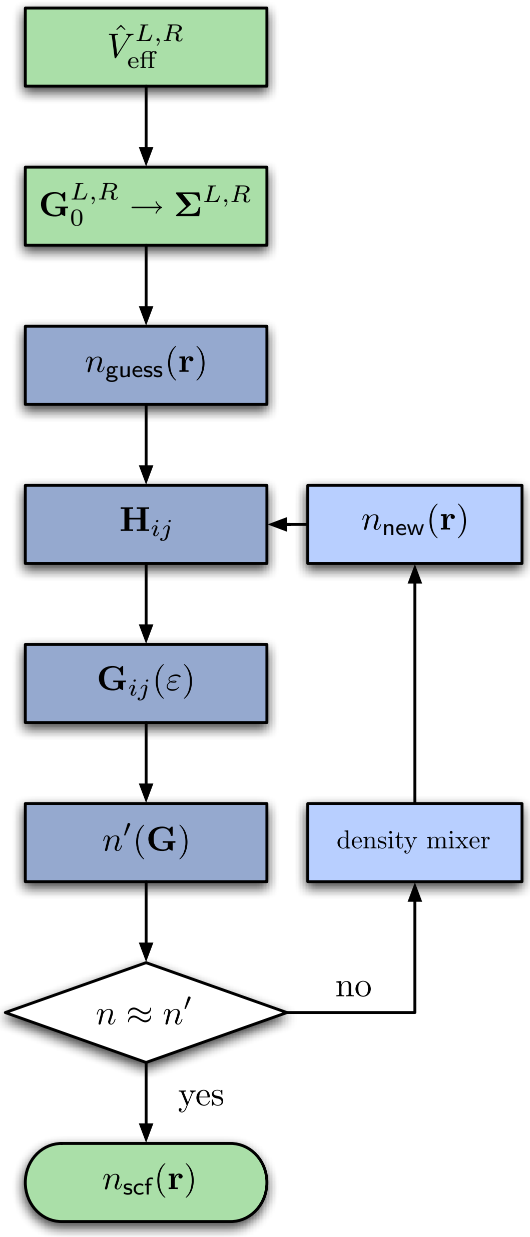

$\hat{V}^{L,R}_{\text{eff}}$
$\mathbf{G}^{L,R}_0 \rightarrow \mathbf{\Sigma}^{L,R}$
$n_{\text{guess}}(\mathbf{r})$
$\mathbf{H}_{ij}$
$n_{\text{new}}(\mathbf{r})$
$\mathbf{G}_{ij}(\varepsilon)$
$n'(\mathbf{G})$
density mixer
$n \approx n'$
no
yes
$n_{\text{scf}}(\mathbf{r})$

Figure 2.8: The flow of the self–consistent procedure involved in calculating the current over a two–probe system out of equilibrium.

input density. If it is the same to within a given tolerance, we terminate with a converged density $n_{\mathsf{scf}}(\mathbf{r})$, otherwise a mixing strategy is called on in order to construct a new density that should bring us closer to the true ground state density.

The important fact to note from Fig. 2.8 is that the step in calculating the Green's function $\mathbf{G}$ has traditionally been the step involving the highest complexity, being cubic in the order of electrons in the system. Furthermore, it is also only necessary to determine the portion of $\mathbf{G}$ for which the overlap matrix $\mathbf{S}$ is nonzero. This leads us to only need to determine $\mathbf{G}$ for a certain block tridiagonal structure described next in Sec. 2.6. Improving this step, which involves the inversion of a large matrix of a particular structure, is the goal of this thesis.

Another characteristic to note is that a lot of weak parallelism is able to be exploited from the program flow of NEGF. Typically, to construct the new density, a large number of Green's function matrices need to be evaluated for a range of complex energy points [26] which can be done independently. Furthermore, the calculation of the hamiltonian matrix elements also lends itself to parallelization. This thesis focuses on the harder problem of parallelizing the inversion of the matrix that leads to the Green's function matrix, $\mathbf{G}$, potentially multiplying the number of processes that can effectively work on a single problem.

2.6 The Block Tridiagonal Matrix

As it has been shown in the previous section, the Green's function matrix $\mathbf{G}$ is a central value in the computational DFT procedure. Furthermore, being involved in the most algorithmically time consuming steps, with respect to algorithmic complexity, it is of great interest to apply the steps using efficient algorithms, both serial and in parallel.

We focus on the following block tridiagonal matrix

$$\mathbf{A} = \varepsilon\mathbf{S} - \mathbf{H}^C - \boldsymbol{\Sigma}^L - \boldsymbol{\Sigma}^R \tag{2.56}$$

where the Green's function matrix we seek is $\mathbf{G} = \mathbf{A}^{-1}$.

2.6.1 Structure

The structure of $\mathbf{A}$, subdivided into its commensurate sub–block element matrices is resolved as

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} & & & & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \mathbf{a}_{n-1,n-2} & \mathbf{a}_{n-1,n-1} & \mathbf{a}_{n-1,n} \\ & & & & \mathbf{a}_{n,n-1} & \mathbf{a}_{nn} \end{pmatrix}, . \tag{2.57}$$

where we wish to note the notation of lower case bold letters $\mathbf{a}_{ij}$ denoting matrices as elements of their upper case bold letter $\mathbf{A}$ block matrix. Furthermore, the (i,j) indexing of these sub–block elements will correspond to their location with regards to the rows and columns of block matrices in $\mathbf{A}$

The self energy matrices $\mathbf{\Sigma}^{L,R}$, as previously mentioned, are only nonzero in their corners, and usually overlap the locations corresponding to $\mathbf{a}_{11}$ and $\mathbf{a}_{nn}$, respectively:

$$\mathbf{\Sigma}^{L} = \begin{pmatrix} \boldsymbol{\sigma}^{L}_{11} & & \\ & & \\ & & \end{pmatrix} \qquad \mathbf{\Sigma}^{R} = \begin{pmatrix} & & \\ & & \\ & & \boldsymbol{\sigma}^{R}_{nn} \end{pmatrix}. \tag{2.58}$$

The reason this is the case is usually because the central region of the two probe system may incorporate several principal layers of the electrodes as a buffer to ease the transition of the differing electronic state of a device in the middle and the infinite bulk crystals surrounding it. The self energy matrices $\boldsymbol{\sigma}^{L}_{11}$ and $\boldsymbol{\sigma}^{R}_{nn}$ themselves correspond to the size of $\mathbf{h}^{L}_{00}$ and $\mathbf{h}^{R}_{00}$, respectively, and represent the effect of the semi–infinite electrodes exactly [22].

2.6.2 Properties

Based on the properties of the overlap matrix $\mathbf{S}$, the hamiltonian matrix $\mathbf{H}$, the self–energy matrices $\mathbf{\Sigma}^{L,R}$, and the energy ε, we can perhaps ascertain some properties of $\mathbf{A}$. The hamiltonian $\mathbf{H}$ (cf. Eq. (2.39)) is known to be both real and Hermitian, so we have

$$\mathbf{H}^{C} = \left[\mathbf{H}^{C}\right]^{\dagger} \tag{2.59}$$

$$\mathbf{H}^{C} = \left[\mathbf{H}^{C}\right]^{T} \tag{2.60}$$

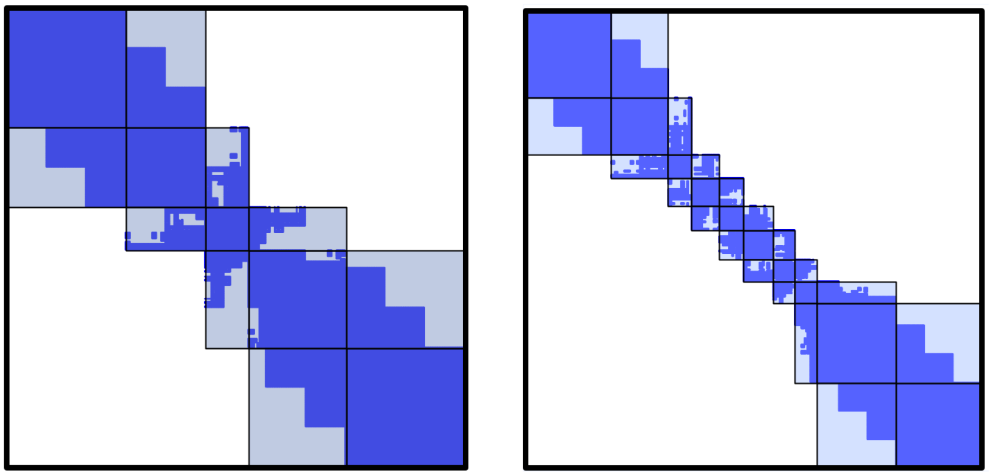


Figure 2.9: The block tridiagonal structures $\mathbf{A}$ involved in the calculation for an Au(111) electrode Di–thiol benzene (DTB) system [27] on the left and an Au(111) electrode Aviram–Ratner (AR) diode system [28] on the right. The block tridiagonal matrix for the DTB system is a 5×5 block matrix with 928×928 scalar elements, while for AR on the right we have a 10×10 block system with 1295×1295 scalar elements.

where $\mathbf{H}^T$ denotes transposition, and where $\mathbf{H}^\dagger$ denotes both transposition and complex conjugation of the elements in $\mathbf{H}$. The overlap matrix $\mathbf{S}$ (cf. Eq. (2.40)) also obeys these properties, and

$$\mathbf{S} = \mathbf{S}^\dagger \tag{2.61}$$
$$\mathbf{S} = \mathbf{S}^T \tag{2.62}$$

The self–energies $\mathbf{\Sigma}^L$ and $\mathbf{\Sigma}^R$ are mostly all–zero, except for the sub–blocks $\boldsymbol{\sigma}^L_{11}$ and $\boldsymbol{\sigma}^R_{nn}$. These blocks can generally be assumed to be complex, and moreover are only Hermitian under certain conditions[1]. In the case of modeling electron–phonon interactions or photonic effects, the self–energies may be nonzero beyond their corners as dictated by Eq. (2.58), but in this thesis we only look to model the effects of coupling of the central region with the semi–infinite electrodes. Ultimately, we assume the nontrivial portion of the self–energies to be generally dense in structure and complex.

Finally, the combination of these various matrices in constructing $\mathbf{A}$ also depends on ε. The chosen energy value can vary from being real, to being slightly perturbed as in Eq. (2.49), to being fully complex when following a

[1]A non–Hermitian self–energy can be generated through the use of a technique known as k–point sampling.

complex contour integral in the determination of a new density via the NEGF program flow.

Since ε may be real or complex, we have developed the algorithms in this thesis focusing on handling the most general case of a fully complex non–symmetric $\mathbf{A}$ with dense *square* blocks $\mathbf{a}_{ij}$ on the block tridiagonal. Special cases can arise for when $\mathbf{A}$ are symmetric and positive definite and in that case, operations involving the LU factorization of blocks can be substituted with Cholesky decompositions. This has not been implemented though, due to the easily varying nature of $\mathbf{A}$.

2.7 Mathematical Notation

In order to develop and document the algorithms and theory presented in this paper, a series of notational conveniences have been adopted in order to do so. It can be debated what notation would be more suited to convey the meaning of what is written, but a notation that was most consistent with the published work [1] and eventually [4] was chosen.

2.7.1 The Block Matrix Class

This chapter will introduce a series of algorithms on which the input is assumed to be understood as a block tridiagonal matrix $\mathbf{A}$ (cf. Eq. (2.57)). The block tridiagonal matrix $\mathbf{A}$ can be understood to belong to a class of matrices such as $\mathbb{C}^{r,s}$, but in order to specify the class of matrices which have block structure, we define $\mathbb{B}^{m,n}$ to represent the matrices comprised of $m \times n$ blocks, where all the constituent blocks have the correct sizes in terms of their neighbor blocks, i.e. all blocks in a row have the same height and all blocks in a column have the same width. Furthermore, each diagonal block is *square*, i.e. belongs to $\mathbb{C}^{k,k}$ for some integer $k > 0$.

Thus a general matrix composed of $r \times s$ scalar elements can be thought of as a *block* matrix of $m \times n$ matrix blocks,

$$\mathbb{B}^{m,n} \equiv \mathbb{C}^{r,s}, \tag{2.63}$$

where $0 < m \leq r$ and $0 < n \leq s$ and keeping in mind that all diagonal blocks are square.

2.7.2 Extracting a Block Tridiagonal Part

A useful notation we wish to introduce is one that enables the extraction of the block tridiagonal part of an arbitrary matrix $\mathbf{B}$ with respect to a given block tridiagonal structure, $\mathbf{A}$, into a block tridiagonal matrix $\mathbf{C}$. This is written as:

$$\mathbf{C} = \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{B}\right\} \tag{2.64}$$

This is only possible under certain conditions, namely that $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{r,s}$. This leads to $\mathbf{C} \in \mathbb{R}^{r,s}$ and that if $\mathbf{A} \in \mathbb{B}^{m,n}$, then $\mathbf{C} \in \mathbb{B}^{m,n}$. If $\mathbf{B}$ has any nonzero elements outside the block tridiagonal as defined by $\mathbf{A}$, they are lost.

Furthermore, for the sake of brevity in the specification of the algorithms in this chapter, we group sequential arguments that are block tridiagonal extractions in the following manner:

$$\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{B}\right\}, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{C}\right\}, \ldots, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{Z}\right\} = \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{B}, \mathbf{C}, \ldots, \mathbf{Z}\right\} \tag{2.65}$$

2.7.3 Extracting a Block Diagonal Part

Similar to notation of extracting the block tridiagonal part of a block matrix, we introduce notation that only extracts the block diagonal. This is written as:

$$\mathbf{C} = \mathrm{Diag}_{\mathbf{A}}\left\{\mathbf{B}\right\} \tag{2.66}$$

Again, this only possible under certain conditions, namely that $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{r,s}$. This leads to $\mathbf{C} \in \mathbb{R}^{r,s}$ and that if $\mathbf{A} \in \mathbb{B}^{m,n}$, then $\mathbf{C} \in \mathbb{B}^{m,n}$. If $\mathbf{B}$ has any nonzero elements outside the block diagonal as defined by $\mathbf{A}$, they are lost.

2.7.4 Extracting a Sub–block

Another useful notation we introduce enables us to extract a sub–block of a known block matrix structure. We denote this by $[\mathbf{A}]_{ij}$, by which we mean the matrix block on the (i, j)th block position in $\mathbf{A}$'s block structure. From Eq. (2.57), we know this to be

$$[\mathbf{A}]_{ij} = \mathbf{a}_{ij}. \tag{2.67}$$

This, however, extends to blocks off the block tridiagonal positions, which in $\mathbf{A}$ are generally all–zero matrices, but for other matrices may have nontrivial content, e.g. in $\mathbf{G} = \mathbf{A}^{-1}$.

Also, we assume generally that the lower case bold letter refers to a block element from its upper case block matrix counterpart, e.g. the block matrix element a to the block matrix $\mathbf{A}$, the block matrix element g to the block matrix $\mathbf{G}$, etc.

2.7.5 Augmented Matrix

The algorithms developed in this paper are explained through the concept of matrix augmentation. Thus in order to obtain the inverse of $\mathbf{A}$, we augment it with the identity matrix, $\mathbf{I}$, obtaining

$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{I} \end{array}\right] = \left(\begin{array}{cccc|cccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & & \mathbf{i}_{22} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & & \mathbf{i}_{33} & \\ & & \ddots & \ddots & \ddots & & & \ddots \end{array}\right), \tag{2.68}$$

where each diagonal block of the identity matrix, $\mathbf{i}_{ii}$ has the same square block size of the corresponding block $\mathbf{a}_{ii}$ of the matrix $\mathbf{A}$, and are themselves identity matrices. Then by a series of row operations that change the right hand side to the identity matrix $\mathbf{I}$, we can read the inverse $\mathbf{G} = \mathbf{A}^{-1}$ on the left.

$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{I} \end{array}\right] \overset{\text{row}}{\underset{\text{ops}}{\Longleftrightarrow}} \left[\begin{array}{c|c} \mathbf{I} & \mathbf{A}^{-1} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{I} & \mathbf{G} \end{array}\right] \tag{2.69}$$

2.7.6 Operation Count

In order to analyze the complexity of the algorithms we develop, we are forced to identify some basic operations we perform on our block tridiagonal matrices. In this paper, we have identified and counted three types of matrix–matrix operations, namely LU factorization, matrix–matrix multiplication and matrix–matrix addition. These operations are represented by $\mathsf{op}(\mathcal{LU})$, $\mathsf{op}(\times)$ and $\mathsf{op}(+)$, respectively.

2.7.7 Block Elements vs. Scalar Elements

As a consequence of our work dealing with the class of block matrices $\mathbb{B}^{m,n}$ we defined earlier, we will be dealing explicitly more often with operations on the matrix blocks that make up these matrices as a whole than with the basic operations that involve the scalar values making up these matrices.

Thus throughout the paper, we will imply that we are dealing with *block* Gaussian elimination and with the basic matrix–matrix operations of addition, multiplication as well as LU factorization of matrix *blocks*, rather than the scalar operations and elements understood when dealing with regular matrices, unless expressly noted. This extends to row operations on block matrices involving the updating of whole matrices at a time, rather than on rows of single scalars as usually expected when confronted with matrix row operations on regular matrices in $\mathbb{R}^{r,s}$.

2.8 Pseudocode Notation

In order to document the algorithms developed in this thesis, a customized pseudo code was used that should mostly be self explanatory to read and understand. The pseudocode is not strict in its specification, and its purpose is not to provide stringent proof of their behavior, but more for the purposes of illustrating the algorithms which are described in this thesis. This short section will attempt to clarify the notation used in the pseudocode.

2.8.1 Calling and Arguments

Algorithms themselves are indicated by the small cap font : ALGORITHM. They are called with a certain number of arguments, for which any special requirements are given by an initial **Require** statement. Any special requirements for the output or return state of the algorithm is given by an **Ensure** statement following a possible require statement and before the code body of the algorithm.

2.8.2 Assignment

Assignment of values from one variable to another is done using the assignment symbol $\leftarrow$. For example, we can write

$$\mathbf{a}_{ij} \leftarrow \mathbf{a}_{ij} + \mathbf{a}_{ii}\mathbf{a}_{ij} \tag{2.70}$$

where the original block matrix $\mathbf{a}_{ij}$ is replaced with the value represented by the expression on the right hand side of the arrow. In this case, we have $\mathbf{a}_{ij}$ being summed into by the product $\mathbf{a}_{ii}\mathbf{a}_{ij}$.

2.8.3 Arrays

One dimensional arrays are specified with curly braces, and since we only use them to store indices, we use the lower case symbol $\mathbf{i}$ for their identification.

An example array would be

$$\mathbf{i} = \{i_1, i_2, \ldots, i_n\}, \tag{2.71}$$

where we show an array of length n composed of integers, which will be the only type of content our one dimensional index arrays will contain. Appending an integer to the end of the array is done like this:

$$\mathbf{i} \curvearrowleft i_{n+1} = \{i_1, i_2, \ldots, i_n, i_{n+1}\}. \tag{2.72}$$

Obtaining the length of an array can be done using a length operator:

$$n = \textbf{length of } \mathbf{i}. \tag{2.73}$$

Extracting an element of an array at index j is done using bracket notation:

$$i_j = \mathbf{i}[j]. \tag{2.74}$$

Finally, obtaining the last element of an array can be done with the **end** keyword, as known from Matlab:

$$i_n = \mathbf{i}[\textbf{end}]. \tag{2.75}$$

2.8.4 Loops

We employ two kinds of loops in our pseudocode, namely *for* loops and *while* loops. The while loops are given by the keyword **while** followed by a test condition and will iterate until the test condition evaluates to be false. The for loops are given by the keyword **for** and are followed by initialization of the iterating variable, the keywords **down to** or **up to** , which indicate our iterating variable to be decrementing or incrementing, and finally the value for which the iterating variable is expected to terminate on reaching (inclusive). In this thesis, all increments and decrements in this fashion have a magnitude of 1. In relation to the test conditions, we define the boolean constants **true** and **false**.

2.8.5 Returning Values

Values are returned from functions via the keyword **return** followed by the value or list of values to return.

2.8.6 Process Count and Identification

Once our algorithms are called in a parallel environment, where multiple processes may be present, it will become necessary to have methods allowing us to identify each process and to know how many processes there are available. Thus, in order to obtain the number of processes available, we have the value $\mathcal{P}$. To determine the calling process' id, we use the keyword **myPID**.

In the pseudocode as well as implementation, use the fact that the processes are enumerated from 0 and upwards, such that for an environment with p processes, we have process ids being $0, 1, \ldots, p-1$, which can alternatively be labeled $p_0, p_1, \ldots, p_{\mathcal{P}-1}$.

2.8.7 Parallel Communication

With respect to communication between processes, we specify *blocking* send and receive routines **send** and **recv**. They are blocking in the sense that they first return when the recipient of the communication has acknowledged receiving the transmitted data, thus effectively blocking execution on the calling process until the transaction is complete.

Non–blocking versions are also specified, named **isend** and **irecv**, that return a boolean request value $\mathit{request}_{\text{tag}}$ with some identifying tag. This value evaluates to true if the recipient has received the data, and false otherwise. These calls return immediately, regardless of whether or not the recipient of the transmission has received and acknowledged any data.

In connection with the non–blocking transmissions, we specify a call **wait for** $\mathit{request}_{\text{tag}}$ which will pause execution until the transaction represented by $\mathit{request}_{\text{tag}}$ is complete.

2.8.8 Ownership and Distribution

With parallelism comes distribution of memory among the processors, and as we will be distributing ownership of the block tridiagonal matrix $\mathbf{A}$ and its related LU factors $\mathbf{L}$, $\mathbf{U}$ and Green's function $\mathbf{G}$ in the same manner, each process will need to know its domain of ownership. Due to the horizontal stripe characteristic our implemented ownership regions will exhibit, each process may need to only know the indices of the upper most and lower most rows it owns. These are accessible via the values *top* and *bot*, respectively.

We also enable the ability to explicitly determine if the calling process owns a certain row i by calling **row** i **is mine**, which returns true if the ith row is owned by the calling process, and false otherwise. We can negate this expression by calling **row i is not mine**.

Another ownership issue we address in the pseudocode is a method of determining on which process a certain block row of a block matrix resides. This is accomplished by calling **owner of row i**, in order to determine what process owns the ith row. Similarly, we have **owner of $\mathbf{a}_{ij}$**, that enables us to determine what process owns the block $\mathbf{a}_{ij}$ of a distributed block matrix $\mathbf{A}$.

As data is distributed and occasionally needs sharing among processes, a routine is facilitated in order to determine if a certain process owns a certain nontrivial block of a block matrix that it might need for further calculations. This is provided by the $\mathbf{a}_{ij}$ **exists** call, that returns a boolean value for true if $\mathbf{a}_{ij}$ exists on the calling process, and false otherwise.

2.8.9 Inverse Blocks

A number of matrix blocks are listed in the pseudocode as being explicit inverses of some other block, e.g. $\mathbf{g}_{kk} = \mathbf{a}_{kk}^{-1}$ or are involved with multiplication with these inverses. The calculation of these "inverses" happens through partial pivoted LU factorization as provided by standard libraries such as LAPACK[1]. These block inverses are not calculated explicitly, and are preserved as an LU decomposition

$$\mathbf{a} = \mathbf{lu}. \tag{2.76}$$

When we need the result $\mathbf{x} = \mathbf{a}^{-1}\mathbf{b}$ of a multiplication of an inverse block with another block, we have (omitting pivoting)

$$\begin{aligned} \mathbf{ax} &= \mathbf{b} \\ \mathbf{lux} &= \mathbf{b} \\ \mathbf{ux} &= \mathbf{l}^{-1}\mathbf{b} \\ \mathbf{x} &= \mathbf{u}^{-1}\mathbf{l}^{-1}\mathbf{b} \end{aligned} \tag{2.77}$$

which can be handled by e.g. the LAPACK routine ZGESV, that handles calls to ZGETRF for LU factorization of $\mathbf{a}$, ZGETRS for performing the triangular back solves via the BLAS[2] routine ZTRSM, and a column swapping routine in order to account for pivoting. The overall cost of this is counted as a single LU factorization and what accounts as a single matrix multiplication. The routine

[1]Available from `http://netlib.org/lapack`.

[2]Available from `http://netlib.org/blas`.

ZGESV comes in handy for the calculation of certain upper LU factors for $\mathbf{A}$ and an expression may be

$$\begin{aligned} \mathbf{ax} &= \mathbf{b} \\ \mathbf{a}_{ii}\mathbf{u}_{ij} &= -\mathbf{a}_{ij} \\ \mathbf{u}_{ij} &= -\mathbf{a}_{ii}^{-1}\mathbf{a}_{ij} \end{aligned} \tag{2.78}$$

however, in the case of lower LU factors

$$\begin{aligned} \mathbf{xa} &= \mathbf{b} \\ \mathbf{l}_{ji}\mathbf{a}_{ii} &= -\mathbf{a}_{ji} \\ \mathbf{l}_{ji} &= -\mathbf{a}_{ji}\mathbf{a}_{ii}^{-1} \end{aligned} \tag{2.79}$$

LAPACK does not provide a routine for handling this reverse case, and a replacement was coded in order to handle this, using the factorization routine ZGETRF, BLAS's own ZTRSM triangular back solve routines appropriately, as well as a row permutation routine to account for pivoting. As for the case when we need to determine the explicit inverse, LAPACK provides the routine ZGETRI that is equivalent to solving Eq. (2.77) where $\mathbf{b} = \mathbf{i}$.

Finally, as the value $\mathbf{a}_{ii}^{-1}$ may need sharing among processes during parallel computation, either the pivoting array and LU factorization may be sent, or the original $\mathbf{a}_{ii}$ may be sent, and then the LU factorization is performed as needed on either involved process. Ideally the LU factorization may be sent, but sending just $\mathbf{a}_{ii}$ may prove itself easier to code for.

Chapter 3

Serial Algorithms

$$2 + 2 = 5$$

(for extremely large values of 2)

Per Christian Hansen's t-shirt

3.1 Block Gaussian Elimination

One of the classical methods for solving a system of linear equations, and under which we can determine the inverse of a matrix, is the method known as Gaussian elimination. Where the original Gaussian elimination algorithm is designed to work on general nonsingular matrices A of no particular internal structure having scalars a_{ij} as elements, we adapt it to our block tridiagonal problem $\mathbf{A}$ from Eq. (2.57) by converting all scalars in the algorithm to matrix blocks $\mathbf{a}_{ij}$, and changing operations such as division into ones incorporating the inversion of matrix blocks. Furthermore, as the matrix $\mathbf{A}$ has no block elements outside the block tridiagonal, we can take this into account for an improved Gaussian elimination algorithm.

3.1.1 Description

Block Gaussian elimination, in this paper, comes in two varieties. A variety that proceed downwards, eliminating the subdiagonal elements of $\mathbf{A}$, and a variety that proceeds upwards, eliminating the superdiagonal elements of $\mathbf{A}$. We first begin by describing the downwards eliminating variety, which is characterized with the superscript ^{L} since the elimination procedure proceeds from the upper *left* and *down* towards the lower right.

A block Gaussian elimination step is performed on the matrix given in Eq. (2.68) by multiplying the first block row by the matrix $\mathbf{c}_1^L = -\mathbf{a}_{21}\mathbf{a}_{11}^{-1}$ and subsequently adding it to the second block row. This produces a zero block in the $(2,1)$ position:

$$\left(\begin{array}{cccc|cccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{a}_{21}+\mathbf{c}_1^L\mathbf{a}_{11} & \mathbf{a}_{22}+\mathbf{c}_1^L\mathbf{a}_{12} & \mathbf{a}_{23} & & \mathbf{c}_1^L\mathbf{i}_{11} & \mathbf{i}_{22} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & & \mathbf{i}_{33} & \\ & & \ddots & \ddots\ \ddots & & & & \ddots \end{array}\right) =$$

$$\left(\begin{array}{cccc|cccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{0} & \mathbf{a}_{22}-\mathbf{a}_{21}\mathbf{a}_{11}^{-1}\mathbf{a}_{12} & \mathbf{a}_{23} & & \mathbf{a}_{21}\mathbf{a}_{11}^{-1}\mathbf{i}_{11} & \mathbf{i}_{22} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & & \mathbf{i}_{33} & \\ & & \ddots & \ddots\ \ddots & & & & \ddots \end{array}\right).$$

Next, a block Gaussian elimination step is performed by multiplying the second row by the factor $\mathbf{c}_2^L = -\mathbf{a}_{32}(\mathbf{a}_{22} - \mathbf{a}_{21}\mathbf{a}_{11}^{-1}\mathbf{a}_{12})^{-1}$ and then adding it to the third row. This produces a zero block in the $(3,2)$ position. A recursive routine that will complete a full downward block Gaussian elimination is now defined:

$$\begin{array}{rcl rcl}
\mathbf{d}_{11}^L &=& \mathbf{a}_{11} & \mathbf{c}_1^L &=& -\mathbf{a}_{21}(\mathbf{d}_{11}^L)^{-1} \\
\mathbf{d}_{22}^L &=& \mathbf{a}_{22}-\mathbf{a}_{21}(\mathbf{d}_{11}^L)^{-1}\mathbf{a}_{12} & \mathbf{c}_2^L &=& -\mathbf{a}_{32}(\mathbf{d}_{22}^L)^{-1} \\
\mathbf{d}_{33}^L &=& \mathbf{a}_{33}-\mathbf{a}_{32}(\mathbf{d}_{22}^L)^{-1}\mathbf{a}_{23} & \mathbf{c}_3^L &=& -\mathbf{a}_{43}(\mathbf{d}_{33}^L)^{-1} \\
&\vdots& & &\vdots& \\
\mathbf{d}_{ii}^L &=& \mathbf{a}_{ii}-\mathbf{a}_{i,i-1}(\mathbf{d}_{i-1,i-1}^L)^{-1}\mathbf{a}_{i-1,i} & \mathbf{c}_i^L &=& -\mathbf{a}_{i+1,i}(\mathbf{d}_{ii}^L)^{-1} \\
&\vdots& & &\vdots& \\
\mathbf{d}_{nn}^L &=& \mathbf{a}_{nn}-\mathbf{a}_{n,n-1}(\mathbf{d}_{n-1,n-1}^L)^{-1}\mathbf{a}_{n-1,n} & \mathbf{c}_{n-1}^L &=& -\mathbf{a}_{n,n-1}(\mathbf{d}_{n-1,n-1}^L)^{-1}.
\end{array}$$

The matrices $\mathbf{d}_{ii}^L$ are the diagonal blocks of the resulting matrix on the left. It can be seen that each diagonal block is calculated from the following relation:

$$\mathbf{d}_{ii}^L = \mathbf{a}_{ii} + \mathbf{c}_{i-1}^L\mathbf{a}_{i-1,i}, \quad \text{where } i = 2,3,\ldots,n \text{ and } \mathbf{d}_{11}^L = \mathbf{a}_{11}, \tag{3.1}$$

and each row multiplication factor is:

$$\mathbf{c}_i^L = -\mathbf{a}_{i+1,i}\left(\mathbf{d}_{ii}^L\right)^{-1}, \quad \text{where } i = 1,2,\ldots,n-1. \tag{3.2}$$

The similar upward procedure is characterized with the superscript ^{R} since the elimination procedure moves from the lower *right* and *up* towards the upper left of $\mathbf{A}$. The derivation of the upwards recursive expressions follows that of the downwards elimination. Each diagonal block can be calculated from the following relation:

$$\mathbf{d}_{ii}^R = \mathbf{a}_{ii} + \mathbf{c}_{i+1}^R \mathbf{a}_{i+1,i}, \quad \text{where } i = n-1, \dots, 2, 1 \text{ and } \mathbf{d}_{nn}^R = \mathbf{a}_{nn}, \tag{3.3}$$

and each row multiplication factor is:

$$\mathbf{c}_i^R = -\mathbf{a}_{i-1,i} \left(\mathbf{d}_{ii}^R\right)^{-1}, \quad \text{where } i = n, \dots, 3, 2. \tag{3.4}$$

After a complete downward or upward block Gaussian elimination sweep, the augmented matrices, named $[\mathbf{D}^L \mid \mathbf{J}^L]$ and $[\mathbf{D}^R \mid \mathbf{J}^R]$ respectively, will look as follows where the matrices $\mathbf{J}^L$ and $\mathbf{J}^R$ are lower and upper block triangular, respectively:

$$[\mathbf{D}^L \mid \mathbf{J}^L] = \left(\begin{array}{cccccc|cccc} \mathbf{d}_{11}^L & \mathbf{a}_{12} & & & & & \mathbf{i}_{11} & & & \\ \mathbf{0} & \mathbf{d}_{22}^L & \mathbf{a}_{23} & & & & \mathbf{c}_1^L\mathbf{i}_{11} & \mathbf{i}_{22} & & \\ & \mathbf{0} & \mathbf{d}_{33}^L & \mathbf{a}_{34} & & & \mathbf{c}_{2,1}^L\mathbf{i}_{11} & \mathbf{c}_2^L\mathbf{i}_{22} & \mathbf{i}_{33} & \\ & & \ddots & \ddots & \ddots & & \vdots & \vdots & & \ddots \end{array}\right), \tag{3.5}$$

$$[\mathbf{D}^R \mid \mathbf{J}^R] = \left(\begin{array}{cccccc|cccc} \mathbf{d}_{11}^R & \mathbf{0} & & & & & \mathbf{i}_{11} & \mathbf{c}_2^R\mathbf{i}_{22} & \mathbf{c}_{2,3}^R\mathbf{i}_{33} & \dots \\ \mathbf{a}_{21} & \mathbf{d}_{22}^R & \mathbf{0} & & & & & \mathbf{i}_{22} & \mathbf{c}_3^R\mathbf{i}_{33} & \dots \\ & \mathbf{a}_{32} & \mathbf{d}_{33}^R & \mathbf{0} & & & & & \mathbf{i}_{33} & \dots \\ & & \ddots & \ddots & \ddots & & & & & \ddots \end{array}\right). \tag{3.6}$$

We can see that $\mathbf{D}^L$ and $\mathbf{D}^R$ are the block row echelon forms of $\mathbf{A}$ after a full downward or upward block Gaussian elimination sweep, respectively. Furthermore, the following notation was introduced:

$$\begin{array}{rcl} \mathbf{c}_i^R\mathbf{c}_{i+1}^R \cdots \mathbf{c}_j^R & = & \mathbf{c}_{i,i+1,\dots,j}^R \text{ where } i < j \\ \mathbf{c}_i^L\mathbf{c}_{i-1}^L \cdots \mathbf{c}_j^L & = & \mathbf{c}_{i,i-1,\dots,j}^L \text{ where } i > j \end{array} \tag{3.7}$$

In this thesis, we will present figures that illustrate the execution of various algorithms, in order to visualize the work the algorithms perform. For these figures, we present in Fig. 3.1 a legend that will hold generally, unless otherwise explicitly redefined in a given figure. The first such figure is given in Fig. 3.2 which showcases the downwards Gaussian elimination sweep recently discussed in two forms: a *full* form as is the case for what has just been discussed, and a *block tridiagonal* form which will be discussed later.

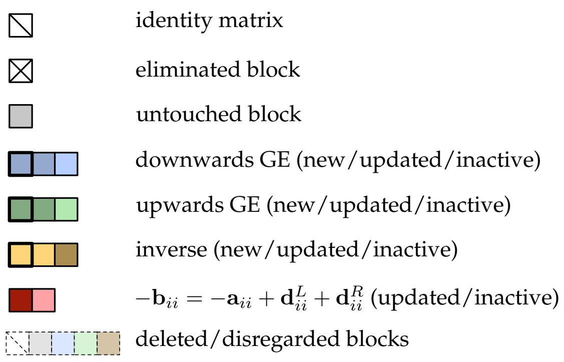

identity matrix
eliminated block
untouched block
downwards GE (new/updated/inactive)
upwards GE (new/updated/inactive)
inverse (new/updated/inactive)
$-\mathbf{b}_{ii} = -\mathbf{a}_{ii} + \mathbf{d}^L_{ii} + \mathbf{d}^R_{ii}$ (updated/inactive)
deleted/disregarded blocks

Figure 3.1: Legend for the types of matrix blocks present in the execution diagrams of the algorithms in this thesis, unless otherwise explicitly provided. The red blocks concern a matrix **B**, presented in later in Sec. 3.2.

Assuming we have performed a full downwards elimination ending with Eq. (3.5), we then proceed to back solve and start constructing the inverse $\mathbf{G} = \mathbf{A}^{-1}$. We begin with the block row echelon form Eq. (3.5) written out to include the lower rows,

$$[\mathbf{D}^L \mid \mathbf{J}^L] = \left(\begin{array}{ccccc|ccccc}
\mathbf{d}^L_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & \\
\mathbf{0} & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & \mathbf{c}^L_1 & \mathbf{i}_{22} & & & \\
 & \ddots & \ddots & \ddots & & \vdots & \vdots & \ddots & & \\
 & & \mathbf{0} & \mathbf{d}^L_{n-1,n-1} & \mathbf{a}_{n-1,n} & \mathbf{c}^L_{n-2,\dots,1} & \mathbf{c}^L_{n-2,\dots,2} & \cdots & \mathbf{i}_{n-1,n-1} & \\
 & & & \mathbf{0} & \mathbf{d}^L_{nn} & \mathbf{c}^L_{n-1,\dots,1} & \mathbf{c}^L_{n-1,\dots,2} & \cdots & \mathbf{c}^L_{n-1} & \mathbf{i}_{nn}
\end{array}\right), \tag{3.8}$$

and where multiplication by the identity matrices $\mathbf{i}_{ii}$ in $\mathbf{J}^L$ has been omitted. We can then immediately solve for the lower block row of the Green's function matrix **G** by inverting $\mathbf{d}^L_{nn}$ and multiplying it across the lower row in the augmented matrix Eq. (3.8) yielding:

$$\left(\begin{array}{ccccc|ccccc}
\mathbf{d}^L_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & \\
\mathbf{0} & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & \mathbf{c}^L_1 & \mathbf{i}_{22} & & & \\
 & \ddots & \ddots & \ddots & & \vdots & \vdots & \ddots & & \\
 & & \mathbf{0} & \mathbf{d}^L_{n-1,n-1} & \mathbf{a}_{n-1,n} & \mathbf{c}^L_{n-2,\dots,1} & \mathbf{c}^L_{n-2,\dots,2} & \cdots & \mathbf{i}_{n-1,n-1} & \\
 & & & \mathbf{0} & \mathbf{i}_{nn} & \mathbf{g}_{n1} & \mathbf{g}_{n2} & \cdots & \mathbf{g}_{n,n-1} & \mathbf{g}_{nn}
\end{array}\right), \tag{3.9}$$

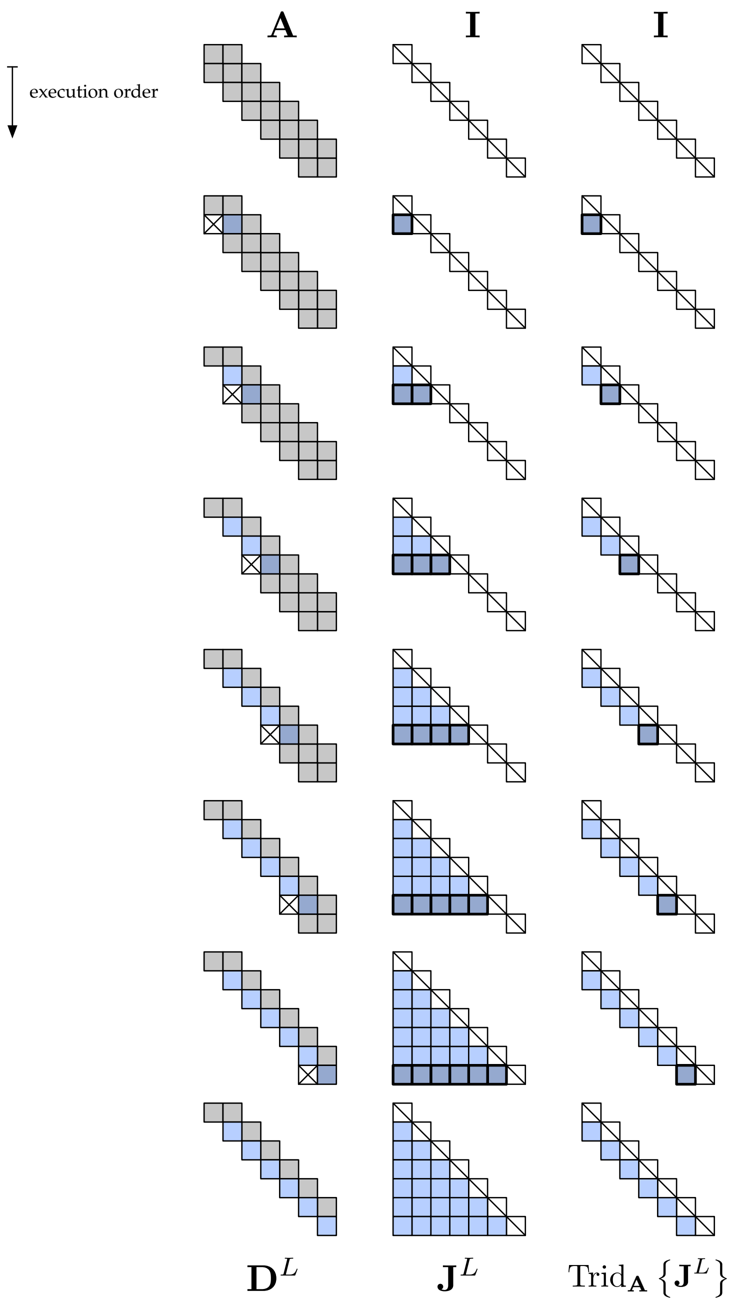


Figure 3.2: This figure presents the effect of a downwards Gaussian elimination sweep on an example 7×7 block tridiagonal matrix $\mathbf{A}$. The middle column shows the matrix $\mathbf{J}^L$ during a regular downwards Gaussian elimination sweep. A tridiagonal form is shown on the right column, where we omit calculating any elements outside the block tridiagonal.

where

$$\mathbf{g}_{ni} = (\mathbf{d}^L_{nn})^{-1}[\mathbf{J}^L]_{ni} = \begin{cases} (\mathbf{d}^L_{nn})^{-1}\mathbf{c}^L_{n-1,\dots,i} & \text{for} \quad 0 < i < n \\ (\mathbf{d}^L_{nn})^{-1} & \text{for} \quad i = n \end{cases} \tag{3.10}$$

In order to determine the second lowest block row of $\mathbf{G}$, we first eliminate the superdiagonal element $\mathbf{a}_{n-1,n}$ from the left hand side of the augmented matrix and update the right hand side elements via

$$[\mathbf{J}^L]_{n-1,i} - \mathbf{a}_{n-1,n}(\mathbf{d}^L_{nn})^{-1}[\mathbf{J}^L]_{ni} \equiv [\mathbf{J}^L]_{n-1,i} - \mathbf{a}_{n-1,n}\mathbf{g}_{ni}, \tag{3.11}$$

where $i = 1, \dots, n$ and subsequently multiplying across by the inverse of the corresponding unmodified[1] diagonal block on the left hand side of the augmented matrix, yielding the $(n-1)$th block row of the Green's function matrix $\mathbf{G}$:

$$\mathbf{g}_{n-1,i} = (\mathbf{d}^L_{n-1,n-1})^{-1}\left([\mathbf{J}^L]_{n-1,i} - \mathbf{a}_{n-1,n}\mathbf{g}_{ni}\right) \tag{3.12}$$

We can then start a process of upwards row updates in the augmented matrix terminating on the top row, yielding for each row

$$\begin{aligned}
\mathbf{g}_{n-1,i} &= (\mathbf{d}^L_{n-1,n-1})^{-1}\left([\mathbf{J}^L]_{n-1,i} - \mathbf{a}_{n-1,n}\mathbf{g}_{ni}\right), \\
\mathbf{g}_{n-2,i} &= (\mathbf{d}^L_{n-2,n-2})^{-1}\left([\mathbf{J}^L]_{n-2,i} - \mathbf{a}_{n-2,n-1}\mathbf{g}_{n-1,i}\right), \\
&\vdots \\
\mathbf{g}_{ji} &= (\mathbf{d}^L_{jj})^{-1}\left([\mathbf{J}^L]_{ji} - \mathbf{a}_{j,j+1}\mathbf{g}_{j+1,i}\right), \\
&\vdots \\
\mathbf{g}_{1i} &= (\mathbf{d}^L_{11})^{-1}\left([\mathbf{J}^L]_{1i} - \mathbf{a}_{12}\mathbf{g}_{2i}\right),
\end{aligned}$$

where $i = 1, \dots, n$, finally leaving us with the augmented matrix $[\mathbf{I}|\mathbf{G}]$, from which we can read the full inverse on the right hand side. We can also recognize from e.g. Eq. (3.8) that $[\mathbf{J}^L]_{ij} = \mathbf{0}_{ij}$ for $j > i$, and thus Eq. (3.12) can be simplified for these cases such that

$$\begin{aligned}
\mathbf{g}_{ij} &= (\mathbf{d}^L_{ii})^{-1}\left([\mathbf{J}^L]_{ij} - \mathbf{a}_{i,i+1}\mathbf{g}_{i+1,j}\right), & \text{for} \quad i > j \\
\mathbf{g}_{ij} &= -(\mathbf{d}^L_{ii})^{-1}\mathbf{a}_{i,i+1}\mathbf{g}_{i+1,j}, & \text{for} \quad i \leq j
\end{aligned} \tag{3.13}$$

for $i = n-1, n-2, \dots, 1$ and thus we save a trivial matrix addition operation.

The back solve process is visualized in Fig. 3.3, for the same example 7×7 block tridiagonal $\mathbf{A}$ treated in the figure for downwards Gaussian elimination.

[1]The value at $[\mathbf{D}^L]_{n-1,n-1}$ is unaltered due to $\mathbf{D}^L$ being upper block triangular as seen in Eq. (3.8), and thus $[\mathbf{D}^L]_{n,n-1} = \mathbf{0}_{n,n-1}$ means no modification is made to the diagonal block above it.

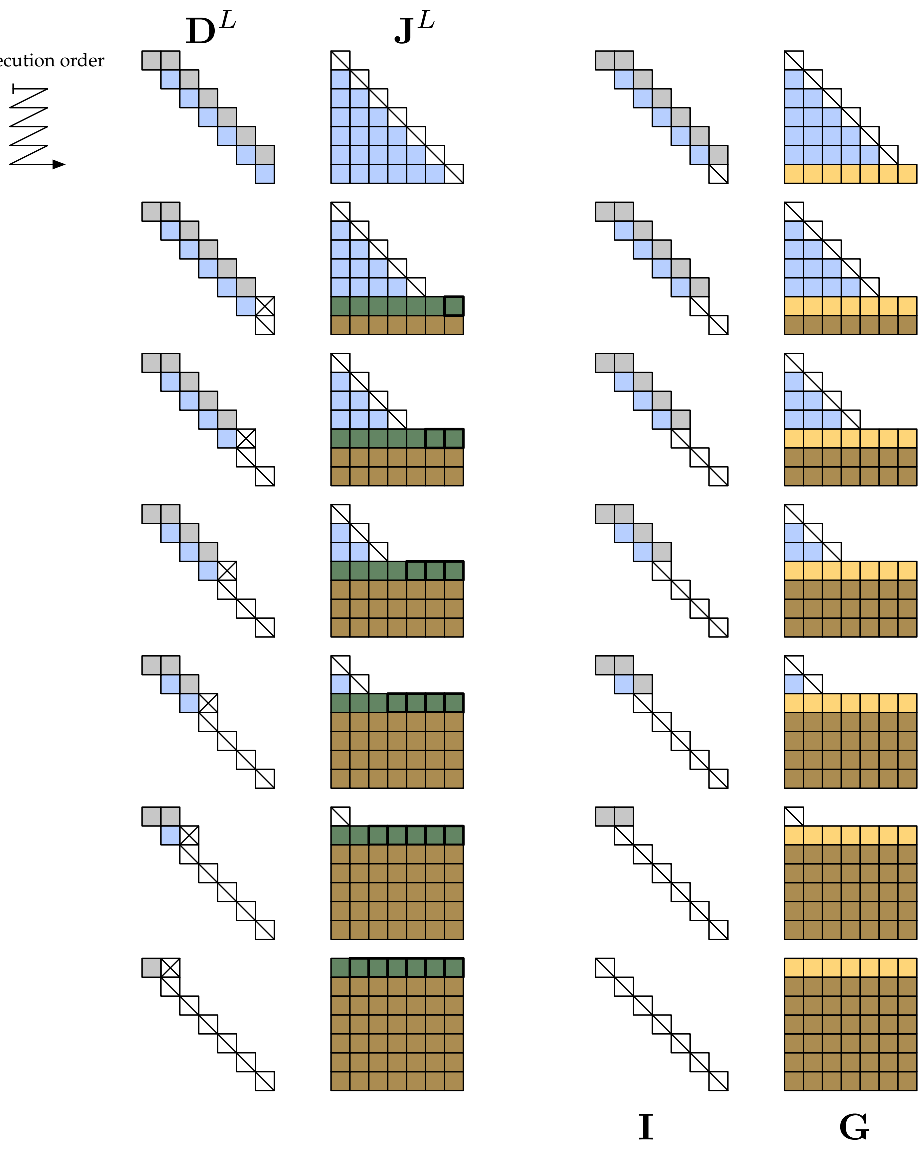


Figure 3.3: Visualizing the back solving process for calculating the full inverse **G** of a block tridiagonal matrix **A** with 7 diagonal blocks.

3.1.2 Algorithm

We now proceed to describe the process of determining $\mathbf{G}$ in an algorithmic framework, where first we concentrate on developing a set of algorithms that have the task of determining the full block matrix $\mathbf{G}$. Once this is done, we look to improve the performance of these algorithms by employing the fact that we only desire the block tridiagonal part of $\mathbf{G}$, namely $\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{G}\right\}$ and proceed to describe the modified algorithms for this optimization.

3.1.2.1 The Full Inverse

The first step in determining the full inverse is to follow a series of block Gaussian elimination steps on $\mathbf{A}$ terminating with the *echelon form* $\mathbf{D}^L$ in Eq. (3.8), which we perform by calling Alg. (3.1) GAUSSELIMINATEFULL with proper arguments.

When determining $\mathbf{G}$ via Gaussian elimination, we only need to perform a single downwards block elimination sweep from the top of $\mathbf{A}$ to the bottom. However, we provide a generalized algorithm that can handle sweeping between any two distinct rows in $\mathbf{A}$ and in either a downwards or an upwards fashion, for the sake of reusability further on. Thus this algorithm takes as arguments not only the block tridiagonal structure $\mathbf{A}$, but also a starting row k_{from} and a finishing row k_{to}. It then proceeds to perform a series of row operations that eliminate the subdiagonal blocks $\mathbf{a}_{k_{\text{from}}+1,k_{\text{from}}}, \mathbf{a}_{k_{\text{from}}+2,k_{\text{from}}+1}, \ldots, \mathbf{a}_{k_{\text{to}},k_{\text{to}}-1}$ in the case of downwards[1] Gaussian elimination, or the superdiagonal blocks $\mathbf{a}_{k_{\text{from}}-1,k_{\text{from}}}, \mathbf{a}_{k_{\text{from}}-2,k_{\text{from}}-1}, \ldots, \mathbf{a}_{k_{\text{to}},k_{\text{to}}+1}$ in the case of upwards[2] Gaussian elimination.

The result of GAUSSELIMINATEFULL are the pair of matrices $\mathbf{D}^{L,R}_{k_{\text{from}}\rightarrow k_{\text{to}}}$ and $\mathbf{J}^{L,R}_{k_{\text{from}}\rightarrow k_{\text{to}}}$ that correspond to the block row operations undertaken by block Gaussian elimination from block row k_{from} to block row k_{to}. By the superscript ^{L,R}, we mean that the algorithm can deliver matrices that correspond to either downwards elimination from upper left to lower right, ^{L}, or upwards elimination from lower left to upper right, ^{R}, depending on k_{from} and k_{to}. In the special case of $k_{\text{from}} = 1$ and $k_{\text{to}} = n$, we obtain $\mathbf{D}^L$ and $\mathbf{J}^L$ as in Eq. (3.5), while for $k_{\text{from}} = n$ and $k_{\text{to}} = 1$, we get $\mathbf{D}^R$ and $\mathbf{J}^R$ as in Eq. (3.6).

Looking at Alg. (3.1) GAUSSELIMINATEFULL, we see how a downward sweep is handled when the case on line 1 evaluates to be true. We initialize the matrix $\mathbf{D}^L$ by line 2, and we proceed to iterate for the remaining rows by

[1]In downwards elimination we have $k_{\text{from}} < k_{\text{to}}$.

[2]In upwards elimination we have $k_{\text{from}} > k_{\text{to}}$.

the for loop on line 3. The execution of line 4 determines all the subdiagonal blocks in $\mathbf{J}^L$, while the for loop on line 5 takes care of determining all remaining subdiagonal blocks on the current row. The last statement in the for loop running over the remaining rows in $\mathbf{A}$ is on line 6, which updates the diagonal block in $\mathbf{D}^L$ for the current row in the iteration. The upwards sweep handled by the case on line 9 is a mirror image of the downwards sweep, and is not discussed further.

Algorithm 3.1 GAUSSELIMINATEFULL($\mathbf{A}, k_{\text{from}}, k_{\text{to}}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$, $k_{\text{from}} \in \mathbb{N}$, $k_{\text{to}} \in \mathbb{N}$, $0 < k_{\text{from}} \leq n, 0 < k_{\text{to}} \leq n$, $k_{\text{from}} \neq k_{\text{to}}$
 1: if $k_{\text{from}} < k_{\text{to}}$ then                                   downwards elimination
 2:     $\mathbf{d}^L_{k_{\text{from}},k_{\text{from}}} \leftarrow \mathbf{a}_{k_{\text{from}},k_{\text{from}}}$                                   initialize
 3:     for $i \leftarrow k_{\text{from}}$ up to $k_{\text{to}} - 1$ do                         sweep down
 4:         $\mathbf{c}^L_i \leftarrow -\mathbf{a}_{i+1,i}(\mathbf{d}^L_{ii})^{-1}$                    1op($\mathfrak{L}\mathfrak{U}$), 1op($\times$)
 5:         for $j \leftarrow i - 1$ down to $k_{\text{from}}$ do                only for full inversion
 6:             $\mathbf{c}^L_{i,\ldots,j} \leftarrow \mathbf{c}^L_i \mathbf{c}^L_{i-1,\ldots,j}$                                1op($\times$)
 7:         $\mathbf{d}^L_{i+1,i+1} \leftarrow \mathbf{a}_{i+1,i+1} + \mathbf{c}^L_i \mathbf{a}_{i,i+1}$                  1op($\times$), 1op($+$)
 8:     return $\mathbf{D}^L_{k_{\text{from}}\to k_{\text{to}}}, \mathbf{J}^L_{k_{\text{from}}\to k_{\text{to}}}$
 9: else                                                          upwards elimination
10:     $\mathbf{d}^R_{k_{\text{from}},k_{\text{from}}} \leftarrow \mathbf{a}_{k_{\text{from}},k_{\text{from}}}$                                   initialize
11:     for $i \leftarrow k_{\text{from}}$ down to $k_{\text{to}} + 1$ do                          sweep up
12:         $\mathbf{c}^R_i \leftarrow -\mathbf{a}_{i-1,i}(\mathbf{d}^R_{ii})^{-1}$                    1op($\mathfrak{L}\mathfrak{U}$), 1op($\times$)
13:         for $j \leftarrow k_{\text{to}}$ up to $i + 1$ do                   only for full inversion
14:             $\mathbf{c}^R_{i,\ldots,j} \leftarrow \mathbf{c}^R_i \mathbf{c}^R_{i+1,\ldots,j}$                                1op($\times$)
15:         $\mathbf{d}^R_{i-1,i-1} \leftarrow \mathbf{a}_{i-1,i-1} + \mathbf{c}^R_i \mathbf{a}_{i,i-1}$                  1op($\times$), 1op($+$)
16:     return $\mathbf{D}^R_{k_{\text{from}}\to k_{\text{to}}}, \mathbf{J}^R_{k_{\text{from}}\to k_{\text{to}}}$
```

Following a full Gaussian elimination sweep, we need to perform a back solve sweep in order to produce the desired inverse $\mathbf{G}$. This is done by the Alg. (3.2) BACKSOLVEFULL that assumes as input arguments the output of a full downwards block Gaussian elimination sweep resulting in Eq. (3.8).

The algorithm initializes by calculating the block g_{nn} of the inverse on line 1, and uses the for loop on line 2 to determine the entire bottom block row of the inverse $\mathbf{G}$. With this initialization, the algorithm loops over the remaining rows in an upwards back solve sweep on line 4, where sub diagonal and diagonal blocks of the inverse are determined by the loop on line 5, and where super diagonal blocks of the inverse are determined by the loop on line 7, where we can save a trivial matrix addition operation.

The algorithm for determining the full inverse $\mathbf{G}$ is detailed in Alg. (3.3)

Algorithm 3.2 BACKSOLVEFULL$(\mathbf{A}, \mathbf{D}^L, \mathbf{J}^L)$

```
Require: $\mathbf{A}, \mathbf{D}^L, \mathbf{J}^L \in \mathbb{B}^{n,n}$
Ensure: $\mathbf{G} = \mathbf{A}^{-1} \in \mathbb{B}^{n,n}$
1: $\mathbf{g}_{nn} \leftarrow (\mathbf{d}^L_{nn})^{-1}$                                    initialize: $1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
2: for $j \leftarrow 1$ up to $n-1$ do                                   determine the nth row
3:     $\mathbf{g}_{nj} \leftarrow \mathbf{g}_{nn}[\mathbf{J}^L]_{nj}$                                     $1\mathsf{op}(\times)$
4: for $i \leftarrow n-1$ down to $1$ do                                   eliminate upwards
5:     for $j \leftarrow 1$ up to $i$ do                              sub diagonals and diagonals
6:         $\mathbf{g}_{ij} \leftarrow (\mathbf{d}^L_{ii})^{-1}\left([\mathbf{J}^L]_{ij} - \mathbf{a}_{i,i+1}\mathbf{g}_{i+1,j}\right)$         $1\mathsf{op}(+), 2\mathsf{op}(\times)$
7:     for $j \leftarrow i+1$ up to $n$ do                                  super diagonals
8:         $\mathbf{g}_{ij} \leftarrow -(\mathbf{d}^L_{ii})^{-1}\mathbf{a}_{i,i+1}\mathbf{g}_{i+1,j}$                       $2\mathsf{op}(\times)$
9: return $\mathbf{G}$
```

GEINVERSEFULL, and takes care of calling GAUSSELIMINATEFULL on line 1 to perform a full, downwards Gaussian elimination sweep followed by BACKSOLVEFULL on line 2 for a full back solve sweep in order to construct $\mathbf{G}$.

Algorithm 3.3 GEINVERSEFULL$(\mathbf{A})$

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\mathbf{G} = \mathbf{A}^{-1} \in \mathbb{B}^{n,n}$
1: $\mathbf{D}^L, \mathbf{J}^L \leftarrow$ GAUSSELIMINATEFULL$(\mathbf{A}, 1, n)$   full downwards elimination sweep
2: $\mathbf{G} \leftarrow$ BACKSOLVEFULL$(\mathbf{A}, \mathbf{D}^L, \mathbf{J}^L)$                    back–solve sweep
3: return $\mathbf{G}$
```

3.1.2.2 The Block Tridiagonal Inverse

We are actually only interested in obtaining the same block tridiagonal structure of $\mathbf{G}$ that $\mathbf{A}$ has, i.e. we want $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$. This motivates us to improve Alg. (3.2) BACKSOLVEFULL and Alg. (3.3) GEINVERSEFULL to take advantage of this fact, and we thus present an improved back solve in Alg. (3.5) BACKSOLVETRI.

This leads to the improved algorithm Alg. (3.6) GEINVERSETRI that calls both GAUSSELIMINATEFULL and BACKSOLVETRI in sequence in order to give us $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ for a given block tridiagonal matrix $\mathbf{A}$.

The algorithm Alg. (3.1) GAUSSELIMINATEFULL cannot be improved upon in this case, as the improved Alg. (3.5) BACKSOLVETRI still needs the same full output $\mathbf{J}^L$ as delivered by GAUSSELIMINATEFULL. We do, however, present an algorithm in Alg. (3.4) GAUSSELIMINATETRI that omits calculating elements of

J off the tridiagonal. This routine will be of use in the *sweep* approach based algorithm introduced later in Sec. 3.2, but is included here due to its close relationship to GaussEliminateFull.

The main difference between the optimized GaussEliminateTri and its full version GaussEliminateFull lies in the omission of the for loop on line 5 of GaussEliminateFull that takes care of determining elements below the block sub diagonal of $\mathbf{J}^L$. This, as we will see later, leads to a substantial difference in the computational complexity of the algorithm over GaussEliminateFull.

Algorithm 3.4 GaussEliminateTri$(\mathbf{A}, k_{\text{from}}, k_{\text{to}})$

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$, $k_{\text{from}} \in \mathbb{N}$, $k_{\text{to}} \in \mathbb{N}$, $0 < k_{\text{from}} \leq n, 0 < k_{\text{to}} \leq n$, $k_{\text{from}} \neq k_{\text{to}}$
 1: if $k_{\text{from}} < k_{\text{to}}$ then                                   downwards elimination
 2:     $\mathbf{d}^L_{k_{\text{from}},k_{\text{from}}} \leftarrow \mathbf{a}_{k_{\text{from}},k_{\text{from}}}$                  initialize
 3:     for $i \leftarrow k_{\text{from}}$ up to $k_{\text{to}} - 1$ do                           sweep down
 4:         $\mathbf{c}^L_i \leftarrow -\mathbf{a}_{i+1,i}(\mathbf{d}^L_{ii})^{-1}$                 $1\text{op}(\mathcal{LU}), 1\text{op}(\times)$
 5:         $\mathbf{d}^L_{i+1,i+1} \leftarrow \mathbf{a}_{i+1,i+1} + \mathbf{c}^L_i \mathbf{a}_{i,i+1}$        $1\text{op}(+), 1\text{op}(\times)$
 6:     return $\mathbf{D}^L_{k_{\text{from}} \to k_{\text{to}}}, \text{Trid}_{\mathbf{A}} \left\{ \mathbf{J}^L_{k_{\text{from}} \to k_{\text{to}}} \right\}$
 7: else                                                  upwards elimination
 8:     $\mathbf{d}^R_{k_{\text{from}},k_{\text{from}}} \leftarrow \mathbf{a}_{k_{\text{from}},k_{\text{from}}}$                  initialize
 9:     for $i \leftarrow k_{\text{from}}$ down to $k_{\text{to}} + 1$ do                           sweep up
10:         $\mathbf{c}^R_i \leftarrow -\mathbf{a}_{i-1,i}(\mathbf{d}^R_{ii})^{-1}$                 $1\text{op}(\mathcal{LU}), 1\text{op}(\times)$
11:         $\mathbf{d}^R_{i-1,i-1} \leftarrow \mathbf{a}_{i-1,i-1} + \mathbf{c}^R_i \mathbf{a}_{i,i-1}$        $1\text{op}(+), 1\text{op}(\times)$
12:     return $\mathbf{D}^R_{k_{\text{from}} \to k_{\text{to}}}, \text{Trid}_{\mathbf{A}} \left\{ \mathbf{J}^R_{k_{\text{from}} \to k_{\text{to}}} \right\}$
```

The difference between BackSolveTri and BackSolveFull also lies in the replacement of a for loop, namely line 7 in BackSolveFull, with the single statement on line 7 that ensures we only calculate the super diagonal blocks. This, however, does not lead to the same sort of computational complexity change that GaussEliminateTri exhibits, since the for loop on line 5 remains. The optimized back solve process can be seen in Fig. 3.4 where we can see the necessity of calculating the entire subdiagonal portion of **G**, despite only seeking the block tridiagonal.

Finally, Alg. (3.6) GEInverseTri has no true difference over the structure of the unoptimized GEInverseFull, other than taking care to call the optimized version of the back solve routine.

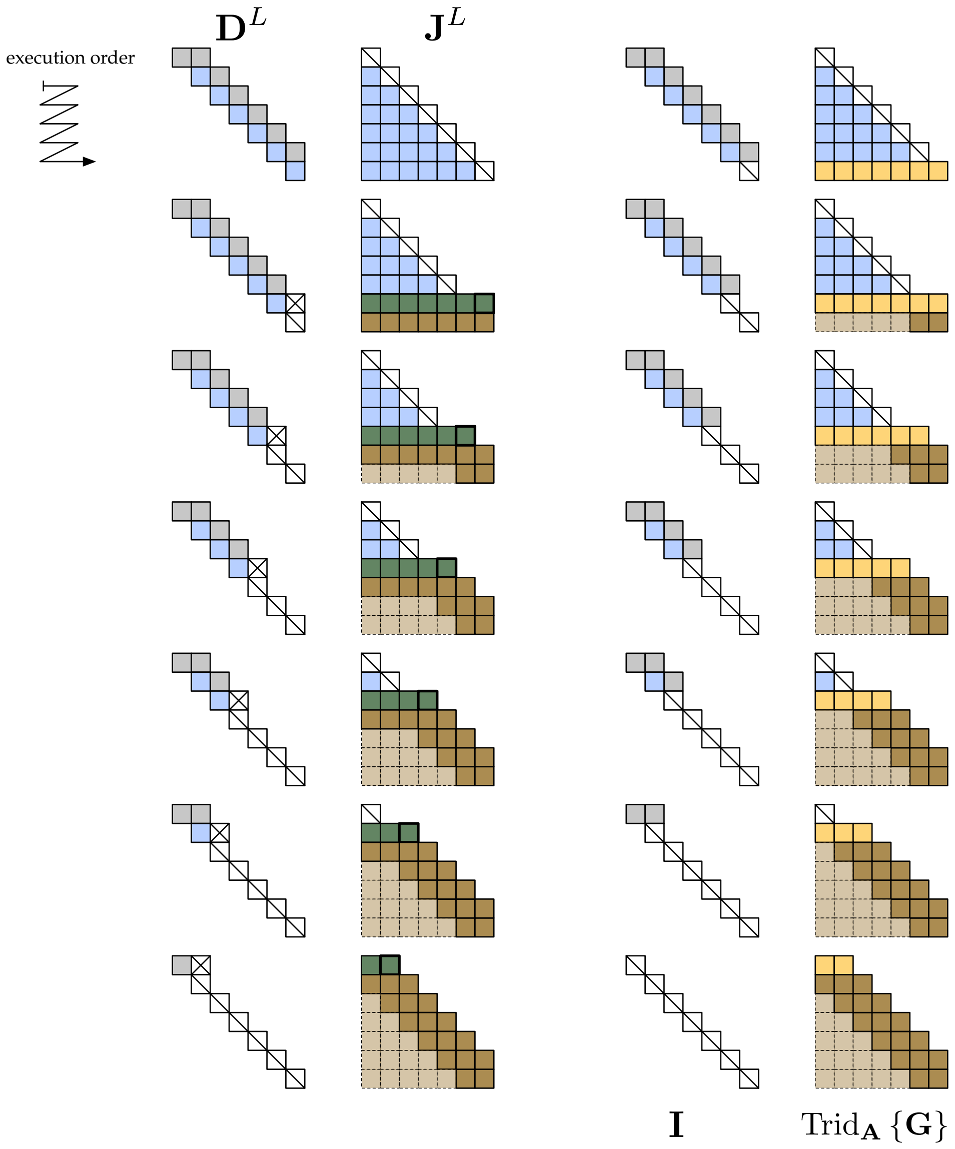


Figure 3.4: Visualizing the back solving process for calculating the block tridiagonal portion of the inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$ with 7 diagonal blocks.

Algorithm 3.5 BACKSOLVETRI($\mathbf{A}, \mathbf{D}^L, \mathbf{J}^L$)

```
Require: $\mathbf{A}, \mathbf{D}^L, \mathbf{J}^L \in \mathbb{B}^{n,n}$
Ensure: $\text{Trid}_\mathbf{A}\{\mathbf{G}\} = \text{Trid}_\mathbf{A}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: $\mathbf{g}_{nn} \leftarrow (\mathbf{d}^L_{nn})^{-1}$                          initialize: $1\text{op}(\mathcal{LU}), 1\text{op}(\times)$
2: for $j \leftarrow 1$ up to $n-1$ do                                 determine the nth row
3:     $\mathbf{g}_{nj} \leftarrow \mathbf{g}_{nn}[\mathbf{J}^L]_{nj}$                                  $1\text{op}(\times)$
4: for $i \leftarrow n-1$ down to $1$ do                               eliminate upwards
5:     for $j \leftarrow 1$ up to $i$ do                            propagate sub diagonals upwards
6:         $\mathbf{g}_{ij} \leftarrow (\mathbf{d}^L_{ii})^{-1}\left([\mathbf{J}^L]_{ij} - \mathbf{a}_{i,i+1}\mathbf{g}_{i+1,j}\right)$        $1\text{op}(+), 2\text{op}(\times)$
7:     $\mathbf{g}_{i,i+1} \leftarrow -(\mathbf{d}^L_{ii})^{-1}\mathbf{a}_{i,i+1}\mathbf{g}_{i+1,i+1}$               super diagonals: $2\text{op}(\times)$
8: return $\text{Trid}_\mathbf{A}\{\mathbf{G}\}$
```

Algorithm 3.6 GEINVERSETRI($\mathbf{A}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\text{Trid}_\mathbf{A}\{\mathbf{G}\} = \text{Trid}_\mathbf{A}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: $\mathbf{D}^L, \mathbf{J}^L \leftarrow$ GAUSSELIMINATEFULL($\mathbf{A}, 1, n$)    full downwards elimination sweep
2: $\text{Trid}_\mathbf{A}\{\mathbf{G}\} \leftarrow$ BACKSOLVETRI($\mathbf{A}, \mathbf{D}^L, \mathbf{J}^L$)          solve up to the tridiagonal
3: return $\text{Trid}_\mathbf{A}\{\mathbf{G}\}$
```

3.1.3 Complexity Analysis

We now look to perform a complexity analysis on the algorithms presented in this section such that we can make a qualitative assessment of their running times, and to eventually compare the efficiency of the Gaussian elimination approach to that of the sweep approach, presented in Sec. 3.2.

We will be considering the case of determining either $\mathbf{G}$ or $\text{Trid}_\mathbf{A}\{\mathbf{G}\}$ using either GEINVERSEFULL or GEINVERSETRI, respectively. In this case, and throughout this paper, we consider the case where $\mathbf{A} \in \mathbb{B}^{n,n}$, that is, it is a square block tridiagonal matrix with n diagonal blocks.

3.1.3.1 Gaussian Elimination

Looking at Alg. (3.3) GEINVERSEFULL, we see that it calls a full Gaussian elimination sweep on line 1 using GAUSSELIMINATEFULL followed by a back solve sweep on line 2 with BACKSOLVEFULL. The block tridiagonal optimized version Alg. (3.6) GEINVERSETRI has its optimization in the back–solving process, and thus uses the same algorithm GAUSSELIMINATEFULL, however, as a close relative to this algorithm is presented here in GAUSSELIMINATETRI, we include complexity analysis for it here also.

Looking at GAUSSELIMINATEFULL, the for loop on line 3 will loop a total of $n-1$ times, generating $n-1$ $\mathbf{c}_i^L$ factors from line 4, each costing an LU factorization and a multiplication. Where GAUSSELIMINATEFULL and GAUSSELIMINATETRI differ is in GAUSSELIMINATETRI's lack of the for loop on line 5 that generates all the elements in $\mathbf{J}^L$ below the block sub diagonal[1]. This for loop will loop a total of $1, 2, \ldots, n-2$ times, depending on how far we have proceeded in block eliminations, for a total of $\frac{1}{2}(n^2-3n+2)$ executions, determined using the arithmetic series identity

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{1}{2}n(n+1). \tag{3.14}$$

The matrix $\mathbf{A}$ is then updated to produce $\mathbf{D}^L$ on line 7 in GAUSSELIMINATEFULL and on line 6 a total of $n-1$ times, each time costing a matrix addition and multiplication operation.

3.1.3.2 Back Solving

After a full downwards block Gaussian elimination sweep, the algorithms BACKSOLVEFULL or BACKSOLVETRI are called to construct the inverse $\mathbf{G}$ or $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$, with the results of Alg. (3.1) GAUSSELIMINATEFULL and Alg. (3.4) GAUSSELIMINATETRI, respectively.

Taking BACKSOLVEFULL, we initialize with a simple matrix inversion on line 1 involving an LU factorization and a relative matrix multiplication. This is the only LU factorization we need to explicitly calculate in this routine, provided we have saved the factorizations from GAUSSELIMINATEFULL. This is followed by determination of the entire bottom block row of $\mathbf{G}$, which involves $n-1$ multiplications. Finally, we sweep upwards, where blocks on and below the diagonal are determined on line 6 and blocks strictly above the diagonal are determined on line 8. Since we assume we save the LU factorizations performed in GAUSSELIMINATEFULL, line 6 will incur an addition operation as well as what corresponds to two matrix multiplications, while line 8 only needs two matrix multiplications.

The main difference between BACKSOLVEFULL and the optimized BACKSOLVETRI rests on line 8 in BACKSOLVEFULL, where in the tridiagonal optimized version we only calculate the necessary super diagonal blocks. Thus

[1]similarly for line 13 in GAUSSELIMINATEFULL and $\mathbf{J}^R$ with respect to elements above the super diagonal.

the for loop becomes a single statement in BACKSOLVETRI on line 7 executed a total of $n-1$ times, needing two matrix multiplications each time.

All in all, line 6 in BACKSOLVEFULL is executed a total of $n-1, n-2, \ldots, 1$ times as we propagate upwards in constructing $\mathbf{G}$, while line 8 executes a total of $1, 2, \ldots, n-1$ times as we proceed. The total amount of executions can again be determined via Eq. (3.14), and we have a total of $\frac{1}{2}n(n-1)$ executions of line 6 and line 8, each.

3.1.3.3 Results

Complexity Analysis for Gaussian Elimination

Calculation	LU–factorizations op($\mathcal{LU}$)	Multiplications op($\times$)	Additions op($+$)
GAUSSELIMINATEFULL	$n-1$	$\frac{1}{2}(n^2+n-2)$	$n-1$
BACKSOLVEFULL	1	$2n^2-n$	$\frac{1}{2}(n^2-n)$
GEINVERSEFULL	n	$\frac{1}{2}(5n^2-n-2)$	$\frac{1}{2}(n^2+n-2)$
GAUSSELIMINATEFULL	$n-1$	$\frac{1}{2}(n^2+n-2)$	$n-1$
BACKSOLVETRI	1	n^2+2n-2	$\frac{1}{2}(n^2-n)$
GEINVERSETRI	n	$\frac{1}{2}(3n^2+5n-6)$	$\frac{1}{2}(n^2+n-2)$

Table 3.1: This table illustrates the amount of basic operations performed in calculating either the full inverse $\mathbf{G}$ of $\mathbf{A}$, or only the block tridiagonal part of it $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$, using the Gaussian elimination algorithms presented in this section. The second, third and fourth columns refer to the amount of basic matrix operations of LU–factorization, multiplication and addition involved in each algorithm. The term n is the total amount of diagonal blocks in $\mathbf{A} \in \mathbb{B}^{n,n}$.

A summary of the complexity analysis results for Alg. (3.3) GEINVERSEFULL and Alg. (3.6) GEINVERSETRI is given in Table 3.1, where we have divided the tally of operation counts up among the routines called by each algorithm. We will focus mainly on the complexities of LU factorization and matrix multiplication as they are cubic with respect to matrix dimension and will dominate overall algorithmic cost, while addition is only quadratic and has a far weaker influence on the total running time of the algorithm. It is, however, included for the sake of completeness. As we can see, both algorithms have the same linear $\mathcal{O}(n)$ complexity in terms of LU factorizations.

With respect to multiplications, even though GAUSSELIMINATETRI seeks to only determine the block tridiagonal portion of $\mathbf{G}$, it still has the overall same quadratic complexity $\mathcal{O}(n^2)$ as GAUSSELIMINATEFULL which calculates the full inverse $\mathbf{G}$. The only difference is that GEINVERSETRI has a slightly cheaper cost factor of 1.5 versus a cost factor of 2.5 for GEINVERSEFULL.

3.2 Sweep

As we saw in the previous section, the complexity for the standard Gauss elimination algorithm in determining either $\mathbf{G}$ or $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ is still quadratic. With a different approach, we show in this section that it is possible to determine the inverse with linear complexity. The work presented here is based on work published in [1] which is included in appendix A.

3.2.1 Description

The new approach seeks to combine the results of a downwards and upwards Gaussian elimination sweep on $\mathbf{A}$ that results in their echelon forms $\mathbf{D}^L$ and $\mathbf{D}^R$, respectively, in a way such that we remain with a matrix solely composed of blocks on the main diagonal. This is because such a matrix is easy to work with, as each row is independent of the others. With such a matrix, generating the inverse is trivial.

Looking at these echelon forms, we see that they are both bidiagonal, where the nonzero off–diagonal has elements identical to that of $\mathbf{A}$. Combining the echelon forms and $\mathbf{A}$ in the following manner, leaves us with the desired block diagonal matrix form:

Combining the results obtained from Eqs. (2.68), (3.5), and (3.6) by employing the fact that

$$\mathbf{AG} = \mathbf{I}, \quad \mathbf{D}^L\mathbf{G} = \mathbf{J}^L, \quad \mathbf{D}^R\mathbf{G} = \mathbf{J}^R, \tag{3.15}$$

the expression

$$\left(\mathbf{A} - \mathbf{D}^L - \mathbf{D}^R\right)\mathbf{G} = \mathbf{I} - \mathbf{J}^L - \mathbf{J}^R \tag{3.16}$$

is examined, which can be viewed as the following augmented matrix expression:

$$\left[\begin{array}{c|c} \mathbf{B} & \mathbf{F} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{I} \end{array}\right] - \left[\begin{array}{c|c} \mathbf{D}^L & \mathbf{J}^L \end{array}\right] - \left[\begin{array}{c|c} \mathbf{D}^R & \mathbf{J}^R \end{array}\right], \tag{3.17}$$

where

$$\mathbf{B} = \begin{pmatrix} \mathbf{a}_{11} - \mathbf{d}^L_{11} - \mathbf{d}^R_{11} & & & \\ & \mathbf{a}_{22} - \mathbf{d}^L_{22} - \mathbf{d}^R_{22} & & \\ & & \mathbf{a}_{33} - \mathbf{d}^L_{33} - \mathbf{d}^R_{33} & \\ & & & \ddots \end{pmatrix} \tag{3.18}$$

and

$$\mathbf{F} = \begin{pmatrix} -\mathbf{i}_{11} & -\mathbf{c}_2^R & -\mathbf{c}_{2,3}^R & -\mathbf{c}_{2,3,4}^R & \cdots \\ -\mathbf{c}_1^L & -\mathbf{i}_{22} & -\mathbf{c}_3^R & -\mathbf{c}_{3,4}^R & \cdots \\ -\mathbf{c}_{2,1}^L & -\mathbf{c}_2^L & -\mathbf{i}_{33} & -\mathbf{c}_4^R & \cdots \\ -\mathbf{c}_{3,2,1}^L & -\mathbf{c}_{3,2}^L & -\mathbf{c}_3^L & -\mathbf{i}_{44} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{3.19}$$

When $\mathbf{B}$ is subsequently reduced to the identity matrix $\mathbf{I}$, $\mathbf{F}$ will simultaneously be transformed into the Green's function matrix $\mathbf{G}$. In other words, the Green's function matrix sought for can be expressed as $\mathbf{G} = \mathbf{B}^{-1}\mathbf{F}$. The Green's function matrix is:

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{11}\mathbf{c}_2^R & \mathbf{g}_{11}\mathbf{c}_{2,3}^R & \cdots & \mathbf{g}_{11}\mathbf{c}_{2,\dots,n}^R \\ \mathbf{g}_{22}\mathbf{c}_1^L & \mathbf{g}_{22} & \mathbf{g}_{22}\mathbf{c}_3^R & \cdots & \mathbf{g}_{22}\mathbf{c}_{3,\dots,n}^R \\ \mathbf{g}_{33}\mathbf{c}_{2,1}^L & \mathbf{g}_{33}\mathbf{c}_2^L & \mathbf{g}_{33} & \cdots & \mathbf{g}_{33}\mathbf{c}_{4,\dots,n}^R \\ \mathbf{g}_{44}\mathbf{c}_{3,2,1}^L & \mathbf{g}_{44}\mathbf{c}_{3,2}^L & \mathbf{g}_{44}\mathbf{c}_3^L & \cdots & \mathbf{g}_{44}\mathbf{c}_{5,\dots,n}^R \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{g}_{nn}\mathbf{c}_{n-1,\dots,1}^L & \mathbf{g}_{nn}\mathbf{c}_{n-1,\dots,2}^L & \mathbf{g}_{nn}\mathbf{c}_{n-1,\dots,3}^L & \cdots & \mathbf{g}_{nn} \end{pmatrix}, \tag{3.20}$$

where the following expression for the diagonal blocks of the Green's function matrix is introduced:

$$\mathbf{g}_{ii} = -\mathbf{b}_{ii}^{-1} = \left(-\mathbf{a}_{ii} + \mathbf{d}_{ii}^L + \mathbf{d}_{ii}^R\right)^{-1} \quad \text{where } i = 1, 2, \dots, n. \tag{3.21}$$

The matrices $\mathbf{g}_{11}$ and $\mathbf{g}_{nn}$ can be found in a simplified manner, however, by considering the e.g. the nth block from Eq. (3.21):

$$\mathbf{g}_{nn} = \left(-\mathbf{a}_{nn} + \mathbf{d}_{nn}^L + \mathbf{d}_{nn}^R\right)^{-1} = \left(\mathbf{d}_{nn}^L\right)^{-1}, \tag{3.22}$$

since $\mathbf{d}_{nn}^R = \mathbf{a}_{nn}$. This holds similarly for the first row of the Green's function matrix. From this, it is seen that the first and last diagonal blocks of the Green's function matrix correspond to the final blocks of upwards and downwards sweeps of block Gaussian elimination, respectively, in the following manner:

$$\mathbf{g}_{11} = \left(\mathbf{d}_{11}^R\right)^{-1} \quad \text{and} \quad \mathbf{g}_{nn} = \left(\mathbf{d}_{nn}^L\right)^{-1}. \tag{3.23}$$

The off diagonal entries are then calculated via appropriate multiplications with calculated diagonal block matrices and factors obtained during block Gaussian elimination as follows using the notation given in Eq. (3.7):

$$\mathbf{g}_{ij} = \mathbf{g}_{ii}\mathbf{c}_{i+1,i+2,\dots,j-1,j}^R, \text{ for } i < j \tag{3.24}$$

$$\mathbf{g}_{ij} = \mathbf{g}_{ii}\mathbf{c}_{i-1,i-2,\dots,j+1,j}^L, \text{ for } i > j. \tag{3.25}$$

3.2.2 Algorithm

We present here the process of determining the full $\mathbf{G}$ and $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ via the sweep method in an algorithmic framework as done before for the standard Gaussian elimination method. Although the sweep algorithm was first developed with determining the block tridiagonal part of $\mathbf{G}$ in mind, we include for completeness a variant of sweep that seeks to determine the full $\mathbf{G}$ that we can compare to the earlier developed Gaussian elimination algorithm.

3.2.2.1 The Full Inverse

The sweep based algorithm for determining the full inverse $\mathbf{G}$ is presented in Alg. (3.9), which combines two calls to Alg. (3.1) GAUSSELIMINATEFULL for the upwards and downwards Gaussian elimination "sweeps" that give name to the algorithm. This is then followed by Alg. (3.7) DIAGONALS that combines the results of the sweeps and generates the block diagonals of the inverse $\mathbf{G}$. Finally, Alg. (3.8) OFFDIAGONALSFULL takes care of generating all the off–diagonal blocks such that we have generated the full inverse $\mathbf{G}$.

The algorithm DIAGONALS, called after the completion of both an upwards and downwards Gaussian elimination step, seeks to determine the diagonal blocks of $\mathbf{G}$ according to Eq. (3.21) along with the optimizations in Eq. (3.23) for $\mathbf{g}_{11}$ and $\mathbf{g}_{nn}$. These optimizations are handled separately on line 1 and line 4, while the combination of the upwards and downwards sweep results as Eq. (3.21) is handled on line 3. This line is wrapped in a for loop that takes care of iterating over all diagonal blocks except $\mathbf{g}_{11}$ and $\mathbf{g}_{nn}$.

Algorithm 3.7 DIAGONALS($\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R$)

```
Require: $\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R \in \mathbb{B}^{n,n}$
Ensure: $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} = \mathrm{Diag}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: $\mathbf{g}_{11} \leftarrow (\mathbf{d}^R_{11})^{-1}$                          $1\mathrm{op}(\mathcal{LU}), 1\mathrm{op}(\times)$
2: for $i \leftarrow 2$ up to $n-1$ do                       determine "inner" diagonal blocks
3:     $\mathbf{g}_{ii} \leftarrow (-\mathbf{a}_{ii} + \mathbf{d}^L_{ii} + \mathbf{d}^R_{ii})^{-1}$      $2\mathrm{op}(+), 1\mathrm{op}(\mathcal{LU}), 1\mathrm{op}(\times)$
4: $\mathbf{g}_{nn} \leftarrow (\mathbf{d}^L_{nn})^{-1}$                          $1\mathrm{op}(\mathcal{LU}), 1\mathrm{op}(\times)$
5: return $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$
```

The procedure to handle the off diagonals of $\mathbf{G}$ is called right after the diagonal blocks have been determined, and uses the results of the Gaussian elimination sweeps as it calculates. Looping over all rows in $\mathbf{G}$ on line 1, sub diagonal blocks are determined via line 2 while super diagonal elements are determined on line 4.

Algorithm 3.8 OFFDIAGONALSFULL($\mathbf{A}, \mathbf{J}^L, \mathbf{J}^R, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$)

```
Require: $\mathbf{A}, \mathbf{J}^L, \mathbf{J}^R \in \mathbb{B}^{n,n}$, $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} = \mathrm{Diag}_{\mathbf{A}}\{\mathbf{A}^{-1}\}$
Ensure: $\mathbf{G} = \mathbf{A}^{-1} \in \mathbb{B}^{n,n}$
1: for $i \leftarrow 1$ up to $n$ do                          loop over all block rows
2:     for $j \leftarrow i-1$ down to $1$ do                 determine blocks under the diagoonal
3:         $\mathbf{g}_{ij} \leftarrow \mathbf{g}_{i,j+1}\mathbf{c}_j^L = \mathbf{g}_{ii}[\mathbf{J}^L]_{ij}$                                      1op($\times$)
4:     for $j \leftarrow i+1$ up to $n$ do                   determine blocks over the diagonal
5:         $\mathbf{g}_{ij} \leftarrow \mathbf{g}_{i,j-1}\mathbf{c}_j^R = \mathbf{g}_{ii}[\mathbf{J}^R]_{ij}$                                      1op($\times$)
6: return G
```

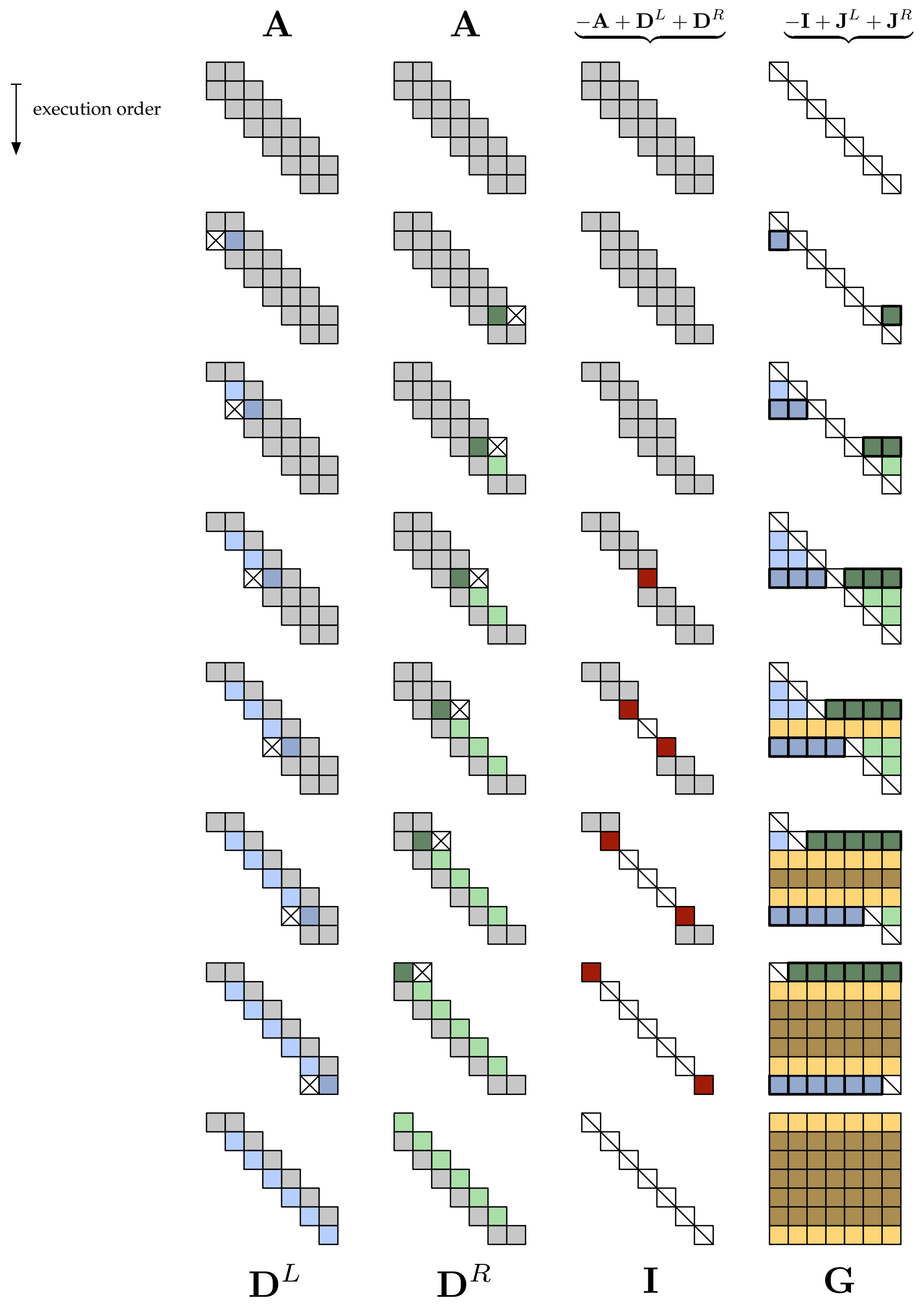
Finally, the procedure that determines the full inverse **G** for some **A** using the sweeps algorithm is given in Alg. (3.9) SWEEPINVERSEFULL. It sequentially executes a downwards Gaussian elimination sweep on line 1 followed by an upwards sweep on line 2. These two sweeps are independent of each other, and may be swapped in order. The diagonal blocks of **G** are then computed on line 3, followed by the routine on line 4 that handles computing all off diagonal blocks of **G**.

Algorithm 3.9 SWEEPINVERSEFULL($\mathbf{A}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\mathbf{G} = \mathbf{A}^{-1} \in \mathbb{B}^{n,n}$
1: $\mathbf{D}^L, \mathbf{J}^L \leftarrow$ GAUSSELIMINATEFULL($\mathbf{A}, 1, n$)
2: $\mathbf{D}^R, \mathbf{J}^R \leftarrow$ GAUSSELIMINATEFULL($\mathbf{A}, n, 1$)
3: $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} \leftarrow$ DIAGONALS($\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R$)
4: $\mathbf{G} \leftarrow$ OFFDIAGONALSFULL($\mathbf{A}, \mathbf{J}^L, \mathbf{J}^R, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$)
5: return G
```

A visualization of the sweep algorithm SWEEPINVERSEFULL working on an example block tridiagonal **A** is given in Fig. 3.5. The figure shows how as a downwards and upwards Gaussian elimination sweep on **A**, on the first and second column from the left, respectively, combines via Eq. (3.21) and Eqs. (3.23)–(3.25) to form **B** and **G** in the third and fourth columns from the left, respectively.

This figure should only be taken as a rough assessment of how the mathematics work, and not the algorithm itself, in that both the upwards and downwards Gaussian elimination sweeps are computed sequentially, and not in parallel. Furthermore, the construction of **G** is done first by sequentially determining $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$ from $\mathbf{a}_{11}$ down to $\mathbf{a}_{nn}$, and then the desired off diagonals.



$\mathbf{A}$
$\mathbf{A}$
$\underbrace{-\mathbf{A} + \mathbf{D}^L + \mathbf{D}^R}$
$\underbrace{-\mathbf{I} + \mathbf{J}^L + \mathbf{J}^R}$
execution order
$\mathbf{D}^L$
$\mathbf{D}^R$
$\mathbf{I}$
$\mathbf{G}$

Figure 3.5: Visualizing the sweep method for calculating the full inverse $\mathbf{G}$ of a block tridiagonal matrix $\mathbf{A}$ with 7 diagonal blocks.

3.2.2.2 The Block Tridiagonal Inverse

Since we only desire the block tridiagonal portion of the inverse, we modify the Sweep algorithm SWEEPINVERSETRI such that it only returns $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$. Since only the OFFDIAGONALSFULL call handles this, we present an optimized version in Alg. (3.10) OFFDIAGONALSTRI which only returns the sub diagonal and super diagonal blocks of $\mathbf{G}$. This is done by replacing the for loops on line 2 and line 4 in OFFDIAGONALSFULL with the single statements on line 3 and line 4 in OFFDIAGONALSTRI.

Algorithm 3.10 OFFDIAGONALSTRI($\mathbf{A}, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L, \mathbf{J}^R\right\}, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$)

```
Require: $\mathbf{A}, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L, \mathbf{J}^R\right\}, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} \in \mathbb{B}^{n,n}$, $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} = \mathrm{Diag}_{\mathbf{A}}\{\mathbf{A}^{-1}\}$
Ensure: $\mathbf{G} = \mathbf{A}^{-1} \in \mathbb{B}^{n,n}$
 1: $\mathbf{g}_{12} \leftarrow \mathbf{g}_{11}\mathbf{c}_2^R = \mathbf{g}_{11}[\mathbf{J}^R]_{12}$          handle first sub diagonal: 1op($\times$)
 2: for $i \leftarrow 2$ up to $n-1$ do                                   loop over nearly all block rows
 3:     $\mathbf{g}_{i,i-1} \leftarrow \mathbf{g}_{ii}\mathbf{c}_{i-1}^L = \mathbf{g}_{ii}[\mathbf{J}^L]_{i,i-1}$          sub diagonals: 1op($\times$)
 4:     $\mathbf{g}_{i,i+1} \leftarrow \mathbf{g}_{ii}\mathbf{c}_{i+1}^R = \mathbf{g}_{ii}[\mathbf{J}^R]_{i,i+1}$        super diagonals: 1op($\times$)
 5: $\mathbf{g}_{n,n-1} \leftarrow \mathbf{g}_{nn}\mathbf{c}_{n-1}^L = \mathbf{g}_{nn}[\mathbf{J}^L]_{n,n-1}$    handle last super diagonal: 1op($\times$)
 6: return $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
```

The Sweep based algorithm for determining the only block tridiagonal $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ is presented in Alg. (3.11) SWEEPINVERSETRI. This algorithm, precisely as for the full inverse version, combines the results of an upwards and downwards Gaussian elimination sweep. This is handled by two calls on line 1 and line 2 to Alg. (3.4) GAUSSELIMINATETRI which is optimized for producing only a block tridiagonal result. This is then followed by Alg. (3.7) DIAGONALS on line 3 that generates the block diagonal $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$, and finally only the off diagonal blocks that belong to the block tridiagonal of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ is produced with Alg. (3.10) OFFDIAGONALSTRI on line 4.

Algorithm 3.11 SWEEPINVERSETRI($\mathbf{A}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\} = \mathrm{Trid}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
 1: $\mathbf{D}^L, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L\right\} \leftarrow$ GAUSSELIMINATETRI($\mathbf{A}, 1, n$)
 2: $\mathbf{D}^R, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^R\right\} \leftarrow$ GAUSSELIMINATETRI($\mathbf{A}, n, 1$)
 3: $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} \leftarrow$ DIAGONALS($\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R$)
 4: $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\} \leftarrow$ OFFDIAGONALSTRI($\mathbf{A}, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L, \mathbf{J}^R\right\}, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$)
 5: return $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
```

A visualization of the sweep algorithm SWEEPINVERSETRI working on an example block tridiagonal $\mathbf{A}$ is given in Fig. 3.6. The figure shows how as a downwards and upwards Gaussian elimination sweep on $\mathbf{A}$, on the first and second column from the left, respectively, combines via Eq. (3.21) and Eqs. (3.23)–(3.25) to form $\mathbf{B}$ and $\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{G}\right\}$ in the third and fourth columns from the left, respectively.

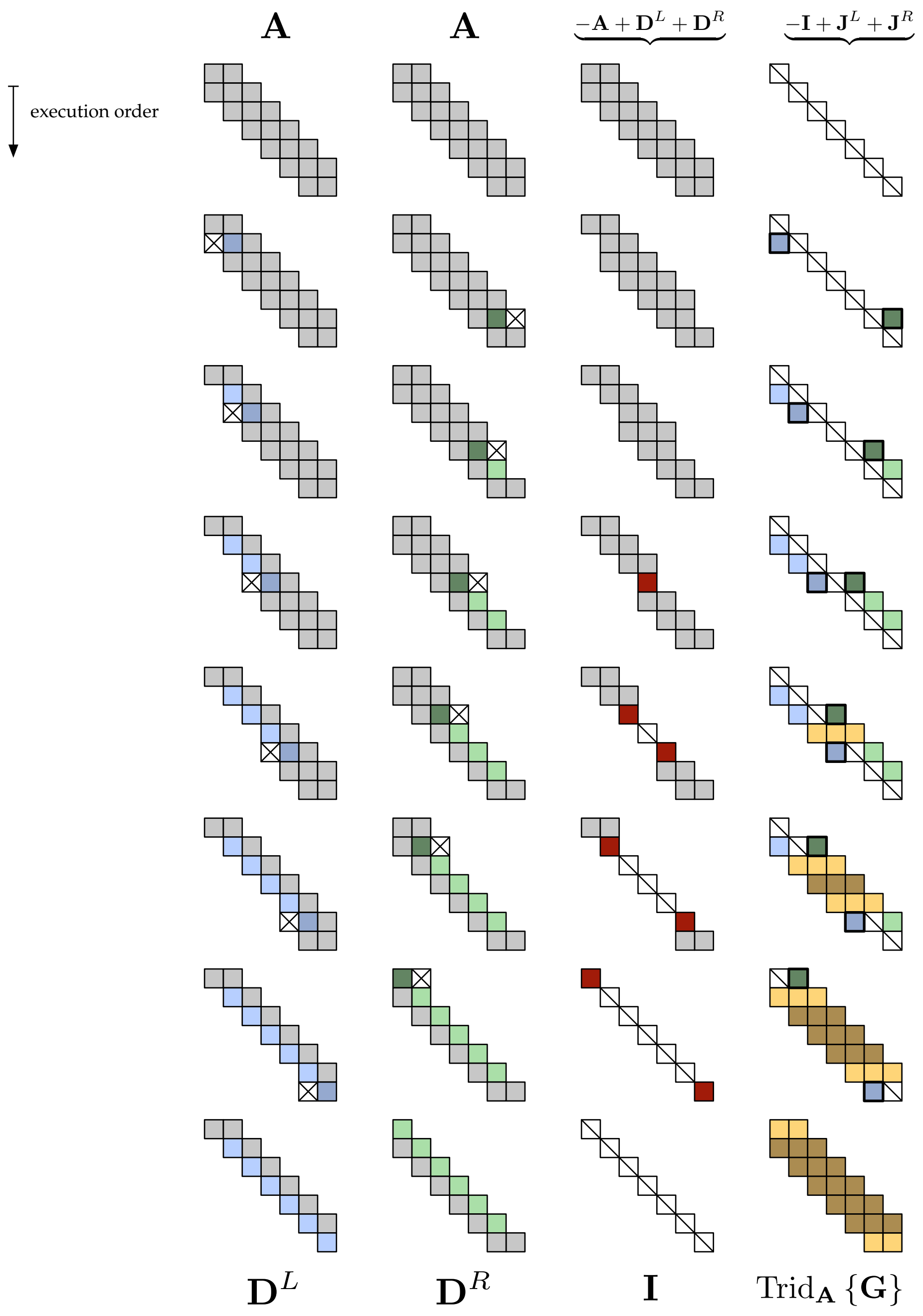
This figure, just as the one for SWEEPINVERSEFULL, should only be taken as a rough assessment of how the mathematics work, and not the algorithm itself, in that both the upwards and downwards Gaussian elimination sweeps are computed sequentially, and not in parallel. Furthermore, the construction of $\mathbf{G}$ is done first by sequentially determining $\mathrm{Diag}_{\mathbf{A}}\left\{\mathbf{G}\right\}$ from $\mathbf{a}_{11}$ down to $\mathbf{a}_{nn}$, and then the desired super and sub diagonal blocks.

3.2.3 Complexity Analysis

We now look to tabulate the matrix–matrix operation counts involved in both the full and the tridiagonal optimized versions of the sweep algorithm. Looking at Alg. (3.9) SWEEPINVERSEFULL we see that it is composed of two calls to GAUSSELIMINATEFULL on line 1 and line 2, before a call on line 3 to generate the diagonal blocks of $\mathbf{G}$ and a call on line 4 to generate the rest of $\mathbf{G}$. The only difference between SWEEPINVERSEFULL and its optimized counterpart Alg. (3.11) SWEEPINVERSETRI is that GAUSSELIMINATEFULL calls are replaced with calls to GAUSSELIMINATETRI and the off diagonals calculated are restricted to the sub and super diagonal blocks by calling OFFDIAGONALSTRI. Since we have the operation counts for both GAUSSELIMINATEFULL and GAUSSELIMINATETRI, we focus now on the algorithms for which we have not yet tabulated operation counts.

3.2.3.1 Diagonals

Both SWEEPINVERSEFULL and SWEEPINVERSETRI calls Alg. (3.7) in order to generate the block diagonals of $\mathbf{G}$. Looking at the algorithm, special cases are made out of the first and last diagonal blocks of $\mathbf{G}$, such that $\mathbf{g}_{11}$ and $\mathbf{g}_{nn}$, calculated on line 1 and line 4, are handled separate from the $n-2$ other diagonal blocks. These special cases each take a single LU factorization and something equivalent to a matrix multiplication to calculate. Otherwise, the other $n-2$ diagonal blocks are calculated on line 3, where each execution costs 2 addition operations, an LU factorization, and the equivalent of a matrix multiplication.



$\mathbf{A}$
$\mathbf{A}$
$\underbrace{-\mathbf{A}+\mathbf{D}^L+\mathbf{D}^R}$
$\underbrace{-\mathbf{I}+\mathbf{J}^L+\mathbf{J}^R}$
execution order
$\mathbf{D}^L$
$\mathbf{D}^R$
$\mathbf{I}$
$\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$

Figure 3.6: Visualizing the sweep method for calculating the block tridiagonal portion of the inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$ with 7 diagonal blocks. This figure should also be taken only as a rough assessment, much as for SWEEPINVERSEFULL. Note how as the algorithm progresses, we need not calculated undesired elements of $\mathbf{G}$ as in the case for GEINVERSETRI.

3.2.3.2 Off Diagonals

The off diagonal blocks are calculated via Alg. (3.8) OFFDIAGONALSFULL in the case of wanting all of $\mathbf{G}$, or via Alg. (3.10) OFFDIAGONALSTRI for determining $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$.

In the case of wanting all of $\mathbf{G}$, OFFDIAGONALSFULL will loop line 1 a total of $n-1$ times, determining the blocks below the diagonal with line 2 and the blocks above with line 4. Thus line 3 which calculates sub blocks is executed a total of $1, 2, \ldots, n-1$ times for a total of $\frac{1}{2}n(n-1)$ times (cf. Eq. (3.14)). Likewise, line 5 for determining blocks above the diagonal is executed a total of $n-1, n-2, \ldots, 1$ times for the same total of $\frac{1}{2}n(n-1)$.

Looking at the block tridiagonal optimized OFFDIAGONALSTRI, the for loops on line 2 and line 4 of OFFDIAGONALSFULL are replaced with single statements such that we only determine the desired super and sub diagonal blocks we need to complete $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$. Thus the single statements of line 3 and line 4 in OFFDIAGONALSTRI are only executed a total of $n-2$ times via the for loop on line 2, where the last couple of statements are handled on line 1 and line 5, each costing a single multiplication.

3.2.3.3 Results

Complexity Analysis for Sweep

Calculation	LU–factorizations op($\mathcal{LU}$)	Multiplications op($\times$)	Additions op($+$)
GAUSSELIMINATEFULL (up)	$n-1$	$\frac{1}{2}(n^2+n-2)$	$n-1$
GAUSSELIMINATEFULL (down)	$n-1$	$\frac{1}{2}(n^2+n-2)$	$n-1$
DIAGONALS	n	n	$2n-4$
OFFDIAGONALSFULL	0	n^2-n	0
SWEEPINVERSEFULL	$3n-2$	$2n^2+n-2$	$4n-6$
GAUSSELIMINATETRI (up)	$n-1$	$2n-2$	$n-1$
GAUSSELIMINATETRI (down)	$n-1$	$2n-2$	$n-1$
DIAGONALS	n	n	$2n-4$
OFFDIAGONALSTRI	0	$2n-2$	0
SWEEPINVERSETRI	$3n-2$	$7n-6$	$4n-6$

Table 3.2: This table illustrates the amount of basic operations performed in calculating either the full inverse $\mathbf{G}$ of $\mathbf{A}$, or only the block tridiagonal part of it $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$, using the sweep algorithms presented in this section. The second, third and fourth columns refer to the amount of basic matrix operations of LU–factorization, multiplication and addition involved in each algorithm. The term n is the total amount of diagonal blocks in $\mathbf{A} \in \mathbb{B}^{n,n}$.

Looking at the results of tabulating the operation counts for SWEEPIN-

VERSEFULL and SWEEPINVERSETRI in Table 3.2, we see that they have the same order $\mathcal{O}(3n)$ LU factorizations and $\mathcal{O}(4n)$ additions. Where they differ significantly is in the number of matrix multiplications, and we can see that the tridiagonal optimized version of Sweeps has linear order $\mathcal{O}(7n)$ complexity, while the calculation of all of $\mathbf{G}$ via Sweeps has quadratic order $\mathcal{O}(2n^2)$. This qualitative difference in complexity did not arise for the case of trying to improve standard block Gaussian elimination presented earlier.

Comparing with Gaussian elimination, the Sweeps method has a slightly better cost factor of 2 over GAUSSELIMINATEFULL's factor of 2.5 regarding matrix multiplications. Sweeps, however, is still a poorer choice for the task of determining all of $\mathbf{G}$ due to it having 3 times as many LU factorizations to perform. Using prefactor information from [29] for LU factorization and matrix–matrix multiplication, it can be shown the flop count for GAUSSELIMINATEFULL will be on the order of $\mathcal{O}(5.67n)$, while Sweeps will be on the order of $\mathcal{O}(6n)$. Thus Gaussian elimination is still preferred for calculating all of $\mathbf{G}$.

What is significant, however, is that when we only desire the block tridiagonal $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$, we can qualitatively improve the speed of execution by choosing SWEEPINVERSETRI over GEINVERSETRI, as SWEEPINVERSETRI has linear complexity in the operation count, while GEINVERSETRI's quadratic order multiplication count lends itself as the poorer choice.

Thus we have developed a method that can successfully calculate the block tridiagonal $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ significantly faster than standard block Gaussian elimination.

Chapter 4

Parallel Algorithms

If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?

Seymour Cray – Unverified

4.1 Parallel Computing

The architectures and implementations that immediately come to mind when the words parallel and computing are encountered in the same sentence may be many and varied, so we choose to begin this chapter with a short introduction to the parallel computing model we assume is the environment our algorithms work in.

4.1.1 Hardware Model

All computers are built from physical components known as hardware. In the parallel computing environment assumed for our algorithms, we find the basic building block to be a single machine which works in a sequential manner. This fundamental piece of the system is also known as the von Neumann model of a sequential computer, presented by John von Neumann in an incomplete report [30] in 1945, and conceptually by Konrad Zuse in a patent dating from 1936.

The von Neumann architecture is visualized in Fig. 4.1, where we see how a processor is connected to local memory storage via a system bus that is tasked

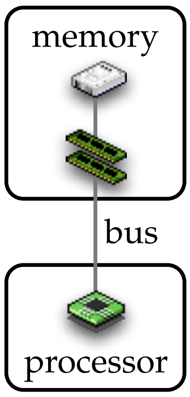


Figure 4.1: The sequential computers involved in our calculations correspond to the Von Neumann model, as visualized above, where the processor is connected to memory via a system bus that transfers data and instructions to the processor for computation and results back to memory for storage.

with routing data and instructions from memory to the processor for computation, and data back to memory for storage.

In order to move from the building block of the sequential computer to a parallel multi–computer, we take a series of von Neumann machines and connect them to a communications network, enabling them to communicate with each other.

Each von Neumann machine only has direct access to its own memory storage, and in order to access data on other machines it becomes necessary to employ a message–passing scheme that can transmit desired data as well as instructions between machines connected to the network. This scheme leads to the message–passing multiprocessor model of a parallel computer, and it is this machine we assume we are running our algorithms on.

One disadvantage of this parallel computing model is that as a programmer one has to explicitly manage the message passing and the sharing of data in memory among available processes, and this will be reflected in the algorithms presented. On the other hand, the programmer is relieved from the task of having to avoid managing multiple processes trying to access the same location in memory at the same time, which may lead to errors and undefined behavior of the program.

Another distinct advantage of programming for this hardware and memory model is that it is a model that is in wide use. This is due to its flexibility and cost effectiveness over other models, such as the *shared memory* multipro-

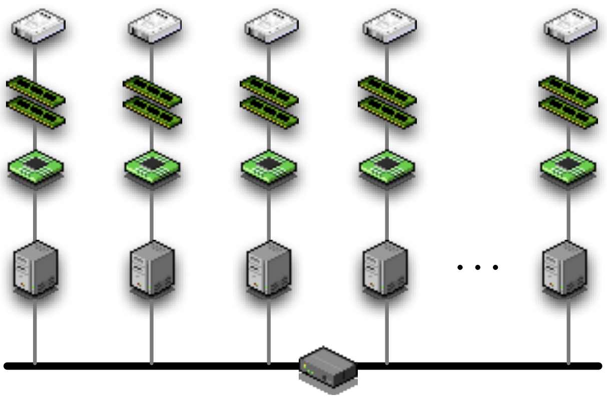


Figure 4.2: The parallel system assumed in this chapter corresponds to a network of Von Neumann computers all connected via a networking switch, as shown above. Every sequential von Neumann machine only has direct access to its own memory on its system bus, and inter–process communication via message–passing is needed for access to the memory of other processes as well as coordination of program execution.

cessor which are notoriously expensive and difficult to expand with additional hardware. Meanwhile, the message–passing multiprocessor composed of individual von Neumann machines can be more easily expanded upon by purchasing relatively cheap individual von Neumann machines as required and aggregating them to the message–passing network.

4.1.2 Memory Model and Data Distribution

Although we assume we are dealing with a multicomputer network architecture, where each connected machine only has direct access to its own physical memory, implementations exist where the collective memory storage facilities in the network are considered shared memory to which all machines have direct access. We, however, have chosen the simpler case where each computer on the network only has direct access to the memory physically associated with it, as seen in Fig. 4.1, relieving us from the earlier mentioned problems of managing memory access for the cost of managing message passing. Thus in order to access memory on other computers on the network, message passing must be employed.

In choosing not to use a shared memory model, we write off the possibility of transparently using all storage on the network directly as a large memory pool. However, we need not conform to the limited memory resources available to each machine on the network. We do this by implementing our algorithms to tackle the block tridiagonal matrix inversion by *distributing* data across the network.

This conceptually expands the available storage for solving our problems from being limited by the amount found on the von Neumann machine with the least amount of local memory to storage represented by pooling all memory on the network, where each machine has a responsibility to manage its own physical portion of the pool. In this way, each process is responsible only for a unique portion of data, and can access other data by posting a message to the appropriate owning process. In this way, we minimize data replication across all von Neumann machines that would otherwise be redundant and limit the problem sizes we can handle.

Thus we distribute ownership of our block matrices among the available processes such that each process has direct ownership of select portions of the block matrices involved, and must rely on the other processes to have their relevant portions in memory.

4.1.3 An Example Distribution

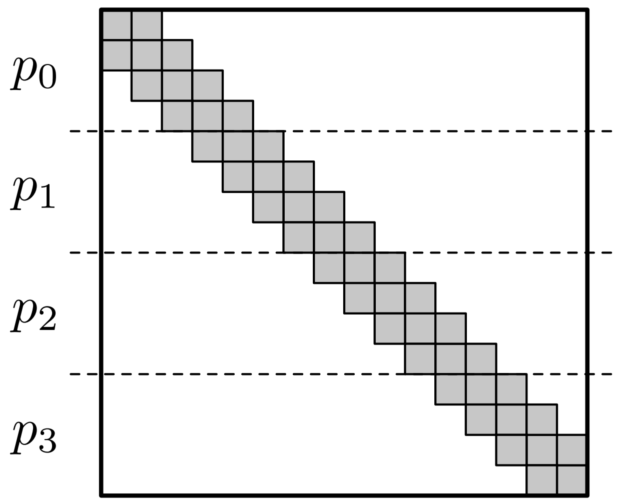


Figure 4.3: Visualizing a simple distribution of a block tridiagonal matrix of identical block dimensions among $\mathcal{P} = 4$ processes named $p_0, \ldots, p_3$.

Looking at Fig. 4.3, we can obtain a sense of how we distribute the content of a block tridiagonal matrix $\mathbf{A}$ among various processes. In the figure, we have $\mathcal{P} = 4$ processes sharing a block tridiagonal matrix with $n = 16$ diagonal blocks of equal size.

In general, the block matrix to be distributed need not be block tridiagonal, the distribution of data among processes need not be such that each process owns the same number of rows, and the block elements themselves are not required to be of equal dimension[1]. The issue of how to distribute the block matrices effectively for the sake of execution speed or memory will be discussed later. An example of a different distribution is given in Fig. 4.4

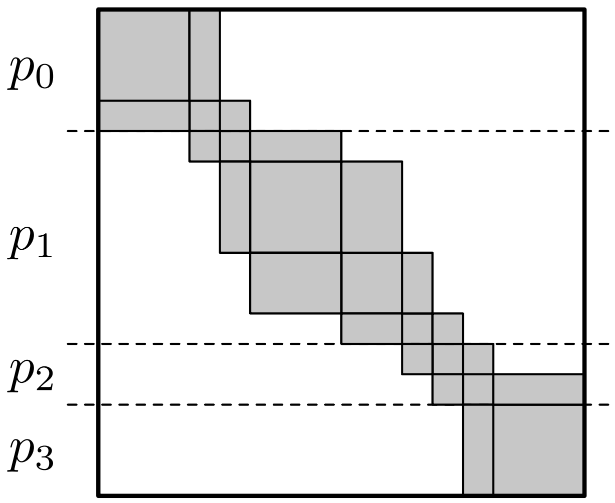


Figure 4.4: Visualizing a distribution of a block tridiagonal matrix with differing block dimensions among $\mathcal{P} = 4$ processes named $p_0, \dots, p_3$.

In the complexity analysis of the various algorithms, we will work under the assumption that the $\mathcal{P}$ different processes will own $m_0, m_1, \dots, m_{\mathcal{P}-1}$ rows of $\mathbf{A}$, in an ordered sequence from the top row of $\mathbf{A}$ to the bottom. This also means that all processes together own the total number of rows in $\mathbf{A}$:

$$\sum_{i=0}^{\mathcal{P}-1} m_i = n. \tag{4.1}$$

This expression will be used later in our complexity analysis.

[1]They are only required to be of dimension such that the matrix may belong to $\mathbb{B}^{r,s}$.

4.1.4 Some Assumptions

In the methods presented in this chapter, we not only assume the block matrices involved are distributed to begin with, but that each is distributed in the same manner, such that the ownership arrangement of $\mathbf{A}$ will be identical to that of the calculated portion of the inverse $\mathbf{G}$, as well as the LU factors $\mathbf{L}$ and $\mathbf{U}$.

In this way we can depend on the fact that if $\mathbf{a}_{ij}$ is owned by process p_i, that same process will also own $\mathbf{l}_{ij}$, $\mathbf{u}_{ij}$ and $\mathbf{g}_{ij}$, for example. Furthermore, as we distribute entire *rows* among processes, every block on a certain row of $\mathbf{A}$, $\mathbf{G}$, $\mathbf{L}$ or $\mathbf{U}$ will be found on the same process.

It should be noted that other processes might have copies of blocks belonging to other processes used locally in other calculations. These copies are retrieved via message–passing, but the ultimately the process responsible for storing a nontrivial[1] block on the (i, j)th location will be process owning the ith row, p_i.

4.2 Parallel Sweep

We saw in Sec. 3.2 a definition of a sweep based algorithm that could determine the block tridiagonal portion of the inverse in an amount of time that scales linearly with the block dimension n of $\mathbf{A}$. In this section, we present a version of this algorithm that has been adapted to the distributed memory layout of $\mathbf{A}$ across a set of $\mathcal{P}$ processes.

4.2.1 Description

The first adaptation taken in order to parallelize the sweep algorithm is for it to handle the distributed nature of $\mathbf{A}$. This means that the Gaussian elimination sweeps that proceed from the top and bottom will need to be "handed off" to neighboring processes as they sweep through $\mathbf{A}$. As we potentially have multiple processes handling the storage of $\mathbf{A}$, we can now perform these two sweeps simultaneously, allowing for the eventuality of a downwards sweep meeting a process currently performing an upwards sweep, or vice versa.

This should immediately yield a theoretical possible speedup of 2, as we can have this proceed perfectly in parallel, as the only communication needed is to hand off the sweeping task to a neighbor.

[1]All–zero blocks $\mathbf{0}_{ij}$ are not explicitly stored.

A consequence of the now parallelized elimination sweeps is that once a process has performed both an upwards and downwards sweep on its owned elements, it can then immediately calculate its owned portion of the inverse **G** independently of all other processes.

We can see a visualization of the parallelized sweeps working on an example 7×7 block matrix **A** in Fig. 4.5, with much similarity to the original figure for the sweep algorithm. We use the same legend as presented in Fig. 3.1. Here we have divided **A** over 3 processes separated by the dashed lines, and inter–process communication is indicated by arrows.

We see how each process hands off its elimination sweep to its nearest neighbor, and in this case, we see how the upwards sweep has to wait for the downwards sweep to complete in the middle process before being allowed to continue. A middle process waiting for sweeps to arrive will handle the first message request that comes along to continue the elimination, while the other elimination pass will have to wait.

We can also see that although the sweeps will overlap at some point, the blocks of the inverse will not be computed until all sweeps have terminated on the owning process before construction of **G** begins.

4.2.2 Algorithm

We present here the process of determining $\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{G}\right\}$ via the new, parallelized sweep method in an algorithmic framework. Having many similarities to the sequential sweep method presented earlier, we will find the main differences lie in the algorithm for handling the Gaussian elimination.

4.2.2.1 Gaussian Elimination

The algorithm that handles both the upwards and downwards Gaussian elimination sweeps is found in Alg. (4.1) GAUSSELIMINATEPARALLEL. The first feature one may notice is that the algorithm is split in two cases, such that we can handle a sequential execution of elimination sweeps in case we only have $\mathcal{P} = 1$ processes available. This is done by line 1, which will call our sequential Gaussian elimination routines one after the other, as was the case for Alg. (3.11) SWEEPINVERSETRI, earlier.

Should we have more than one process available, $\mathcal{P} > 1$, we then execute the parallel case of the algorithm, on line 4. This parallel section of code is then subdivided into tasks, where a task for the process that owns the topmost row is on line 5, a task for the process that owns the bottommost row on line 24,

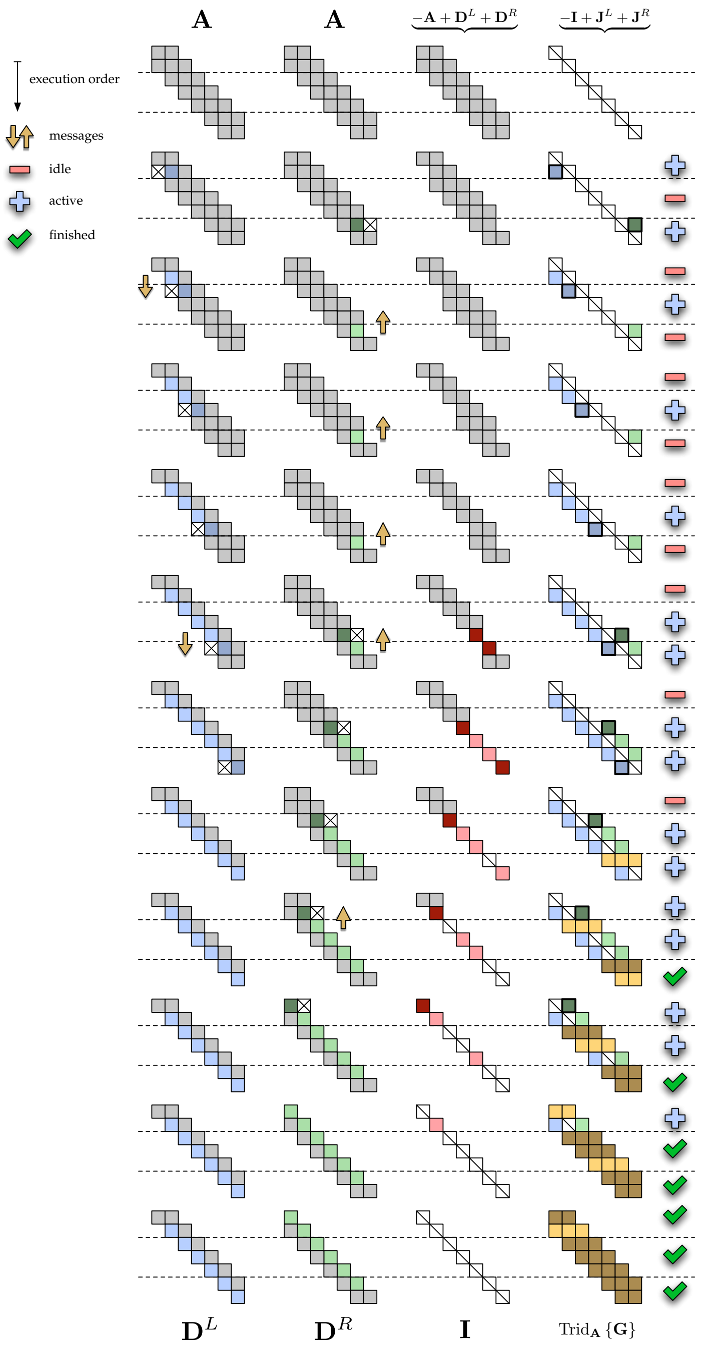

A
A
$\underbrace{-A + D^L + D^R}$
$\underbrace{-I + J^L + J^R}$
execution order
messages
idle
active
finished
D^L
D^R
I
$\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$

Figure 4.5: Visualizing the parallelized sweep method for calculating the block tridiagonal inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$ with 7 diagonal blocks.

Algorithm 4.1 GaussEliminateParallel($\mathbf{A}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
 1: if $\mathcal{P} = 1$ then                                              run in serial
 2:   $\mathbf{D}^L, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, 1, n$)
 3:   $\mathbf{D}^R, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^R\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, n, 1$)
 4: else                                                                 run in parallel
 5:   if myPID $= 0$ then                        top process starts the downwards sweep
 6:     $\mathbf{D}^L, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, top = 1, bot$)
 7:     $request_{\text{send}} \leftarrow$ isend $\mathbf{a}_{bot,bot+1}$, $\mathbf{d}^L_{bot,bot}$ to 1
 8:     $request_{\text{recv}} \leftarrow$ irecv $\mathbf{a}_{bot+1,bot}$, $\mathbf{d}^R_{bot+1,bot+1}$ from 1
 9:     wait for $request_{\text{recv}}$
10:     $\mathbf{D}^R, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^R\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, bot + 1, top = 1$)
11:   if myPID $\neq 0$ and myPID $\neq \mathcal{P} - 1$ then               other processes listen
12:     $request_{\text{above}} \leftarrow$ irecv $\mathbf{a}_{top-1,top}$, $\mathbf{d}^L_{top-1,top-1}$ from myPID$-1$
13:     $request_{\text{below}} \leftarrow$ irecv $\mathbf{a}_{bot+1,bot}$, $\mathbf{d}^R_{bot+1,bot+1}$ from myPID$+1$
14:     $sweep_{\text{down}}, sweep_{\text{up}} \leftarrow$ false
15:     while not $sweep_{\text{down}}$ or not $sweep_{\text{up}}$ do
16:       if $request_{\text{above}}$ and not $sweep_{\text{down}}$ then
17:         $\mathbf{D}^L, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, top - 1, bot$)
18:         $sweep_{\text{down}} \leftarrow$ true
19:         $request_{\text{down}} \leftarrow$ isend $\mathbf{a}_{bot,bot+1}$, $\mathbf{d}^L_{bot,bot}$ to myPID$+1$
20:       if $request_{\text{below}}$ and not $sweep_{\text{up}}$ then
21:         $\mathbf{D}^R, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^R\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, bot + 1, top$)
22:         $sweep_{\text{up}} \leftarrow$ true
23:         $request_{\text{up}} \leftarrow$ isend $\mathbf{a}_{top,top-1}$, $\mathbf{d}^R_{top,top}$ to myPID$-1$
24:   if myPID $= \mathcal{P} - 1$ then              bottom process starts the upwards sweep
25:     $\mathbf{D}^R, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^R\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, bot = n, top$)
26:     $request_{\text{send}} \leftarrow$ isend $\mathbf{a}_{top,top-1}$, $\mathbf{d}^R_{top,top}$ to myPID$-1$
27:     $request_{\text{recv}} \leftarrow$ irecv $\mathbf{a}_{top-1,top}$, $\mathbf{d}^L_{top-1,top-1}$ from myPID$-1$
28:     wait for $request_{\text{recv}}$
29:     $\mathbf{D}^L, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L\right\} \leftarrow$ GaussEliminateTri($\mathbf{A}, top - 1, bot = n$)
30: return $\mathbf{D}^L, \mathbf{D}^R, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L, \mathbf{J}^R\right\}$
```

and a task for all other processes on line 11. In the code, we recall that *top* and *bot* are constants available to each process dictating which is their topmost and bottommost owned row of $\mathbf{A}$.

The topmost process immediately calculates a downwards Gaussian elimination sweep on its owned portion of $\mathbf{A}$ on line 6, before posting an *immediate* send of results to its neighbor such that it may start work on the downwards sweep on line 7. This transmission is immediate in the sense that the command returns control to the sending process regardless of whether or not the receiving process has acknowledged receipt of the message. This is necessary in order to avoid *deadlock*[1], as the neighbor process might have completed an upwards sweep and is trying to send data to the top process.

The immediate send is followed by an *immediate* receive command on line 8 that will wait for a message from the neighbor carrying information to complete the upwards Gaussian elimination sweep. Though not strictly necessary, this receive command is made immediate to match the immediate send, and we wait for it to complete. Alternatively, we could use a blocking regular receive command **recv**. When reception is completed, the upwards Gaussian elimination sweep has finished its pass through the lower processes, and the top process can complete the full pass through $\mathbf{A}$ by executing line 10. This entire behavior is mirrored by the bottom process on line 24.

The task the middle processes have to handle on line 11 is slightly more complex. They risk receiving a message from either a neighbor process above them or below them, but which comes first is not known deterministically. Thus a form of listening strategy has to be employed, where an incoming message has to be handled on a first–come first–served basis.

The listening by the middle processes is carried out by executing the immediate receive commands on line 12 and line 13, which listen to incoming messages from nearest neighbor processes. We furthermore define two flags on line 14 that tell us whether or not the process has completed a downwards or upwards sweep. The process then enters a while loop on line 15 that loops until both elimination sweeps have passed it.

The while loop effectively polls the receive requests we are listening for to complete, such that we can perform an elimination sweep. The requests, however, should only be acted on if the elimination sweep they represent has not already been handled by the process, which we can check by the flags defined on line 14. The code handling an incoming downwards Gaussian elimination

[1]Deadlock occurs when messages between processes are unable to complete because they are blocked from completion as they wait indefinitely for other communication processes to complete [31].

sweep is on line 16 while the counterpart upwards sweep is handled on line 20.

In the case of a downwards Gaussian elimination sweep arriving at the listening process for the first time, the message receipt request flag evaluates to true, while the sweep completion flag evaluates to false, and the code enclosed by the case on line 16 executes. This is encompassed by a downwards elimination sweep handled by GAUSSELIMINATETRI on line 17. We then set the corresponding sweep completion flag to **true** so that we do not re–execute a downwards sweep on repeated polling of the listening requests. Finally, an immediate send of data is performed on line 19 to a neighbor process owning rows below that will continue the downwards Gaussian sweep on $\mathbf{A}$. The message is immediate, such that the process can return to listening for and handling an incoming request that might be waiting, and we can avoid a situation of deadlock. This entire behavior is mirrored for the case of an upwards Gaussian elimination by the case on line 20.

4.2.2.2 Constructing the Inverse

As for the sequential version of sweep, once the Gaussian elimination sweeps have taken place, we are now free to construct the desired inverse. The difference between the sequential and parallel version of sweep, is that individual processes may have completed both Gaussian elimination passes earlier, and are free to construct their part of $\mathbf{G}$, even though the elimination sweeps may not have terminated for other processes.

For those processes that have terminated the elimination sweeps, we have Alg. (4.2) DIAGONALSPARALLEL that takes care of determining the diagonal blocks of $\mathbf{G}$, before we go on to determining the off diagonal blocks.

Algorithm 4.2 DIAGONALSPARALLEL$(\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R)$

```
Require: $\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R \in \mathbb{B}^{n,n}$
Ensure: $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} = \mathrm{Diag}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: if row $1$ is mine then
2:     $\mathbf{g}_{11} \leftarrow (\mathbf{d}^R_{11})^{-1}$                    $1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
3: for $i \leftarrow 2$ up to $n-1$ do                    *determine "inner" diagonal blocks*
4:     if row $i$ is mine then
5:         $\mathbf{g}_{ii} \leftarrow (-\mathbf{a}_{ii} + \mathbf{d}^L_{ii} + \mathbf{d}^R_{ii})^{-1}$          $2\mathsf{op}(+), 1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
6: if row $n$ is mine then
7:     $\mathbf{g}_{nn} \leftarrow (\mathbf{d}^L_{nn})^{-1}$                    $1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
8: return $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$
```

The algorithm takes directly after Alg. (3.7) DIAGONALS, developed for the serial version of Sweeps, only in that some control code is inserted that assures a statement constructing the diagonal block $\mathbf{g}_{ii}$ will only be executed if the ith row belongs to the calling process.

The off diagonal blocks are handled by Alg. (4.3) OFFDIAGONALSPARALLEL, and takes directly after Alg. (3.10) OFFDIAGONALSTRI, developed for the serial version of Sweeps. The difference here is that control code is added to ensure that statements constructing the off diagonal blocks $\mathbf{g}_{i,i-1}$ or $\mathbf{g}_{i,i+1}$ are only executed if the ith row belongs to the calling process.

Algorithm 4.3 OFFDIAGONALSPARALLEL($\mathbf{A}, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L, \mathbf{J}^R\right\}, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\}$)

```
Require: $\mathbf{A}, \mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{J}^L, \mathbf{J}^R\right\}, \mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} \in \mathbb{B}^{n,n}$, $\mathrm{Diag}_{\mathbf{A}}\{\mathbf{G}\} = \mathrm{Diag}_{\mathbf{A}}\{\mathbf{A}^{-1}\}$
Ensure: $\mathbf{G} = \mathbf{A}^{-1} \in \mathbb{B}^{n,n}$
1: if row $1$ is mine then
2:     $\mathbf{g}_{12} = \mathbf{g}_{11}\mathbf{c}_2^R = \mathbf{g}_{11}[\mathbf{J}^R]_{12}$                         1op(×)
3: for $i = 2 \ldots n-1$ do                                              loop over all block rows
4:     if row $i$ is mine then
5:         $\mathbf{g}_{i,i-1} = \mathbf{g}_{ii}\mathbf{c}_{i-1}^L = \mathbf{g}_{ii}[\mathbf{J}^L]_{i,i-1}$        sub–diagonals: 1op(×)
6:         $\mathbf{g}_{i,i+1} = \mathbf{g}_{ii}\mathbf{c}_{i+1}^R = \mathbf{g}_{ii}[\mathbf{J}^R]_{i,i+1}$      super–diagonals: 1op(×)
7: if row $n$ is mine then
8:     $\mathbf{g}_{n,n-1} = \mathbf{g}_{nn}\mathbf{c}_{n-1}^L = \mathbf{g}_{nn}[\mathbf{J}^L]_{n,n-1}$               1op(×)
9: return $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
```

Finally, the algorithm that performs the steps necessary to calculating the block tridiagonal $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ in parallel, is presented in Alg. (4.4) SWEEPINVERSEPARALLEL. The algorithm calls GAUSSELIMINATEPARALLEL on line 1, which takes care of commencing both an upwards and downwards Gaussian elimination sweep in parallel. It also handles the serial case for when $\mathcal{P} = 1$. Processes that have completed both an upwards and downwards sweep proceed immediately to constructing diagonal blocks on line 2, before computing off diagonal blocks on line 3. Each process only computes blocks of the inverse within its domain of responsibility for $\mathbf{G}$, and thus the result is distributed across processes in the same manner as $\mathbf{A}$.

4.2.3 Complexity

In analyzing Alg. (4.4) for complexity, we see that we can break it down into 3 distinct phases, as given by the Gaussian elimination on line 1 by Alg. (4.1) GAUSSELIMINATEPARALLEL, determination of the diagonal blocks on line 2

Algorithm 4.4 SweepInverseParallel($\mathbf{A}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} = \text{Trid}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: $\mathbf{D}^L, \mathbf{D}^R, \text{Trid}_{\mathbf{A}}\{\mathbf{J}^L, \mathbf{J}^R\} \leftarrow$ GaussEliminateParallel($\mathbf{A}$)
2: $\text{Diag}_{\mathbf{A}}\{\mathbf{G}\} \leftarrow$ DiagonalsParallel($\mathbf{A}, \mathbf{D}^L, \mathbf{D}^R$)
3: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} \leftarrow$ OffDiagonalsParallel($\mathbf{A}, \text{Trid}_{\mathbf{A}}\{\mathbf{J}^L, \mathbf{J}^R\}, \text{Diag}_{\mathbf{A}}\{\mathbf{G}\}$)
4: return $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
```

by Alg. (4.2) DiagonalsParallel, and the determination of the off diagonal blocks on line 3 by Alg. (4.3) OffDiagonalsParallel.

Starting with the Gaussian elimination phase, we can first identify the fact that the basic operations carried out are done by the sequential algorithm GaussEliminateTri, for which we have already determined complexities earlier on. We can then relegate ourselves to determining how many rows each process will call GaussEliminateTri with.

Looking at process p_0, which owns the first m_0 rows of $\mathbf{A}$, it performs two sweeps, where the first downwards sweep on line 6 covers m_0 rows. The second sweep on line 10 covers $m_0 + 1$ rows. This is similar for the final process $p_{\mathcal{P}-1}$ but with $m_{\mathcal{P}-1}$ and $m_{\mathcal{P}-1} + 1$ rows, instead. A middle process p_i, owning m_i rows, also performs two sweeps, on line 17 and line 21, but where each sweep covers $m_i + 1$ rows.

We can then calculate the total operation counts, using the tabulated complexities for GaussEliminateTri in Table 3.1, and the LU decomposition count becomes for the down sweeping eliminations become

$$\begin{aligned} \mathsf{op}(\mathcal{LU}) &= \sum_{i=0}^{\mathcal{P}-2}((m_i + 1) - 1) + (m_{\mathcal{P}-1} - 1) \\ &= \sum_{i=0}^{\mathcal{P}-1} m_i - 1 \\ &= n - 1 \end{aligned} \tag{4.2}$$

and likewise for the upwards sweeping eliminations, giving us $2(n-1)$ LU factorizations for GaussEliminateParallel. As the complexity for additions is the same in GaussEliminateTri as for LU factorizations, we also get $2(n-1)$ matrix additions for GaussEliminateParallel. Multiplications, on the other hand, have a factor 2 on top of this, giving us $4(n-1)$ multiplications.

Moving on to the calculation of diagonal blocks via DiagonalsParallel, we can see that a middle process p_i owning m_i middle rows will have to execute $m_i\mathsf{op}(\mathcal{LU})$, $m_i\mathsf{op}(\times)$ and $2m_i\mathsf{op}(+)$ operations. A corner process will have to execute the same number of LU decompositions and multiplications, but saves a little with respect to addition, in only having to calculate a total of $2(m_0 - 1)\mathsf{op}(+)$ and $2(m_{\mathcal{P}-1} - 1)\mathsf{op}(+)$, for the upper and lower process, respectively. This sums up to be $n\mathsf{op}(\mathcal{LU})$, $n\mathsf{op}(\times)$ and $2(n-2)\mathsf{op}(+)$ operations for DIAGONALSPARALLEL.

Finally, we can count the operations for the off diagonal blocks obtained with OFFDIAGONALSPARALLEL. A middle process p_i owning m_i rows executes a total of $2m_i\mathsf{op}(\times)$ to calculate its owned off diagonal blocks, while the corner processes executes a total of $(2m_0-1)\mathsf{op}(\times)$ and $(2m_{\mathcal{P}-1}-1)\mathsf{op}(\times)$, for the upper and lower process, respectively. This adds up to a total of $2(n-1)\mathsf{op}(\times)$ in total for all processes.

Complexity Analysis for parallelized Sweep

Calculation	LU–factorizations $\mathsf{op}(\mathcal{LU})$	Multiplications $\mathsf{op}(\times)$	Additions $\mathsf{op}(+)$
GAUSSELIMINATEPARALLEL	$2n-2$	$4n-4$	$2n-2$
DIAGONALSPARALLEL	n	n	$2n-4$
OFFDIAGONALSPARALLEL	0	$2n-2$	0
SWEEPINVERSEPARALLEL	$3n-2$	$7n-6$	$4n-6$

Table 4.1: This table illustrates the amount of basic operations performed in calculating the block tridiagonal inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ using the parallelized sweep algorithms presented in this section. The second, third and fourth columns refer to the amount of basic matrix operations of LU–factorization, multiplication and addition involved in each algorithm. The term n is the total amount of diagonal blocks in $\mathbf{A} \in \mathbb{B}^{n,n}$. Interesting to note is that the overall complexity is identical to that of Alg. (3.11) SWEEPINVERSETRI.

Finally, we tabulate the complexity for SWEEPINVERSEPARALLEL in Table 4.1, and as we may expect, the overall complexity is identical to that of SWEEPINVERSETRI. This is because the only significant difference between the algorithms is the incorporation of message passing to "hand–off" a sweep between neighbor processes and the handling of only computing elements of $\mathbf{G}$ owned by the calling process, while no change exists in the the numerical aspects of the code.

4.3 Block Cyclic Reduction

Block cyclic reduction, or BCR, is born out of the technique called *cyclic reduction*, which was developed many decades ago by Gene Golub to deal with the scalar tridiagonal systems that arise in solving finite element discretizations of the Poisson equation in 2D. A historical treatment of its development is given by Gander and Golub in [32]. In this paper, we consider the case of *block* cyclic reduction, where the scalar elements of traditional cyclic reduction is replaced with matrix blocks.

4.3.1 Description

Block cyclic reduction takes as its starting point our now familiar block tridiagonal matrix $\mathbf{A}$ and performs, in parallel, operations that effectively eliminate the odd–numbered indices of the unknowns in what we call the *reduction* phase. As we are working with block matrices, we effectively eliminate the odd–numbered block columns and rows of $\mathbf{A}$. This continues until we remain with a single final block. Once the inverse corresponding to this final block is calculated, we proceed with a *production* phase in order to calculate the rest of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$.

The BCR method is visualized in Fig. 4.6, where we see first a reduction phase bring $\mathbf{A}$ down to a single block $\mathbf{a}_{kk}^{\mathrm{BCR}}$, from which we can calculate the first block of the inverse, $\mathbf{g}_{kk}^{\mathrm{BCR}}$. From there, the method proceeds through a production phase in order to eventually yield the block tridiagonal part of the inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$. As can be seen on the resulting inverse, some undesired elements of the inverse need calculating, and is an effect of the *fill–in* characteristics of the method, as we will see later.

4.3.1.1 Reduction

The reduction phase begins with the complete block tridiagonal $\mathbf{A}$ and seeks to eliminate the odd–numbered rows/columns by a series of row operations. This is accomplished in the following manner, we we take our augmented matrix Eq. (2.68), where rows i, k are odd and row j is even:

$$\left(\begin{array}{cccccc|cccc}
\ddots & \ddots & \ddots & & & & \ddots & & & \\
 & \mathbf{a}_{ih} & \mathbf{a}_{ii} & \mathbf{a}_{ij} & & & & \mathbf{i}_{ii} & & \\
 & & \mathbf{a}_{ji} & \mathbf{a}_{jj} & \mathbf{a}_{jk} & & & & \mathbf{i}_{jj} & \\
 & & & \mathbf{a}_{kj} & \mathbf{a}_{kk} & \mathbf{a}_{kl} & & & & \mathbf{i}_{kk} \\
 & & & & \ddots & \ddots & \ddots & & & & \ddots
\end{array}\right). \tag{4.3}$$

We eliminate the coupling element $\mathbf{a}_{ji}$ by a row operation involving row i the factor $\mathbf{l}_{ji} = -\mathbf{a}_{ji}\mathbf{a}_{ii}^{-1}$. Likewise, we eliminate $\mathbf{a}_{jk}$ by a row operation involing row k and the factor $\mathbf{l}_{jk} = -\mathbf{a}_{jk}\mathbf{a}_{kk}^{-1}$. It is no coincidence that these factors are

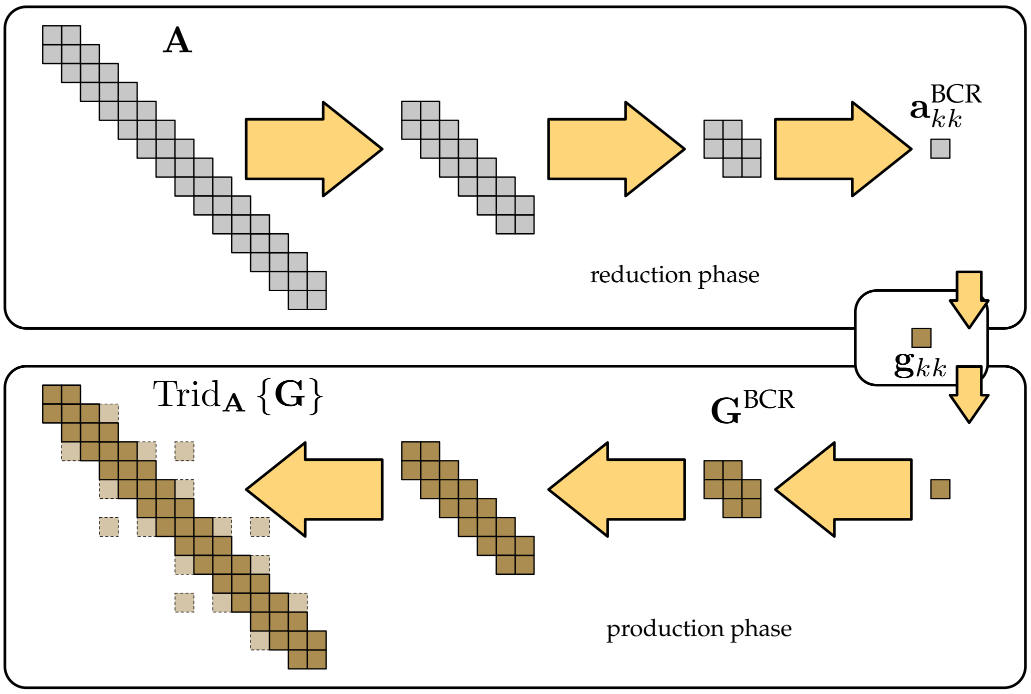

$\mathbf{A}$
$\mathbf{a}_{kk}^{\text{BCR}}$
reduction phase
$\mathbf{g}_{kk}$
$\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
$\mathbf{G}^{\text{BCR}}$
production phase

Figure 4.6: The block cyclic reduction method eliminates half the number of block unknowns in each reduction step, until it remains with a single block $\mathbf{a}_{kk}^{\text{BCR}}$. This block can then easily be inverted to determine the first block of the inverse, $\mathbf{g}_{kk}$. A series of production steps is then undertaken using stored LU factors from the reduction phase, that in the end produce our desired block tridiagonal $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$. A number blocks off the block tridiagonal is also calculated but this number is linear $\mathcal{O}(n)$ with respect to the number of diagonal blocks in $\mathbf{A}$.

lower case $\mathbf{L}$s, since Golub [32, 33] recognized that cyclic reduction is equivalent to unpivoted Gaussian elimination on a permuted system, and the computed factors are related to an LU factorization of this permuted system. The $\mathbf{U}$ factors can be stored for later reconstruction of the inverse on the odd rows and we have $\mathbf{u}_{ij} = -\mathbf{a}_{ii}^{-1}\mathbf{a}_{ij}$ and $\mathbf{u}_{kj} = -\mathbf{a}_{kk}^{-1}\mathbf{a}_{kj}$. Doing this, we obtain

$$\left(\begin{array}{ccccccc|ccccc} \ddots & \ddots & \ddots & & & & & \ddots & & & & \\ & \mathbf{a}_{ih} & \mathbf{a}_{ii} & \mathbf{a}_{ij} & & & & & \mathbf{i}_{ii} & \mathbf{u}_{ij} & & \\ & \mathbf{a}_{jh}^{\mathrm{BCR}} & \mathbf{0}_{ji} & \mathbf{a}_{jj}^{\mathrm{BCR}} & \mathbf{0}_{jk} & \mathbf{a}_{jl}^{\mathrm{BCR}} & & & \mathbf{l}_{ji} & \mathbf{i}_{jj} & \mathbf{l}_{jk} & \\ & & & \mathbf{a}_{kj} & \mathbf{a}_{kk} & \mathbf{a}_{kl} & & & & \mathbf{u}_{kj} & \mathbf{i}_{kk} & \\ & & & & \ddots & \ddots & \ddots & & & & & \ddots \end{array}\right) \tag{4.4}$$

where we have implicitly "stored" the $\mathbf{U}$ factors on the right hand side of the augmented matrix, and the updated elements on the left hand side, denoted by $\mathbf{a}^{\mathrm{BCR}}$ are

$$\mathbf{a}_{jj}^{\mathrm{BCR}} = \mathbf{a}_{jj} + \mathbf{l}_{ji}\mathbf{a}_{ij} + \mathbf{l}_{jk}\mathbf{a}_{kj}, \tag{4.5}$$
$$\mathbf{a}_{jh}^{\mathrm{BCR}} = \mathbf{l}_{ji}\mathbf{a}_{ih}, \tag{4.6}$$
$$\mathbf{a}_{jl}^{\mathrm{BCR}} = \mathbf{l}_{jk}\mathbf{a}_{kl}. \tag{4.7}$$

On the elimination of the odd–numbered rows, we see that the remaining even–numbered rows are only coupled to themselves, and we can imagine that we have *reduced* our original block matrix $\mathbf{A}$ to one with half the number of rows and columns: $\mathbf{A}^{\mathrm{BCR}}$. Looking at Eq. (4.4), we can ignore the odd numbered rows, and we then are left with:

$$\left(\begin{array}{ccccc|ccc} \ddots & \ddots & \ddots & & & \ddots & & \\ & \mathbf{a}_{jh}^{\mathrm{BCR}} & \mathbf{a}_{jj}^{\mathrm{BCR}} & \mathbf{a}_{jl}^{\mathrm{BCR}} & & & \mathbf{i}_{jj} & \\ & & \ddots & \ddots & \ddots & & & \ddots \end{array}\right). \tag{4.8}$$

This elimination of odd–numbered rows is highly parallel, in that each update of the remaining even rows can be done fully in parallel. The subsequent reconstruction of the odd rows is equally parallel, as the computation of the $\mathbf{U}$ factors mirrors that of the $\mathbf{L}$ factors. On the other hand, the "new" blocks $\mathbf{a}_{jh}^{\mathrm{BCR}}$ and $\mathbf{a}_{jl}^{\mathrm{BCR}}$ can be considered *fill–in*, and will later need elimination.

This process of reduction continues until we are left with only one row, namely $\mathbf{a}_{kk}^{\mathrm{BCR}}$, where k denotes the row/column we finally reduce to. When this is the case, we are left with the augmented system

$$\left(\begin{array}{c|c} \mathbf{a}_{kk}^{\mathrm{BCR}} & \mathbf{i}_{kk} \end{array}\right). \tag{4.9}$$

from which we can directly calculate the first block of the inverse, namely $\mathbf{g}_{kk}$:

$$\mathbf{g}_{kk} = (\mathbf{a}_{kk}^{\mathrm{BCR}})^{-1} \tag{4.10}$$

The reduction phase of BCR is visualized in Fig. 4.7, where we start with the full augmented matrix $[\mathbf{A}|\mathbf{I}]$ on the left, and we visualize the "active" block rows in an elimination tree on the right, where every arrow indicates a row operation.

4.3.1.2 Corner Production

Once we have the first block of the inverse, $\mathbf{g}_{kk}$, we begin to work backwards, and start the *production* phase. Where before we sought to eliminate the odd–numbered rows of $\mathbf{A}$, we now have to start *producing* about the kth row. The types of productions fall into two categories: a *corner* production, and a *center* production, for which we first describe the corner case here.

A corner production can be likened to a one–sided production, since we are only expanding our inverse $\mathbf{G}$ in one direction from an originating row k_{from} to towards the row to be produced k_{to}. This can either be in the upwards or downwards direction in the case $k_{\mathrm{to}} < k_{\mathrm{from}}$ or $k_{\mathrm{to}} > k_{\mathrm{from}}$, respectively.

The simplest BCR case that leads to a corner production can be taken in the following augmented 2×2 block system:

$$\left(\begin{array}{cc|cc} \mathbf{a}_{11} & \mathbf{a}_{12} & \mathbf{i}_{11} & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & & \mathbf{i}_{22} \end{array} \right) \tag{4.11}$$

A single reduction operation then leads to the following system

$$\left(\begin{array}{cc|cc} \mathbf{a}_{11} & \mathbf{a}_{12} & \mathbf{i}_{11} & \\ & \mathbf{s} & \mathbf{l}_{21} & \mathbf{i}_{22} \end{array} \right), \tag{4.12}$$

where $\mathbf{s} = \mathbf{a}_{22} + \mathbf{l}_{21}\mathbf{a}_{12}$ is the Schur complement using the computed LU factor $\mathbf{l}_{21} = -\mathbf{a}_{21}\mathbf{a}_{11}^{-1}$. We have also determined its corresponding factor $\mathbf{u}_{12} = -\mathbf{a}_{11}^{-1}\mathbf{a}_{12}$ during the reduction phase which will be needed during the production step. With the Schur complement determined, we can calculate the inverse $\mathbf{g}_{22}$ on the second row by inverting $\mathbf{s}$ and with this done, we are in the situation prior to a corner production step.

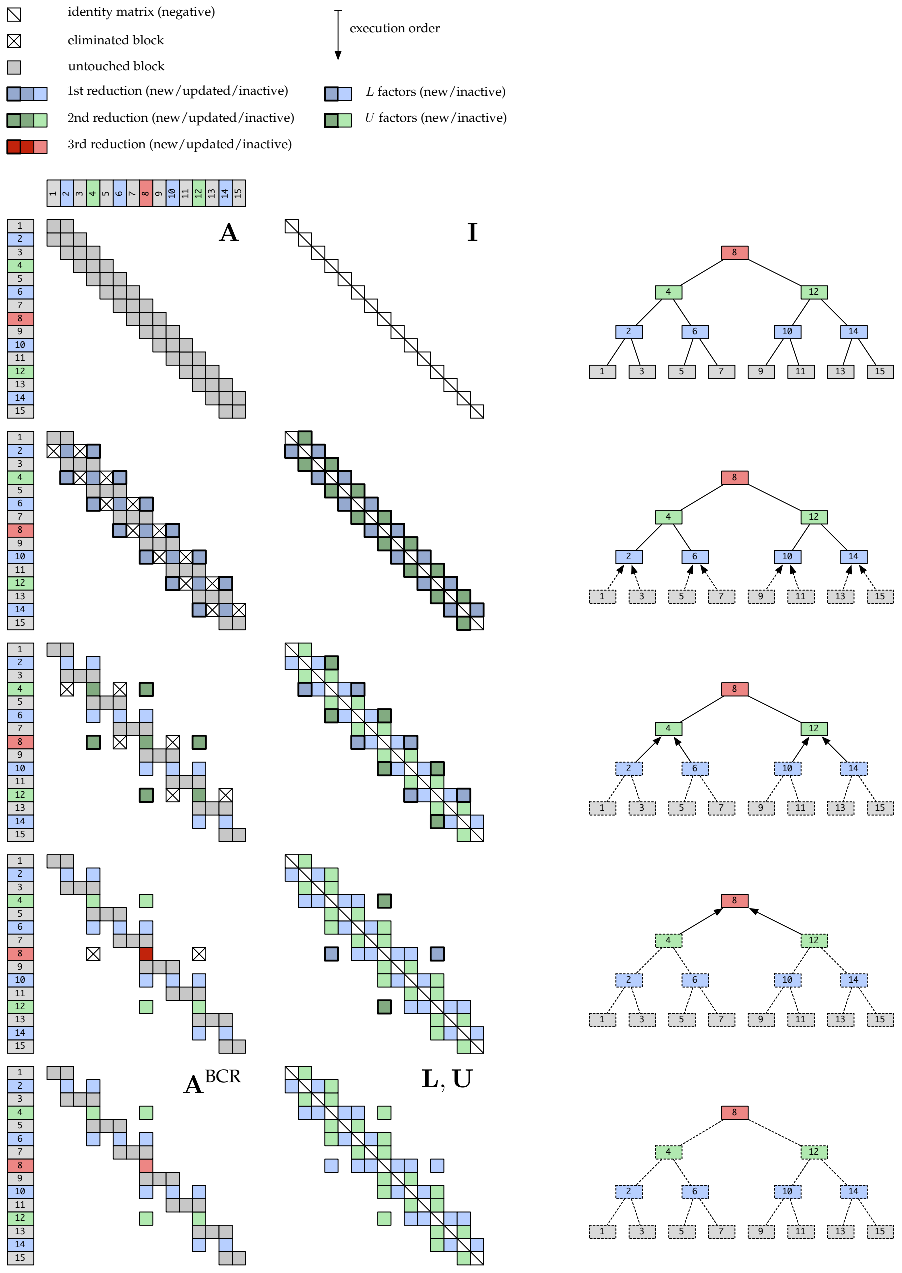

identity matrix (negative)
eliminated block
untouched block
1st reduction (new/updated/inactive)
2nd reduction (new/updated/inactive)
3rd reduction (new/updated/inactive)
execution order
L factors (new/inactive)
U factors (new/inactive)
A
I
A^BCR
L, U

Figure 4.7: This figure shows the full reduction phase in BCR for an example 15×15 block matrix **A**. The modified matrix **A** is shown in the left column, while the center column illustrates the preserved LU factors. The right column illustrates an elimination tree for this example that may aid understanding in how after each set of odd–element eliminations, we end up with a "reduced" system that is about half the size of the former system.

Continuing with Eq. (4.12) we might go through the following

$$\left(\begin{array}{cc|cc} \mathbf{a}_{11} & \mathbf{a}_{12} & \mathbf{i}_{11} & \\ & \mathbf{i}_{22} & \mathbf{g}_{22}\mathbf{l}_{21} & \mathbf{g}_{22} \end{array}\right) \tag{4.13}$$

$$\left(\begin{array}{cc|cc} \mathbf{a}_{11} & & \mathbf{i}_{11} - \mathbf{a}_{12}\mathbf{g}_{22}\mathbf{l}_{21} & -\mathbf{a}_{12}\mathbf{g}_{22} \\ & \mathbf{i}_{22} & \mathbf{g}_{22}\mathbf{l}_{21} & \mathbf{g}_{22} \end{array}\right) \tag{4.14}$$

$$\left(\begin{array}{cc|cc} \mathbf{i}_{11} & & \mathbf{a}_{11}^{-1} - \mathbf{a}_{11}^{-1}\mathbf{a}_{12}\mathbf{g}_{22}\mathbf{l}_{21} & -\mathbf{a}_{11}^{-1}\mathbf{a}_{12}\mathbf{g}_{22} \\ & \mathbf{i}_{22} & \mathbf{g}_{22}\mathbf{l}_{21} & \mathbf{g}_{22} \end{array}\right) \tag{4.15}$$

$$\left(\begin{array}{cc|cc} \mathbf{i}_{11} & & \mathbf{a}_{11}^{-1} + \mathbf{u}_{12}\mathbf{g}_{22}\mathbf{l}_{21} & \mathbf{u}_{12}\mathbf{g}_{22} \\ & \mathbf{i}_{22} & \mathbf{g}_{22}\mathbf{l}_{21} & \mathbf{g}_{22} \end{array}\right) \tag{4.16}$$

and thus assuming we have $\mathbf{g}_{22}$ from the outset, we now have produced the inverse blocks on the positions

$$\mathbf{g}_{12} = \mathbf{u}_{12}\mathbf{g}_{22}, \tag{4.17}$$

$$\mathbf{g}_{21} = \mathbf{g}_{22}\mathbf{l}_{21}, \tag{4.18}$$

$$\mathbf{g}_{11} = \mathbf{a}_{11}^{-1} + \mathbf{u}_{12}\mathbf{g}_{22}\mathbf{l}_{21}, \tag{4.19}$$

which renamed in generalized terms of producing the row k_{to} from knowing the inverse on $(k_{\text{from}}, k_{\text{from}})$, we have the new block diagonal element $\mathbf{g}_{k_{\text{to}},k_{\text{to}}}$ becoming

$$\mathbf{g}_{k_{\text{to}},k_{\text{to}}} = (\mathbf{a}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}})^{-1} + \mathbf{u}_{k_{\text{to}},k_{\text{from}}}\mathbf{g}_{k_{\text{from}},k_{\text{from}}}\mathbf{l}_{k_{\text{from}},k_{\text{to}}} \tag{4.20}$$

where we have used the LU factors stored during the reduction phase of BCR. Furthermore, we use $\mathbf{a}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}}$ in the above expression, as $\mathbf{a}_{k_{\text{to}},k_{\text{to}}}$ may be updated several times during the reduction phase. The off diagonal blocks can be determined as

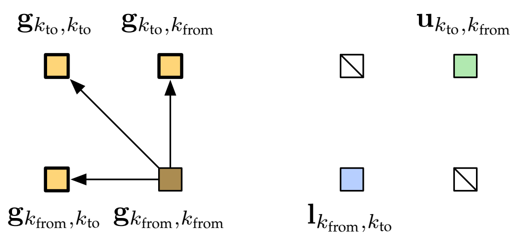
$$\mathbf{g}_{k_{\text{from}},k_{\text{to}}} = \mathbf{g}_{k_{\text{from}},k_{\text{from}}}\mathbf{l}_{k_{\text{from}},k_{\text{to}}}, \tag{4.21}$$

$$\mathbf{g}_{k_{\text{to}},k_{\text{from}}} = \mathbf{u}_{k_{\text{to}},k_{\text{from}}}\mathbf{g}_{k_{\text{from}},k_{\text{from}}}. \tag{4.22}$$

Reusing one of these results, we can save a multiplication and write Eq. (4.20) as

$$\mathbf{g}_{k_{\text{to}},k_{\text{to}}} = (\mathbf{a}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}})^{-1} + \mathbf{g}_{k_{\text{to}},k_{\text{from}}}\mathbf{l}_{k_{\text{from}},k_{\text{to}}}. \tag{4.23}$$

These formulas hold in either case of an upwards or downwards corner production, and presume only that we know $\mathbf{g}_{k_{\text{from}},k_{\text{from}}}$ as well as $\mathbf{l}_{k_{\text{from}},k_{\text{to}}}$ and $\mathbf{u}_{k_{\text{to}},k_{\text{from}}}$, which are determined during the reduction phase of BCR. A visualization of the corner production can be seen in Fig. 4.8.



$\mathbf{g}_{k_{\text{to}},k_{\text{to}}}$
$\mathbf{g}_{k_{\text{to}},k_{\text{from}}}$
$\mathbf{u}_{k_{\text{to}},k_{\text{from}}}$
$\mathbf{g}_{k_{\text{from}},k_{\text{to}}}$
$\mathbf{g}_{k_{\text{from}},k_{\text{from}}}$
$\mathbf{l}_{k_{\text{from}},k_{\text{to}}}$

Figure 4.8: The corner production step in BCR originates from a row k_{from} and produces towards a row k_{to} in the manner as seen above. Three new inverse blocks are produced, as seen on the left, using the stored LU factors from the reduction phase of BCR, as seen on the right.

4.3.1.3 Center Production

A center production takes care of the case when we need to produce a row k_{to} in between two already produced rows, k_{above} and k_{below} as we construct $\mathbf{G}$. A simple case that leads to a center production in BCR can be seen by looking at the following augmented system:

$$\left(\begin{array}{ccc|ccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & \mathbf{i}_{11} & & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & \mathbf{i}_{22} & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & & & \mathbf{i}_{33} \end{array}\right). \tag{4.24}$$

Two reduction operations, originating from the 2nd row upwards and downwards, lead to the following system

$$\left(\begin{array}{ccc|ccc} \mathbf{s}_{11} & & \mathbf{s}_{13} & \mathbf{i}_{11} & \mathbf{l}_{12} & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & \mathbf{i}_{22} & \\ \mathbf{s}_{31} & & \mathbf{s}_{33} & & \mathbf{l}_{32} & \mathbf{i}_{33} \end{array}\right), \tag{4.25}$$

where we have used the computed LU factors $\mathbf{l}_{12} = -\mathbf{a}_{12}\mathbf{a}_{22}^{-1}$ and $\mathbf{l}_{32} = -\mathbf{a}_{32}\mathbf{a}_{22}^{-1}$. We have also determined the corresponding factors $\mathbf{u}_{21} = -\mathbf{a}_{22}^{-1}\mathbf{a}_{21}$ and $\mathbf{u}_{23} = -\mathbf{a}_{22}^{-1}\mathbf{a}_{23}$ during the reduction phase which will be needed during the center production step.

The inverse on the corner blocks can then be determined by

$$\begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{13} \\ \mathbf{g}_{31} & \mathbf{g}_{33} \end{pmatrix} = \begin{pmatrix} \mathbf{s}_{11} & \mathbf{s}_{13} \\ \mathbf{s}_{31} & \mathbf{s}_{33} \end{pmatrix}^{-1}. \tag{4.26}$$

This can be understood easier if we take Eq. (4.24), swap row/column 1 and 2 and perceive it as a 2×2 block problem to obtain

$$\left(\begin{array}{c|cc||c|cc} \mathbf{a}_{22} & \mathbf{a}_{21} & \mathbf{a}_{23} & \mathbf{i}_{22} & & \\ \hline \mathbf{a}_{12} & \mathbf{a}_{11} & & & \mathbf{i}_{11} & \\ \mathbf{a}_{32} & & \mathbf{a}_{33} & & & \mathbf{i}_{33} \end{array}\right), \tag{4.27}$$

which we can compare directly to Eq. (4.11), in order to solve. This has been done for the corner production case and using Eqs. (4.17)–(4.19), we can write

$$\begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{13} \\ \mathbf{g}_{31} & \mathbf{g}_{33} \end{pmatrix} = \begin{pmatrix} \mathbf{s}_{11} & \mathbf{s}_{13} \\ \mathbf{s}_{31} & \mathbf{s}_{33} \end{pmatrix}^{-1} \tag{4.28}$$

$$\begin{pmatrix} \mathbf{g}_{12} \\ \mathbf{g}_{32} \end{pmatrix} = \begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{13} \\ \mathbf{g}_{31} & \mathbf{g}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{l}_{12} \\ \mathbf{l}_{32} \end{pmatrix} \tag{4.29}$$

$$\begin{pmatrix} \mathbf{g}_{21} & \mathbf{g}_{23} \end{pmatrix} = \begin{pmatrix} \mathbf{u}_{21} & \mathbf{u}_{23} \end{pmatrix} \begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{13} \\ \mathbf{g}_{31} & \mathbf{g}_{33} \end{pmatrix} \tag{4.30}$$

$$\mathbf{g}_{22} = \mathbf{a}_{22}^{-1} + \begin{pmatrix} \mathbf{u}_{21} & \mathbf{u}_{23} \end{pmatrix} \begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{13} \\ \mathbf{g}_{31} & \mathbf{g}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{l}_{12} \\ \mathbf{l}_{32} \end{pmatrix} \tag{4.31}$$

Thus we have, again generalizing now in terms of an upper row k_{above}, a middle row k_{to}, and a bottom row k_{from}, the following values for the inverse:

$$\mathbf{g}_{k_{\text{above}},k_{\text{to}}} = \mathbf{g}_{k_{\text{above}},k_{\text{above}}}\mathbf{l}_{k_{\text{above}},k_{\text{to}}} + \mathbf{g}_{k_{\text{above}},k_{\text{below}}}\mathbf{l}_{k_{\text{below}},k_{\text{to}}}, \tag{4.32}$$

$$\mathbf{g}_{k_{\text{to}},k_{\text{above}}} = \mathbf{u}_{k_{\text{to}},k_{\text{above}}}\mathbf{g}_{k_{\text{above}},k_{\text{above}}} + \mathbf{u}_{k_{\text{to}},k_{\text{below}}}\mathbf{g}_{k_{\text{below}},k_{\text{above}}}, \tag{4.33}$$

$$\mathbf{g}_{k_{\text{to}},k_{\text{to}}} = (\mathbf{a}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}})^{-1} + \mathbf{g}_{k_{\text{to}},k_{\text{above}}}\mathbf{l}_{k_{\text{above}},k_{\text{to}}} + \mathbf{g}_{k_{\text{to}},k_{\text{below}}}\mathbf{l}_{k_{\text{below}},k_{\text{to}}}, \tag{4.34}$$

$$\mathbf{g}_{k_{\text{to}},k_{\text{below}}} = \mathbf{u}_{k_{\text{to}},k_{\text{above}}}\mathbf{g}_{k_{\text{above}},k_{\text{below}}} + \mathbf{u}_{k_{\text{to}},k_{\text{below}}}\mathbf{g}_{k_{\text{below}},k_{\text{below}}}, \tag{4.35}$$

$$\mathbf{g}_{k_{\text{below}},k_{\text{to}}} = \mathbf{g}_{k_{\text{below}},k_{\text{above}}}\mathbf{l}_{k_{\text{above}},k_{\text{to}}} + \mathbf{g}_{k_{\text{below}},k_{\text{below}}}\mathbf{l}_{k_{\text{below}},k_{\text{to}}}, \tag{4.36}$$

where we assume we have already determined from the outset the corner blocks of the inverse $\mathbf{g}_{k_{\text{above}},k_{\text{above}}}$, $\mathbf{g}_{k_{\text{above}},k_{\text{below}}}$, $\mathbf{g}_{k_{\text{below}},k_{\text{above}}}$ and $\mathbf{g}_{k_{\text{below}},k_{\text{below}}}$ as well as the LU factors $\mathbf{l}_{k_{\text{above}},k_{\text{to}}}$, $\mathbf{l}_{k_{\text{below}},k_{\text{to}}}$ and $\mathbf{u}_{k_{\text{to}},k_{\text{above}}}$, $\mathbf{u}_{k_{\text{to}},k_{\text{below}}}$. Furthermore, $\mathbf{a}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}}$ is not necessarily equal to $\mathbf{a}_{k_{\text{to}},k_{\text{to}}}$ since this block may be updated several times under repeated reduction steps, as we mentioned earlier for the corner production step. The center production step can also be visualized in these generalized terms in Fig. 4.9, which can be loosely compared to Eq. (4.25).

The full production phase of BCR is visualized in Fig. 4.10, where we commence with the results of the reduction phase $\mathbf{A}^{\text{BCR}}$ on the left, and we visualize the production by the elimination tree on the left.

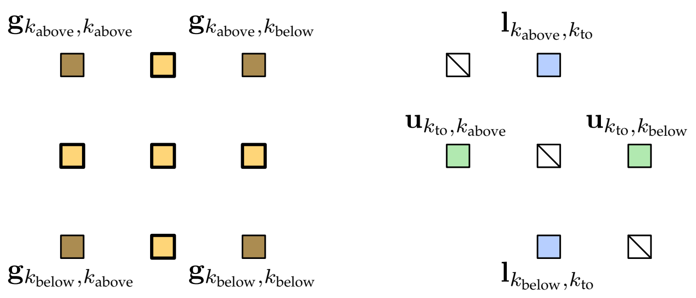

$\mathbf{g}_{k_{\text{above}},k_{\text{above}}}$
$\mathbf{g}_{k_{\text{above}},k_{\text{below}}}$
$\mathbf{l}_{k_{\text{above}},k_{\text{to}}}$
$\mathbf{u}_{k_{\text{to}},k_{\text{above}}}$
$\mathbf{u}_{k_{\text{to}},k_{\text{below}}}$
$\mathbf{g}_{k_{\text{below}},k_{\text{above}}}$
$\mathbf{g}_{k_{\text{below}},k_{\text{below}}}$
$\mathbf{l}_{k_{\text{below}},k_{\text{to}}}$

Figure 4.9: The center production step in BCR from the two rows k_{above} and k_{below} towards a center row k_{to} can be seen to produce the blocks of $\mathbf{G}$ as seen above on the left, while using the LU factors stored from the reduction phase as seen on the right. The lighter blocks on the left are the produced inverse blocks, while the darkened corner blocks are assumed to have been produced in an earlier stage of the BCR production phase.

4.3.2 Algorithm

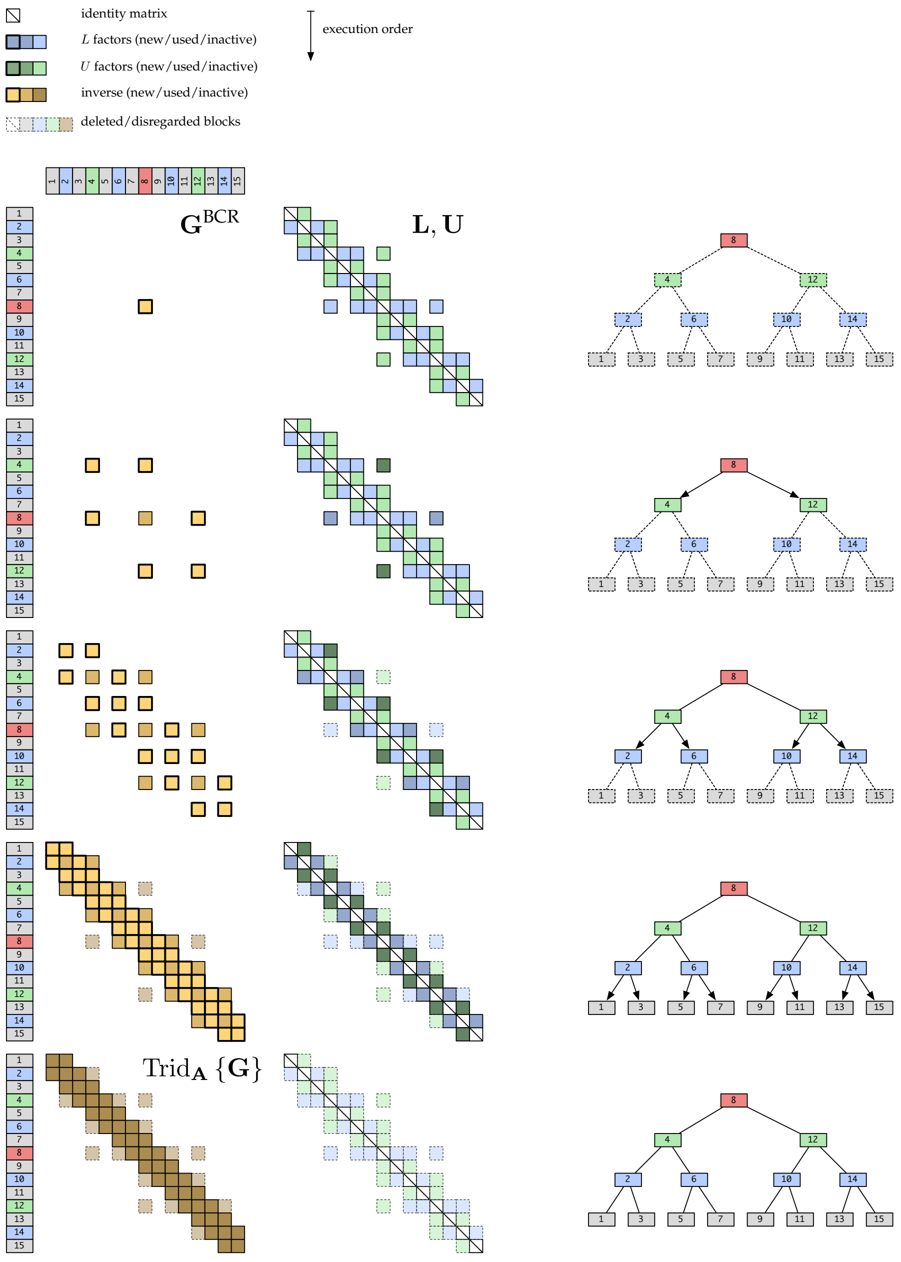
We now proceed to formalize the BCR method in pseudocode, such that it is relatively straight forward to implement, and that we may start to analyze the algorithms that make it up.

4.3.2.1 Control Code

A nontrivial amount of the BCR method code consists of *control* code, in the sense that it controls which rows are eliminate or produced, and what types of production are called.

We begin by looking at Alg. (4.5) REDUCEROWINDICES. This algorithm determines which rows we will be reducing towards, during a certain *level* of elimination in the reduction phase of BCR. The rows it delivers depend on an array of indices $\mathbf{i}^{\text{BCR}}$ that tells us what part of the original matrix $\mathbf{A}$ we want to perform BCR on. In the case of regular BCR as described in this section, our input array of indices $\mathbf{i}^{\text{BCR}} = \{1, 2, \ldots, n\}$, thus we consider all of $\mathbf{A}$ as the problem to be solved with BCR. The reason this might be different than expected will become clear when we introduce a Hybrid method in the next section. The algorithm will finally return the appropriate indices $\mathbf{i}^{\text{elim}}$ which need to be reduced towards in order to complete the reduction step at the *level* of elimination desired.

A counterpart to Alg. (4.5) REDUCEROWINDICES is an algorithm that will eventually tell us what rows we need to produce about, during the production



identity matrix
L factors (new/used/inactive)
U factors (new/used/inactive)
inverse (new/used/inactive)
deleted/disregarded blocks
execution order
$\mathbf{G}^{\text{BCR}}$
$\mathbf{L}, \mathbf{U}$
$\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$

Figure 4.10: This figure shows the full production phase in BCR for an example 15×15 block matrix $\mathbf{A}$. The initialized inverse block matrix $\mathbf{G}$ from the reduction phase is shown in the left column, where only $\mathbf{g}_{kk} = (\mathbf{a}^{\text{BCR}}_{kk})^{-1}$ has been calculated. Using the preserved LU factors, as seen in the center column, a series of production steps will leave us with $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ as seen at the bottom. The elimination tree from the reduction phase is shown in the right column, and guides us in the production phase, showing how we double the size of our computed portion of $\mathbf{G}$ as we move down the levels of the tree.

Algorithm 4.5 ReduceRowIndices($\mathbf{i}^{\text{BCR}}$, *level*)

```
1: $stride \leftarrow 2^{level-1}$
2: $i \leftarrow stride$
3: $\mathbf{i}^{\text{elim}} \leftarrow \{\}$
4: while $i \leq$ length of $\mathbf{i}^{\text{BCR}}$ do
5:     $\mathbf{i}^{\text{elim}} \curvearrowleft \mathbf{i}^{\text{BCR}}[i]$
6:     $i \leftarrow i + stride$
7: return $\mathbf{i}^{\text{elim}}$
```

phase of BCR when we begin reconstructing parts of our desired inverse, **G**. This is done by Alg. (4.6) ProduceRowIndices, which takes the indices $\mathbf{i}^{\text{BCR}}$ representing the matrix we are trying to perform BCR on (cf. Alg. (4.5) ReduceRowIndices) and the *level* of the elimination tree we are trying to produce from. The algorithm will then deliver a vector of indices $\mathbf{i}^{\text{prod}}$ which indicate the rows we will have to produce in order to accomplish the production step and move one level down the elimination tree.

Algorithm 4.6 ProduceRowIndices($\mathbf{i}^{\text{BCR}}$, *level*)

```
1: $stride \leftarrow 2^{level}$
2: $idx \leftarrow 2^{level-1}$
3: $\mathbf{i}^{\text{prod}} \leftarrow \{\}$
4: while $idx \leq n$ do
5:     $\mathbf{i}^{\text{prod}} \curvearrowleft idx$
6:     $idx \leftarrow idx + stride$
7: return $\mathbf{i}^{\text{prod}}$
```

An auxiliary function in the quest to the reduction step, where we eliminate about a certain row, is Alg. (4.7) ReductionIndices. The algorithm takes the argument $\mathbf{i}^{\text{elim}}$, which is a vector of indices involved in the reduction steps at a certain level in the reduction phase. The argument *row* tells us which row in the block tridiagonal matrix represented by the indices $\mathbf{i}^{\text{elim}}$ we are reducing towards. The algorithm then assigns values to a quintet of indices, namely h, i, j, k and l, which will be the indices telling us which blocks in $\mathbf{A}^{\text{BCR}}$ will be used in the reduction step (cf. Eq. (4.4)).

4.3.2.2 Reduction

The single reduction step is taken care of by Alg. (4.8) Reduce, which is relatively long and complex as it takes care of communication between processes

Algorithm 4.7 REDUCTIONINDICES(row, $\mathbf{i}^{\text{elim}}$)

```
1:  $h, i \leftarrow -\infty$
2:  $j \leftarrow \mathbf{i}^{\text{elim}}[row]$
3:  $k, l \leftarrow \infty$
4:  if $row - 2 \geq 1$ then
5:      $h \leftarrow \mathbf{i}^{\text{elim}}[row - 2]$
6:  if $row - 1 \geq 1$ then
7:      $i \leftarrow \mathbf{i}^{\text{elim}}[row - 1]$
8:  if $row + 1 \leq n$ then
9:      $k \leftarrow \mathbf{i}^{\text{elim}}[row + 1]$
10: if $row + 2 \leq n$ then
11:     $l \leftarrow \mathbf{i}^{\text{elim}}[row + 2]$
12: return $h, i, j, k, l$
```

as well as the mathematical operations that carry out the reductions given in Eqs. (4.5)–(4.7).

The very first call by REDUCE is to Alg. (4.7) REDUCTIONINDICES in order to determine the indices h, i, j, k and l involved in the single reduction step as shown in Eq. (4.4). REDUCE then has two cases, which are both mirror images of each other, as the first case on line 2 executes if the row we are reducing towards has a row above it that will be eliminated[1], and the case on line 18 will execute if there is a row to be eliminated below.

Since the matrix $\mathbf{A}$ (and all other block matrices) may be distributed among several machines with only direct access to their owned portions of the matrices, we resort to message passing communication in order to share data among them. Operating with the case on line 2, we have row i to be eliminated, and row j to be updated accordingly. Looking at line 5 to line 8 we take care to send the values in $\mathbf{A}$ to the owner of row j, such that it can proceed with work. The owner of row i, once transmission is completed, can then calculate the corresponding factor $\mathbf{u}_{ij}$ used later in the production phase. The owner of row j receives data via line 9 to line 12, at which point it will be ready to determine the LU factor $\mathbf{l}_{ji}$ and to update $\mathbf{a}_{jj}^{\text{BCR}}$ and $\mathbf{a}_{jh}^{\text{BCR}}$, if it exists. This communication, of course, does not take place if rows i and j are owned by the same process. The portion of REDUCE for the communication and work between row j and k is similar to this, and is not described.

[1]In the manner BCR is specified here, this will always be the case. This is because we choose to eliminate odd–indexed rows starting from the top. One could imagine BCR eliminating odd–indexed rows whose numbering commences from the bottom of $\mathbf{A}$.

Algorithm 4.8 REDUCE($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, row, level, \mathbf{i}^{\text{elim}}$)

```
 1: $h, i, j, k, l \leftarrow$ REDUCTIONINDICES($row, \mathbf{i}^{\text{elim}}$)          get working indices
 2: if $i \geq \mathbf{i}^{\text{elim}}[1]$ then                                  if there is a row above
 3:    if row $i$ is mine then
 4:       $\mathbf{u}_{ij} \leftarrow -(\mathbf{a}_{ii}^{\text{BCR}})^{-1}\mathbf{a}_{ij}^{\text{BCR}}$                    1op($\mathcal{LU}$), 1op($\times$)
 5:    if row $i$ is mine and row $j$ is not mine then
 6:       send $(\mathbf{a}_{ii}^{\text{BCR}})^{-1}, \mathbf{a}_{ij}^{\text{BCR}}$ to owner of row $j$
 7:       if $\mathbf{a}_{ih}^{\text{BCR}}$ exists then
 8:          send $\mathbf{a}_{ih}^{\text{BCR}}$ to owner of row $j$
 9:    if row $i$ is not mine and row $j$ is mine then
10:       recv $(\mathbf{a}_{ii}^{\text{BCR}})^{-1}, \mathbf{a}_{ij}^{\text{BCR}}$ from owner of row $i$
11:       if $\mathbf{a}_{ih}^{\text{BCR}}$ exists then
12:          recv $\mathbf{a}_{ih}^{\text{BCR}}$ from owner of row $i$
13:    if row $j$ is mine then
14:       $\mathbf{l}_{ji} \leftarrow -\mathbf{a}_{ji}^{\text{BCR}}(\mathbf{a}_{ii}^{\text{BCR}})^{-1}$                    1op($\times$)
15:       $\mathbf{a}_{jj}^{\text{BCR}} \leftarrow \mathbf{a}_{jj}^{\text{BCR}} + \mathbf{l}_{ji}\mathbf{a}_{ij}^{\text{BCR}}$                 1op($\times$), 1op($+$)
16:       if $\mathbf{a}_{ih}^{\text{BCR}}$ exists then
17:          $\mathbf{a}_{jh}^{\text{BCR}} \leftarrow \mathbf{l}_{ji}\mathbf{a}_{ih}^{\text{BCR}}$                    1op($\times$)
18: if $k \leq \mathbf{i}^{\text{elim}}[\text{end}]$ then                              if there is a row below
19:    if row $k$ is mine then
20:       $\mathbf{u}_{kj} \leftarrow -(\mathbf{a}_{kk}^{\text{BCR}})^{-1}\mathbf{a}_{kj}^{\text{BCR}}$                    1op($\mathcal{LU}$), 1op($\times$)
21:    if row $k$ is mine and row $j$ is not mine then
22:       send $(\mathbf{a}_{kk}^{\text{BCR}})^{-1}, \mathbf{a}_{kj}^{\text{BCR}}$ to owner of row $j$
23:       if $\mathbf{a}_{kl}^{\text{BCR}}$ exists then
24:          send $\mathbf{a}_{kl}^{\text{BCR}}$ to owner of row $j$
25:    if row $k$ is not mine and row $j$ is mine then
26:       recv $(\mathbf{a}_{kk}^{\text{BCR}})^{-1}, \mathbf{a}_{kj}^{\text{BCR}}$ from owner of row $k$
27:       if $\mathbf{a}_{kl}^{\text{BCR}}$ exists then
28:          recv $\mathbf{a}_{kl}^{\text{BCR}}$ from owner of row $k$
29:    if row $j$ is mine then
30:       $\mathbf{l}_{jk} \leftarrow -\mathbf{a}_{jk}^{\text{BCR}}(\mathbf{a}_{kk}^{\text{BCR}})^{-1}$                    1op($\times$)
31:       $\mathbf{a}_{jj}^{\text{BCR}} \leftarrow \mathbf{a}_{jj}^{\text{BCR}} + \mathbf{l}_{jk}\mathbf{a}_{kj}^{\text{BCR}}$                 1op($\times$), 1op($+$)
32:       if $\mathbf{a}_{kl}^{\text{BCR}}$ exists then
33:          $\mathbf{a}_{jl}^{\text{BCR}} \leftarrow \mathbf{l}_{jk}\mathbf{a}_{kl}^{\text{BCR}}$                    1op($\times$)
34: return $\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}$
```

The full reduction phase of BCR is performed by Alg. (4.9) REDUCEBCR. The algorithm takes as arguments the block matrix $\mathbf{A}$ and the array of indices $\mathbf{i}^{\mathrm{BCR}}$ which defines what part of $\mathbf{A}$ we wish to perform BCR on. In the case of performing simple BCR on all of $\mathbf{A}$, as done in this section, we have $\mathbf{i}^{\mathrm{BCR}} = \{1, 2, \ldots, n\}$. In this case, $\mathbf{L}$ and $\mathbf{U}$ will also simply be identity matrices $\mathbf{I}$, and it is only in the next section, when we introduce the Hybrid method, that these arguments will differ from this.

Algorithm 4.9 REDUCEBCR$(\mathbf{A}, \mathbf{L}, \mathbf{U}, \mathbf{i}^{\mathrm{BCR}})$

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\mathbf{A}^{\mathrm{BCR}}, \mathbf{L}, \mathbf{U} \in \mathbb{B}^{n,n}$
1: $k \leftarrow$ length of $\mathbf{i}^{\mathrm{BCR}}$
2: $h \leftarrow \log_2(k)$
3: for $level = 1$ up to $h$ do
4:     $\mathbf{i}^{\mathrm{elim}} \leftarrow$ REDUCEROWINDICES$(\mathbf{i}^{\mathrm{BCR}}, level)$        determine active rows
5:     for $row = 1$ up to length of $\mathbf{i}^{\mathrm{elim}}$ do                  eliminate active rows
6:         $\mathbf{A}^{\mathrm{BCR}}, \mathbf{L}, \mathbf{U} \leftarrow$ REDUCE$(\mathbf{A}, \mathbf{L}, \mathbf{U}, row, level, \mathbf{i}^{\mathrm{elim}})$
7: return $\mathbf{A}^{\mathrm{BCR}}, \mathbf{L}, \mathbf{U}$
```

The first two lines of REDUCEBCR determines the height of the elimination tree characterized by our reduction/production phases. In the case of plain BCR, k on line 1 evaluates to n. The elimination tree, being a balanced binary tree, will have a height characterized by the base 2 logarithm taken on the number of nodes it has. This is performed on line 2, where h is taken to be the integer part of $\log_2(k)$. Thus

$$h = \lfloor \log_2(k) \rfloor \tag{4.37}$$

where k denotes the size (number of diagonal blocks) of the tridiagonal system undergoing BCR.

The for loop on line 3 then takes care of looping over every level of the elimination tree, while the loop on line 5 will loop over all indices involved at the respective level of elimination, and call Alg. (4.8) REDUCE for every index. These involved indices are provided by a preliminary call to Alg. (4.5) REDUCEROWINDICES at the start of the outer for loop on line 4.

Once all levels of elimination have been processed, the algorithm returns the modified block matrix $\mathbf{A}^{\mathrm{BCR}}$, which now may have nonzero blocks away from the block tridiagonal (cf. Fig. 4.7 and Fig. 4.11).

The full reduction phase and communication pattern for an example 15×15 block matrix $\mathbf{A}$ is shown in Fig. 4.11, where we have distributed data across 5 processes. It can be seen that initially, the reduction phase is highly parallel, but as we proceed up the elimination tree, more and more processes will remain idle. In this case, we also only have nearest–neighbor communications, but if **A** were larger and we might have more processes, communication will stretch further.

Another figure that might aid understanding is presented in Fig. 4.12. Here we show the reduction phase of BCR on the same example 15×15 block **A**, where we have re–ordered the indices in **A** such that we can see how BCR corresponds to a downwards Gaussian elimination[1] phase on all subdiagonals, and subsequent fill–in, of the re–ordered **A**. This also leads to the strictly triangular **L** and **U** matrices as might be expected of an LU factorization.

4.3.2.3 Production

The single reduction step is taken care of by Alg. (4.8) REDUCE, which is relatively long and complex as it takes care of communication between processes as well as the mathematical operations that carry out the reductions given in Eqs. (4.5)–(4.7).

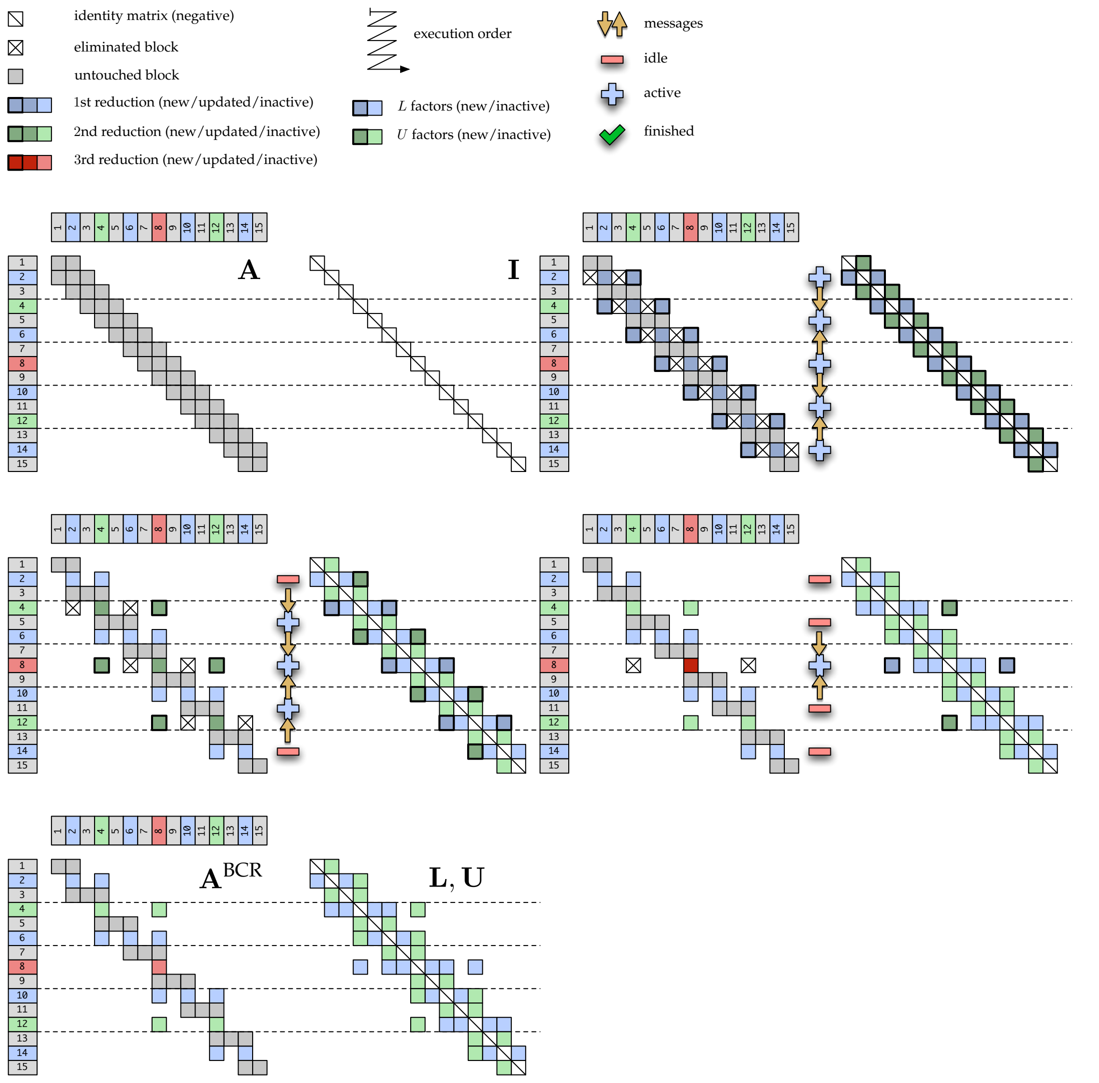
The production phase of BCR is characterized by one of two basic operations, namely those of corner and center productions. The corner production step is detailed in Alg. (4.10) CORNERPRODUCE, and determines blocks of the inverse as given by Eqs. (4.21)–(4.23) and as visualized in Fig. 4.8.

As the data may be distributed, the involved block rows k_{from} and k_{to} may reside on different processes. Thus the first part of the algorithm via line 1 to line 3 takes care of sending data from k_{from} to k_{to}, such that the process owning k_{to} can determine the inverse blocks belonging to it. The row owning k_{from} can then finish its job by determining $\mathbf{g}_{k_{\text{from}},k_{\text{to}}}$ done on line 4.

The process owning row k_{to} will wait to receive data from the process owning row k_{from} via code between line 5 and line 7, before producing the inverse blocks belonging to it on line 8 and line 9. If the same process owns rows k_{to} and k_{from}, no message passing takes place, and the inverse blocks are calculated quickly.

The center production step, given in in Alg. (4.11) CENTERPRODUCE handles determining the inverse as given by Eqs. (4.32)–(4.36) and visualized by Fig. 4.9. Again, as for REDUCE, and as seen on Fig. 4.9, 3 distinct rows are involved named k_{above}, k_{below} and k_{to}, for the rows above, below and the row to be produced, respectively.

[1]As first identified by Golub [32].



identity matrix (negative)
eliminated block
untouched block
1st reduction (new/updated/inactive)
2nd reduction (new/updated/inactive)
3rd reduction (new/updated/inactive)
execution order
L factors (new/inactive)
U factors (new/inactive)
messages
idle
active
finished
A
I
$\mathbf{A}^{\mathrm{BCR}}$
L, U

Figure 4.11: Visualizing the reduction phase of the block cyclic reduction method for calculating the block tridiagonal inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix **A** with 15 diagonal blocks. The matrices are distributed among the processes according to the horizontal dashed lines, and inter–process communications are indicated by arrows. Idle processes are indicated by "−", while working processes (not counting communication) are indicated by "+". The **L** and **U** factors are displayed on the right hand side of the augmented matrices, with blue and green blocks, respectively.

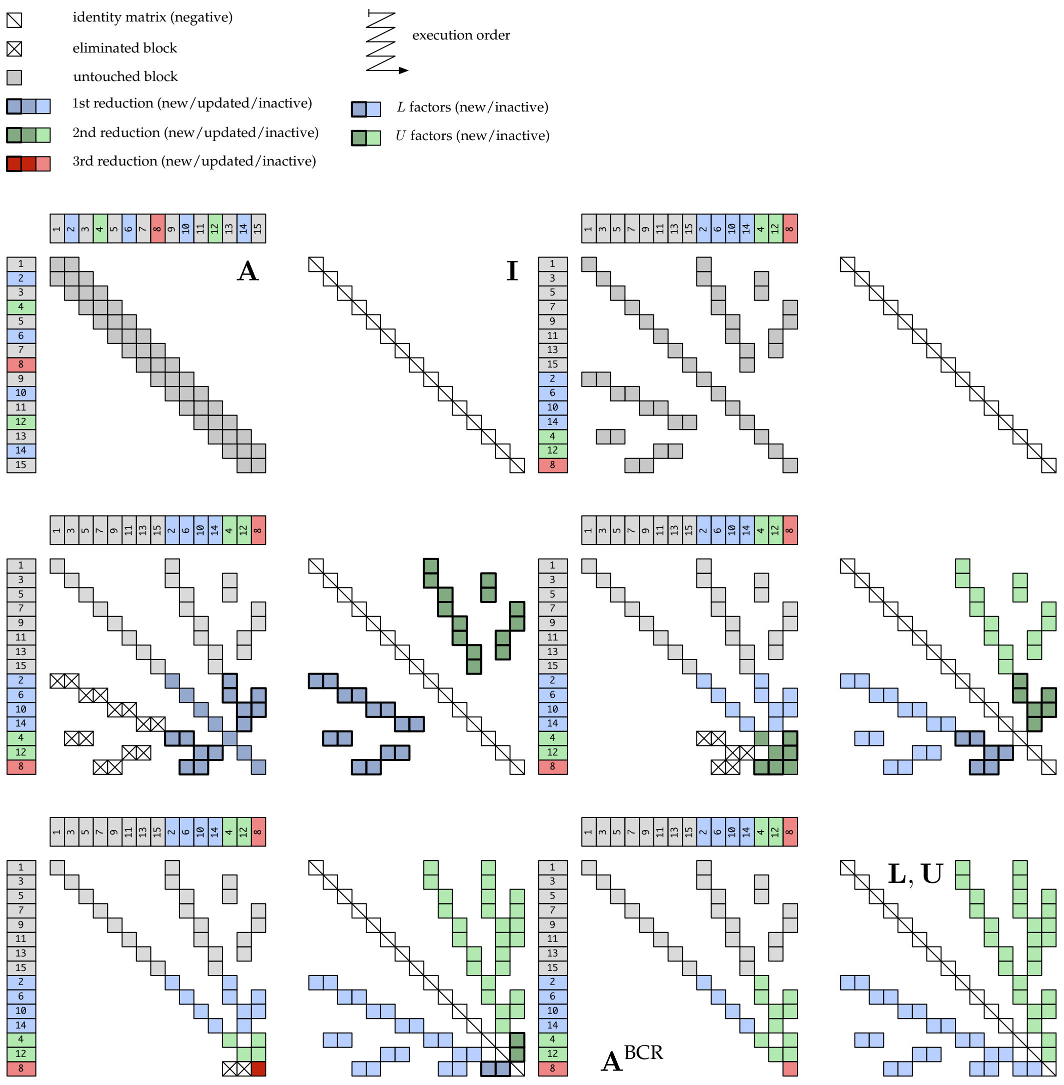

identity matrix (negative)
eliminated block
untouched block
1st reduction (new/updated/inactive)
2nd reduction (new/updated/inactive)
3rd reduction (new/updated/inactive)
execution order
L factors (new/inactive)
U factors (new/inactive)
A
I
A^BCR
L, U

Figure 4.12: Visualizing the reduction phase of the block cyclic reduction method for calculating the block tridiagonal inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$ with 15 diagonal blocks in reordered fashion. The $\mathbf{L}$ and $\mathbf{U}$ factors are displayed on the right hand side of the augmented matrices, with blue and green blocks, respectively.

Algorithm 4.10 CornerProduce($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, k_{\text{from}}, k_{\text{to}}$)

```
1: if row $k_{\text{from}}$ is mine then
2:     if row $k_{\text{to}}$ is not mine then
3:         send $\mathbf{g}_{k_{\text{from}},k_{\text{from}}}, \mathbf{l}_{k_{\text{from}},k_{\text{to}}}$ to owner of row $k_{\text{to}}$
4:     $\mathbf{g}_{k_{\text{from}},k_{\text{to}}} \leftarrow \mathbf{g}_{k_{\text{from}},k_{\text{from}}} \mathbf{l}_{k_{\text{from}},k_{\text{to}}}$        1op($\times$)
5: if row $k_{\text{to}}$ is mine then
6:     if row $k_{\text{from}}$ is not mine then
7:         recv $\mathbf{g}_{k_{\text{from}},k_{\text{from}}}, \mathbf{l}_{k_{\text{from}},k_{\text{to}}}$ from owner of row $k_{\text{from}}$
8:     $\mathbf{g}_{k_{\text{to}},k_{\text{from}}} \leftarrow \mathbf{u}_{k_{\text{to}},k_{\text{from}}} \mathbf{g}_{k_{\text{from}},k_{\text{from}}}$        1op($\times$)
9:     $\mathbf{g}_{k_{\text{to}},k_{\text{to}}} \leftarrow (\mathbf{a}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}})^{-1} + \mathbf{g}_{k_{\text{to}},k_{\text{from}}} \mathbf{l}_{k_{\text{from}},k_{\text{to}}}$        1op($\times$), 1op($+$)
10: return $\mathbf{G}^{\text{BCR}}$
```

Communication for the process owning the upper row k_{above} is taken care of by line 1 to line 6, as it passes messages to the processes owning k_{to} and k_{below}. In a likewise manner, communication for the process owning the lower row k_{below} is done by line 8 to line 13. When their communications have completed, they finish by performing the work necessary to complete parts of the inverse that belong to them, namely Eq. (4.32) and Eq. (4.36) via line 7 and line 14, respectively.

Finally, communication for the process owning the middle row k_{to} is handled by line 15 to line 19. When it has received this data, line 20 to line 22 handles the work of determining parts of $\mathbf{G}$ as dictacted by Eqs. (4.33)–(4.35). Once a process has calculated their respective owned blocks of the inverse, the method returns.

Something to note is the switched order of the send/receive operations on lines 3,4 and 10,11 that involve direct communications between the processes owning rows k_{above} and k_{below}. This is due to the blocking nature of the send/receive calls used in the algorithm, and in order to avoid deadlock, we ensure that we pass information from k_{above} to k_{below}, before passing information from k_{below} to k_{above}.

The full production phase of BCR is performed by Alg. (4.12) ProduceBCR. The algorithm takes as arguments the block matrix $\mathbf{A}^{\text{BCR}}$, as delivered by the reduction phase completed by ReduceBCR, as well as the associated LU factors $\mathbf{L}$ and $\mathbf{U}$. Furthermore, we specify a near–empty $\mathbf{G}$, which has been initialized to only contain $\mathbf{g}_{kk}$, determined simply after the full reduction phase via Eq. (4.10).

Just as for ReduceBCR, we supply an array of indices $\mathbf{i}^{\text{BCR}}$ which define what part of $\mathbf{A}$ we wish to perform BCR on. Again, in the case of performing

Algorithm 4.11 CENTERPRODUCE($\mathbf{A}^{\mathrm{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}, k_{\mathrm{above}}, k_{\mathrm{to}}, k_{\mathrm{below}}$)

```
 1: if row $k_{\mathrm{above}}$ is mine then
 2:     if row $k_{\mathrm{below}}$ is not mine then
 3:         send $\mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}}$ to owner of row $k_{\mathrm{below}}$
 4:         recv $\mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$ from owner of row $k_{\mathrm{below}}$
 5:     if row $k_{\mathrm{to}}$ is not mine then
 6:         send $\mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{above}}}, \mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{below}}}, \mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}}$ to owner of row $k_{\mathrm{to}}$
 7:     $\mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{to}}} \leftarrow \mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{above}}}\mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}} + \mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{below}}}\mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$        2op(×), 1op(+)
 8: if row $k_{\mathrm{below}}$ is mine then
 9:     if row $k_{\mathrm{above}}$ is not mine then
10:         recv $\mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}}$ from owner of row $k_{\mathrm{above}}$
11:         send $\mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$ to owner of row $k_{\mathrm{above}}$
12:     if row $k_{\mathrm{to}}$ is not mine then
13:         send $\mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{below}}}, \mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{above}}}, \mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$ to owner of row $k_{\mathrm{to}}$
14:     $\mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{to}}} \leftarrow \mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{above}}}\mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}} + \mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{below}}}\mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$        2op(×), 1op(+)
15: if row $k_{\mathrm{to}}$ is mine then
16:     if row $k_{\mathrm{above}}$ is not mine then
17:         recv $\mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{above}}}, \mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{below}}}, \mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}}$ from owner of row $k_{\mathrm{above}}$
18:     if row $k_{\mathrm{below}}$ is not mine then
19:         recv $\mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{below}}}, \mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{above}}}, \mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$ from owner of row $k_{\mathrm{above}}$
20:     $\mathbf{g}_{k_{\mathrm{to}},k_{\mathrm{above}}} \leftarrow \mathbf{u}_{k_{\mathrm{to}},k_{\mathrm{above}}}\mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{above}}} + \mathbf{u}_{k_{\mathrm{to}},k_{\mathrm{below}}}\mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{above}}}$        2op(×), 1op(+)
21:     $\mathbf{g}_{k_{\mathrm{to}},k_{\mathrm{below}}} \leftarrow \mathbf{u}_{k_{\mathrm{to}},k_{\mathrm{above}}}\mathbf{g}_{k_{\mathrm{above}},k_{\mathrm{below}}} + \mathbf{u}_{k_{\mathrm{to}},k_{\mathrm{below}}}\mathbf{g}_{k_{\mathrm{below}},k_{\mathrm{below}}}$        2op(×), 1op(+)
22:     $\mathbf{g}_{k_{\mathrm{to}},k_{\mathrm{to}}} \leftarrow (\mathbf{a}^{\mathrm{BCR}}_{k_{\mathrm{to}},k_{\mathrm{to}}})^{-1} + \mathbf{g}_{k_{\mathrm{to}},k_{\mathrm{above}}}\mathbf{l}_{k_{\mathrm{above}},k_{\mathrm{to}}} + \mathbf{g}_{k_{\mathrm{to}},k_{\mathrm{below}}}\mathbf{l}_{k_{\mathrm{below}},k_{\mathrm{to}}}$        2op(×), 2op(+)
23: return $\mathbf{G}^{\mathrm{BCR}}$
```

simple BCR on all of $\mathbf{A}$, we have $\mathbf{i}^{\text{BCR}} = \{1, 2, \ldots, n\}$, while in the next section, when we introduce the Hybrid method, these arguments will differ.

Algorithm 4.12 ProduceBCR($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}, \mathbf{i}^{\text{BCR}}$)

```
Require: $\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U} \in \mathbb{B}^{n,n}$, $[\mathbf{G}]_{kk} = [\mathbf{A}^{-1}]_{kk}$
Ensure: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} = \text{Trid}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
 1: $stride \leftarrow 2^{level-1}$                                              stride
 2: for $level = h$ down to $1$ do
 3:     $\mathbf{i}^{\text{prod}} \leftarrow$ ProduceRowIndices($\mathbf{i}^{\text{BCR}}, level$)   determine rows to be produced
 4:     for $i = 1$ up to length of $\mathbf{i}^{\text{prod}}$ do
 5:         $k_{\text{to}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i]]$
 6:         if $i = 1$ then                                                    case 1
 7:             $k_{\text{from}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i] + stride]$
 8:             $\mathbf{G}^{\text{BCR}} \leftarrow$ CornerProduce($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, k_{\text{from}}, k_{\text{to}}$)
 9:         if $i \neq 1$ and $i =$ length of $\mathbf{i}^{\text{prod}}$ then                      case 2
10:             if $\mathbf{i}^{\text{prod}}[\text{end}] \leq$ length of $\mathbf{i}^{\text{BCR}} - stride$ then         case 2A
11:                 $k_{\text{above}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i] - stride]$
12:                 $k_{\text{below}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i] + stride]$
13:                 $\mathbf{G}^{\text{BCR}} \leftarrow$ CenterProduce($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, k_{\text{above}}, k_{\text{to}}, k_{\text{below}}$)
14:             else                                                           case 2B
15:                 $k_{\text{from}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i] - stride]$
16:                 $\mathbf{G}^{\text{BCR}} \leftarrow$ CornerProduce($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, k_{\text{from}}, k_{\text{to}}$)
17:         if $i \neq 1$ and $i \neq$ length of $\mathbf{i}^{\text{prod}}$ then                      case 3
18:             $k_{\text{above}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i] - stride]$
19:             $k_{\text{below}} \leftarrow \mathbf{i}^{\text{BCR}}[\mathbf{i}^{\text{prod}}[i] + stride]$
20:             $\mathbf{G}^{\text{BCR}} \leftarrow$ CenterProduce($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, k_{\text{above}}, k_{\text{to}}, k_{\text{below}}$)
21: return $\mathbf{G}^{\text{BCR}}$
```

4.3.2.4 Inversion

Finally, we can present the algorithm that will calculate, via BCR, the block tridiagonal part of the inverse $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ for an arbitrary block tridiagonal $\mathbf{A}$ in Alg. (4.13) InverseBCR. Taking in only $\mathbf{A}$ as an argument, we first dictate that all rows in $\mathbf{A}$ will be active indices in the BCR method, by specifying the index array $\mathbf{i}^{\text{BCR}} = \{1, 2, \ldots, n\}$ on line 1.

The algorithm then launches into the first distinct phase of BCR, as seen in Fig. 4.6, on line 2. After the reduction phase has completed, we prime the inverse $\mathbf{G}$ with the first block on line 3 as determined via Eq. (4.10). We can then proceed with the production phase on line 4 by calling PRODUCEBCR with the primed **G** and the results from REDUCEBCR. When this has completed, we can extract $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ from the result $\mathbf{G}^{\text{BCR}}$, of which the block tridiagonal is a subset, and we can return and terminate execution.

Algorithm 4.13 INVERSEBCR(**A**)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} = \text{Trid}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: $\mathbf{i}^{\text{BCR}} \leftarrow \{1, 2, \ldots, n\}$
2: $\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U} \leftarrow$ REDUCEBCR($\mathbf{A}, \mathbf{i}^{\text{BCR}}$)
3: $\mathbf{g}_{kk}^{\text{BCR}} = (\mathbf{a}_{kk}^{\text{BCR}})^{-1}$                    $1\text{op}(\mathcal{LU}), 1\text{op}(\times)$
4: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} \subset \mathbf{G}^{\text{BCR}} \leftarrow$ PRODUCEBCR($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, \mathbf{i}^{\text{BCR}}$)
5: return $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
```

The entire production phase and communication pattern for an example 15×15 block matrix **A** is shown in Fig. 4.13, where we have distributed data across 5 processes. It can be seen that initially, the production phase is highly sequential, in that only the process owning $\mathbf{g}_{kk}$ and its neighbors with respect to the elimination tree have work to do. However, as we proceed down the elimination tree, more and more processes can take up work, and the method gains parallel efficiency.

A figure presenting a re–ordered version of Fig. 4.13 is given in Fig. 4.12. Here we show the production phase of BCR on the same example 15×15 block **A**, where we have re–ordered the indices in **A** just as before such that BCR can be seen to be a downwards Gaussian elimination phase.

4.3.3 Complexity

In order to perform a complexity analysis on BCR, we can take advantage of the properties of the elimination trees as seen earlier in Fig. 4.7 and Fig. 4.10 in order to perform a worst–case operation count. Taking the same 15×15 block example, we look at the elimination tree again in Fig. 4.15, where we have labeled the individual row operations with arrows as the BCR method goes through its reduction and production phases.

Blue arrows pointing upwards indicate a row operation in conjunction with a center reduction step, while a red arrow heading upwards indicates a row operation in relation to a corner reduction step. This counts for downwards arrows, however here the corner/center steps are of the production type that generate blocks of the inverse **G**.

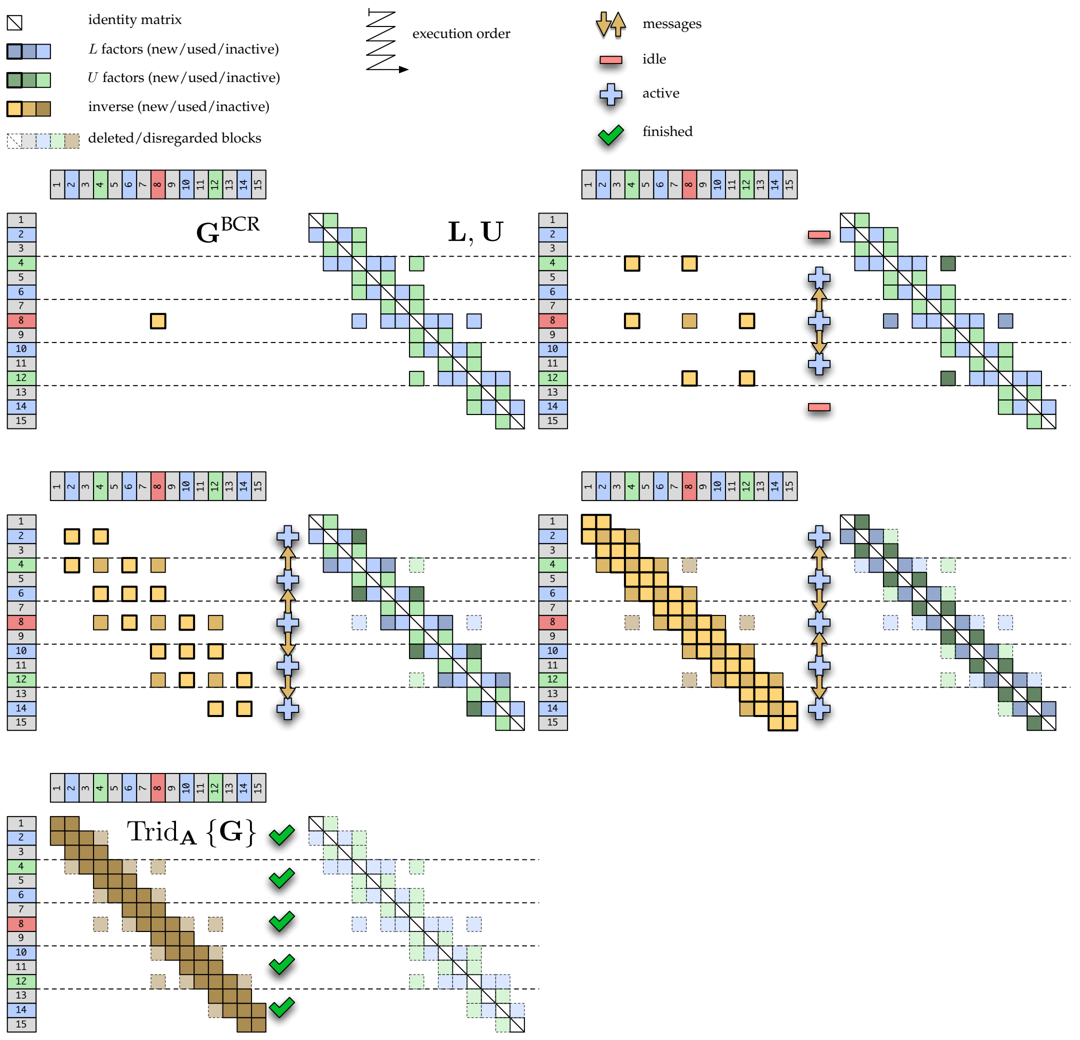

identity matrix
L factors (new/used/inactive)
U factors (new/used/inactive)
inverse (new/used/inactive)
deleted/disregarded blocks
execution order
messages
idle
active
finished
$\mathbf{G}^{\mathrm{BCR}}$
$\mathbf{L}, \mathbf{U}$
$\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$

Figure 4.13: Visualizing the production phase of the block cyclic reduction method for calculating the block tridiagonal inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$ with 15 diagonal blocks. The matrices are distributed among the processes according to the horizontal dashed lines, and inter–process communications are indicated by arrows. Idle processes are indicated by "$-$", while working processes (not counting communication) are indicated by "$+$". The $\mathbf{L}$ and $\mathbf{U}$ factors are displayed on the right hand side of the augmented matrices, with blue and green blocks, respectively.

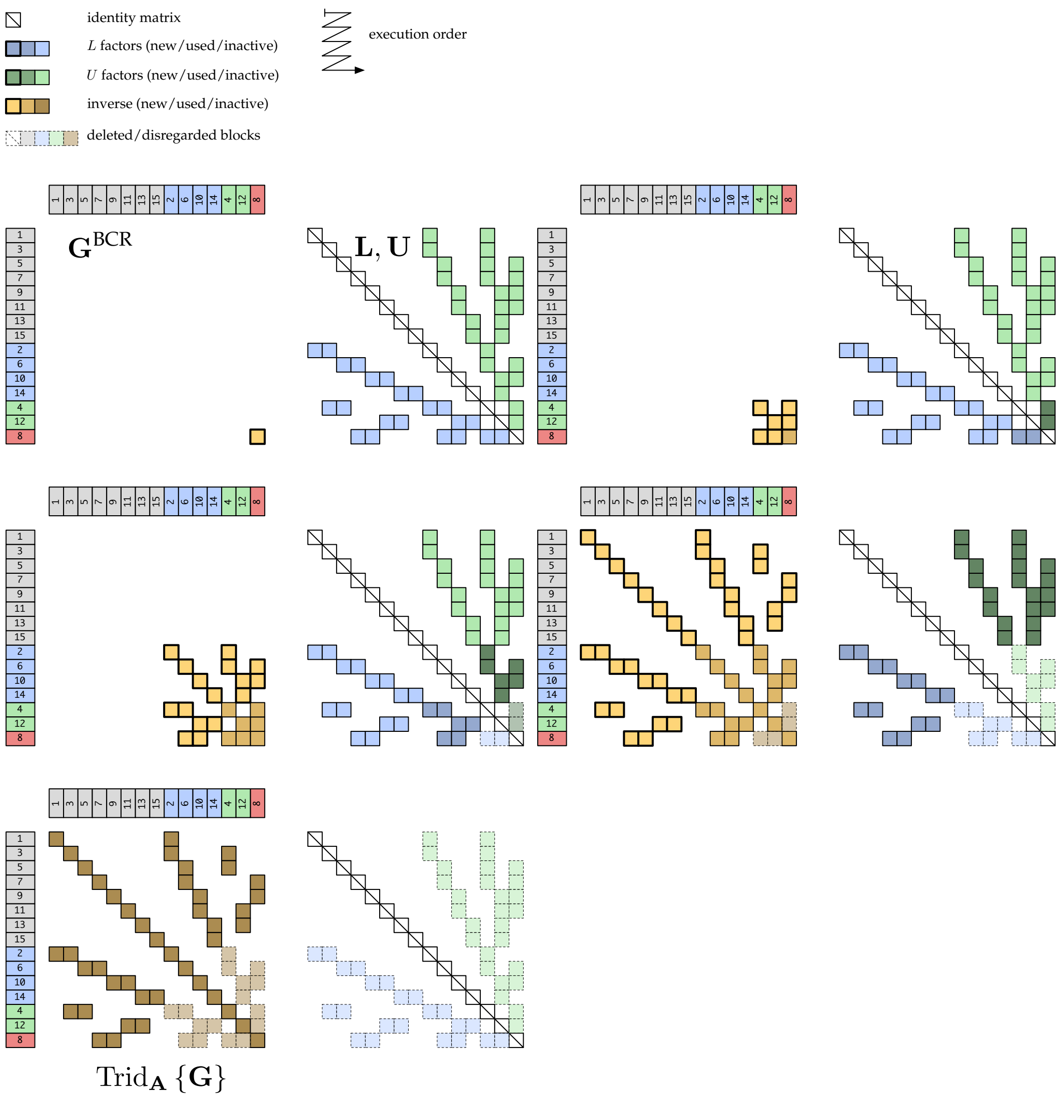

identity matrix
L factors (new/used/inactive)
U factors (new/used/inactive)
inverse (new/used/inactive)
deleted/disregarded blocks
execution order
$\mathbf{G}^{\mathrm{BCR}}$
$\mathbf{L}, \mathbf{U}$
1 3 5 7 9 11 13 15 2 6 10 14 4 12 8
$\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$

Figure 4.14: Visualizing the production phase of the block cyclic reduction method for calculating the block tridiagonal inverse $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$ with 15 diagonal blocks in reordered fashion. The $\mathbf{L}$ and $\mathbf{U}$ factors are displayed on the right hand side of the augmented matrices, with blue and green blocks, respectively.

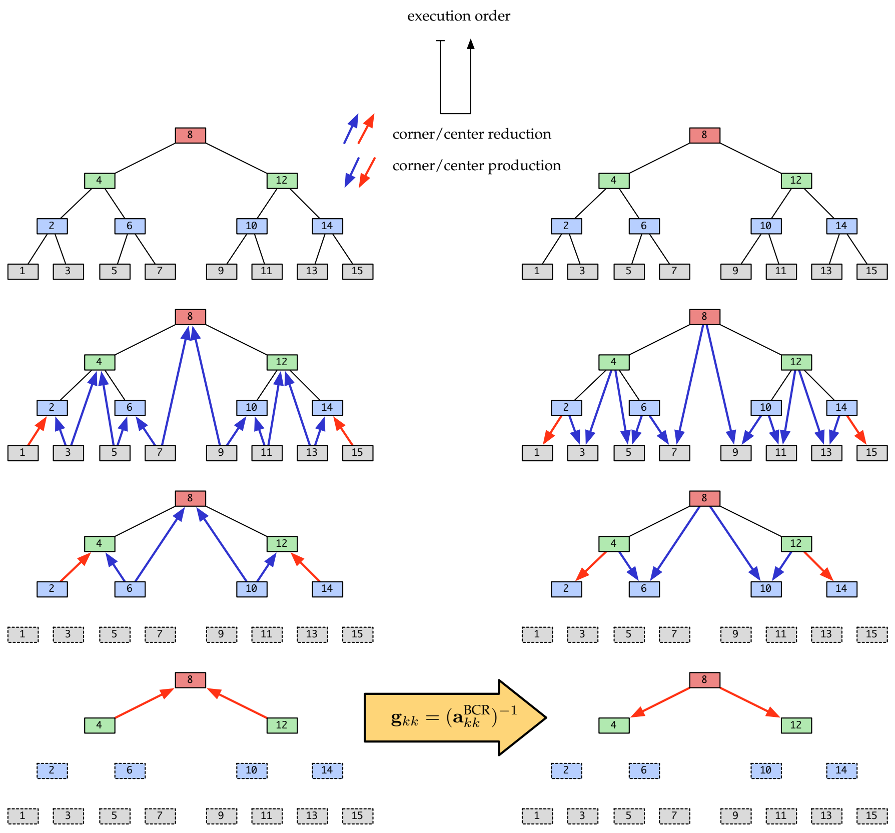
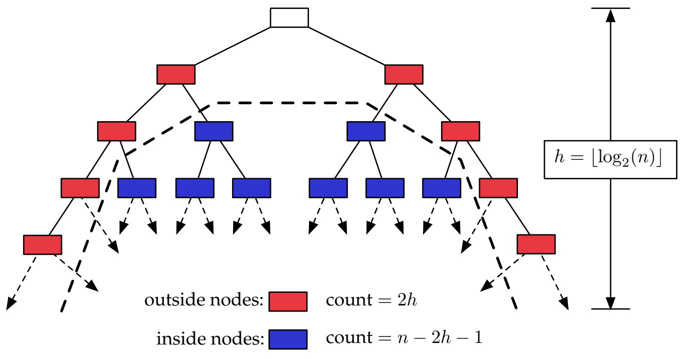


Figure 4.15: The above, in this case complete, binary tree resulting for a 15×15 block matrix $\mathbf{A}$ being treated by block cyclic reduction. The red arrows correspond to corner reduction/production steps, while the blue arrows correspond to center reduction/production steps. Linking the reduction phase on the left to the production phase on the right is the single block inversion $\mathbf{g}_{kk} = (\mathbf{a}_{kk}^{\mathrm{BCR}})^{-1}$ that initializes $\mathbf{G}$.



$h = \lfloor \log_2(n) \rfloor$
outside nodes: 🟥 count $= 2h$
inside nodes: 🟦 count $= n - 2h - 1$

Figure 4.16: This diagram illustrates the different node types in the elimination tree of BCR in terms of whether or not the node will cause a corner reduction/production step or a center reduction/production step. Red nodes on the "outside" of the graph cause corner steps, while blue nodes on the "inside" cause center steps. This is due to the fact that as BCR progresses, the nodes on the outside will consistently be the first/last rows of $\mathbf{A}^{\mathrm{BCR}}$. The root of the tree is not counted, as operations involving it is the single block inversion $\mathbf{g}_{kk} = (\mathbf{a}_{kk}^{\mathrm{BCR}})^{-1}$, as seen in Fig. 4.15.

As can be seen, nodes in the tree fall into one of three categories: the root node $\mathbf{a}_{88}$, an "inside" node, and an "outside" node. The root node is treated separately, as the mathematical operations we will count for it are covered by the large arrow in Fig. 4.15 where $\mathbf{g}_{kk} = \mathbf{g}_{88}$ is determined. The terms of inside and outside node is clarified further in Fig. 4.16, which also provides a count of the different node types for a tree with n nodes. Thus we see that outside nodes will always perform either corner reduction/production steps, while inside nodes will account for the center reduction/production operations. All in all, there will be $2(n - 2h - 1)$ row operations due to inside nodes, as each node updates both its neighbor nodes, and $2h$ row operations due to outside nodes, as it only updates a single neighbor node.

In the reduction phase, each corner reduction needs a single LU factorization, and two multiplications to determine the LU factors involved. A final multiplication and addition are then needed in order to update $\mathbf{A}$, leaving us with a total of $1\mathrm{op}(\mathcal{LU})$, $3\mathrm{op}(\times)$ and $1\mathrm{op}(+)$ for each outside node in the elimination tree.

Regarding an inside node performing center reductions, there will be a single LU factorization, and four multiplications in order to determine the two pairs of LU factors generated as the node needs to update two neighbor nodes, rather than the single neighbor node outside nodes have. The update is then handled by two multiplications and two additions, leading to a total of

$1\text{op}(\mathcal{LU})$, $6\text{op}(\times)$ and $2\text{op}(+)$ for a single inside node.

Something to note is that the algorithm presented for reduction in Alg. (4.8) REDUCE does not explicitly handle determining whether or not $\mathbf{a}_{ii}^{-1}$ or $\mathbf{a}_{kk}^{-1}$ have already been factorized, but this should be done in order to save redundant LU factorizations. Finally, before moving on to the production phase, generation of $\mathbf{g}_{kk}$ is performed requiring a single $\text{op}(\mathcal{LU})$ and a final $\text{op}(\times)$ to determine the inverse block explicitly.

Looking at Alg. (4.10) CORNERPRODUCE, we can count a total of $3\text{op}(\times)$ and a single $\text{op}(+)$ to generate the 3 new inverse blocks. We can save an extra LU factorization if we keep the values stored during the reduction phase, thus we need not calculate $(\mathbf{g}^{\text{BCR}}_{k_{\text{to}},k_{\text{to}}})^{-1}$ again. This algorithm is called once for each outside node in Fig. 4.16.

As for Alg. (4.11) CENTERPRODUCE, we count a total of $10\text{op}(\times)$ and $6\text{op}(+)$ in the generation of the 5 new blocks of the inverse, where again the algorithm is only called once for each inside node in Fig. 4.16. Thus we can now tabulate the operation count involved in BCR in Table 4.2.

Complexity Analysis for Block Cyclic Reduction

Calculation	LU–factorizations $\text{op}(\mathcal{LU})$	Multiplications $\text{op}(\times)$	Additions $\text{op}(+)$
Reduction (outside)	$2h$	$6h$	$2h$
Reduction (inside)	$(n-2h-1)$	$6(n-2h-1)$	$2(n-2h-1)$
Initialize $\mathbf{g}_{kk}$	1	1	0
Production (outside)	0	$6h$	$2h$
Production (inside)	0	$10(n-2h-1)$	$6(n-2h-1)$
INVERSEBCR	n	$16n-20h-15$	$8n-12h-8$

Table 4.2: This table illustrates the amount of basic operations performed in calculating the block tridiagonal inverse $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ using the block cyclic reduction algorithm presented in this section. The second, third and fourth columns refer to the amount of basic matrix operations of LU–factorization, multiplication and addition involved in each algorithm. The term n is the total amount of diagonal blocks in $\mathbf{A} \in \mathbb{B}^{n,n}$ and $h = \lfloor \log_2(n) \rfloor$.

Looking at the results in Table 4.2, we see that BCR used to obtain the block tridiagonal $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ is a linear order method $\mathcal{O}(n)$ in the number of diagonal blocks n of a block tridiagonal matrix $\mathbf{A}$.

Although the prefactor of 16 in the number of matrix multiplies is rather high compared to that of Alg. (3.11) SWEEPINVERSETRI which only has a factor of 7, BCR betters Sweeps in the sense that the prefactor in the number of LU factorizations is 1, where it is 3 for the Sweep algorithm. Furthermore, BCR should have a far greater degree of parallelism over Sweep, despite the overall higher operation count.

4.3.4 Stability

A lot of work has been done concerning the stability of BCR, starting with the stabilization [34, 35] of the first unstable cyclic reduction algorithm developed by Golub. As we can reformulate the process of BCR reduction with an equivalent Gaussian elimination by finding a suitable block permutation matrix $\mathbf{P}$, we can discuss the stability of BCR elimination in terms of how Gaussian elimination is stabilized.

In Gaussian elimination, one is able to employ the technique of either full or partial pivoting in order to ensure stability, however, there is no known way for doing this in BCR [36], due to the lack of block elements to pivot with and the fact that the process of halving the number of unknowns at each step effectively in parallel depends strongly on preserving the layout of block elements in $\mathbf{A}$.

From [32], and recognizing our inability to eliminate with anything but diagonal pivots, we conclude that if elimination with the diagonal blocks is stable, then cyclic reduction itself is stable. It has been noted [37, 33, 38, 39, 32] that if $\mathbf{A}$ is strictly diagonally dominant or symmetric and positive definite, then cyclic reduction will be stable.

By diagonal dominance, we understand the situation where the LU factorization of diagonal blocks is well–behaved and stable. However, we do note the possibility that it could be possible to have a system where Eq. (2.56) may present ill–conditioned diagonal blocks. Subsequent updates of diagonal blocks during the execution of the algorithm may also bring what were perhaps well–conditioned blocks into an ill–conditioned state. These situations however, have not happened in practice during the course of work presented in this thesis.

As $\mathbf{A}$ is indexed by scalar elements over the basis orbitals of all the electrons in a system, and we can order them arbitrarily, the organization of $\mathbf{A}$ into a block tridiagonal form takes place due to an ordering of the orbitals from the left electrode towards the right. This will eventually enable us to subdivide the matrix $\mathbf{A}$ into blocks, where each diagonal block is related to the concept of *principal layers* described earlier, and the off diagonal blocks show the coupling of a principal layer with its nearest neighbor layers.

Due to this arbitrary ordering, it is difficult to determine whether or not the matrix is diagonally dominant, but one can in a hand waving fashion argue that due to nearsightedness and the left–to–right ordering of the orbitals, that the orbital interactions represented in the diagonal blocks of $\mathbf{A}$ are stronger than those in the off diagonal blocks. In other words, most orbitals within a principal layer will interact far more strongly with orbitals within the layer, than with orbitals in the nearest neighbor layers. Thus we can generally assume $\mathbf{A}$ to be diagonally dominant, and BCR should be stable for many if not all physical examples.

4.4 Hybrid Method

So far, we have seen methods that are effective at handling the problem of inversion for the sequential case $\mathcal{P} = 1$ in the case of sweeps and the parallel case $\mathcal{P} \sim n$ in the case of block cyclic reduction.

However, for cases easily encountered in electronic structure calculations, the number n of diagonal blocks in $\mathbf{A}$ is typically much larger than the number of processes $\mathcal{P}$ available. These sorts of problems are typically handled on individual workstations that have 2, 4, or maybe 8 cores, while n for a given matrix $\mathbf{A}$ can generally be made large by looking at problems as large as possible that will fit in memory. This is usually the case since larger problem sizes are closer approximations to reality, and are thus more desirable to solve.

The Hybrid method presented in this section seeks to parallelize the problem of inverting $\mathbf{A}$ effectively by combining the sequential efficiency of the Gaussian elimination sweep based method, with the parallelism of block cyclic reduction.

4.4.1 Description

For the Hybrid method, we assume that the number of processors available for the task is generally much less than the number of diagonal blocks in $\mathbf{A}$, such that $n \gg p$. The Hybrid method itself is visualized in Fig. 4.17.

The method begins by reducing the system $\mathbf{A}$ to be inverted to a smaller system $\mathbf{A}^{\text{Schur}}$ which has on the order of $2\mathcal{P}$ blocks in an approach that is *embarrassingly parallel*[1]. This approach is closely related to determining a Schur complement. This smaller system should be more efficiently handled by BCR as the number of rows it has is on the order of $\mathcal{P}$.

The full block cyclic reduction method is then run on this reduced system from which we obtain the inverse blocks $\mathbf{G}^{\text{Schur}} = \text{Trid}_{\mathbf{A}^{\text{Schur}}}\{\mathbf{G}\}$, corresponding to a certain arrangement of blocks within $\mathbf{G}$. From $\mathbf{G}^{\text{Schur}}$ we then compute

[1]An embarrassingly parallel task is one that can be arbitrarily subdivided among a set of processes and handled without communication.

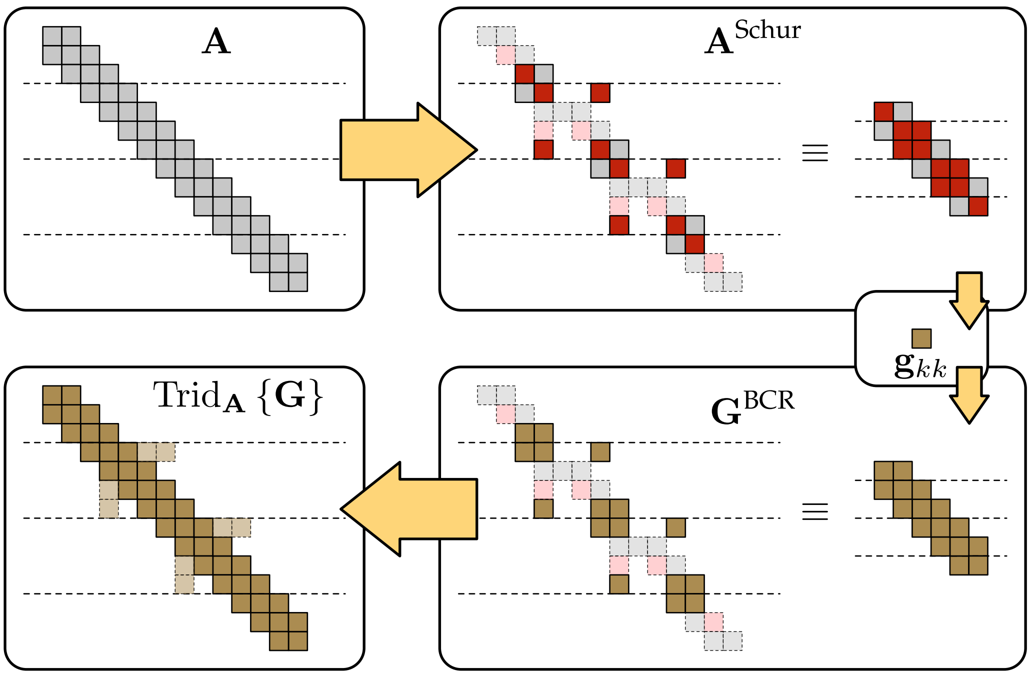

$\mathbf{A}$
$\mathbf{A}^{\text{Schur}}$
$\equiv$
$\mathbf{g}_{kk}$
$\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
$\mathbf{G}^{\text{BCR}}$
$\equiv$

Figure 4.17: The Hybrid method is a method for the computation of the block tridiagonal inverse $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ of a block tridiagonal matrix $\mathbf{A}$. This is accomplished by first performing a series of Schur decompositions in parallel, without communication, followed by a BCR reduction phase until we can determine a single block of the inverse $\mathbf{g}_{kk}$. Following this, a BCR production phase is used to reconstruct the block tridiagonal inverse for the Schur decomposed matrix earlier, followed by a final production phase without communication to determine the remainder of $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$.

our desired $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ by what corresponds to back solving using the Schur complement, and which is again an embarrassingly parallel operation.

The first task of the Hybrid method is to reduce $\mathbf{A}$ to a smaller block tridiagonal system $\mathbf{A}^{\text{Schur}}$ which will be more efficient for the BCR method to handle. This is done on the basis of how $\mathbf{A}$ is divided among processes (cf. Fig. 4.3 or cf. Fig. 4.4), and involves two kinds of reduction operations, depending on whether the rows of $\mathbf{A}$ belong to a *corner* process (**myPID** $= 0$ or **myPID** $= \mathcal{P} - 1$) or a *central* process ($0 <$ **myPID** $< \mathcal{P} - 1$).

4.4.1.1 Corner Schur Reduction

We take our original augmented matrix from Eq. (2.68), and partition each side of it into 4 regions at row/column k, such that we have

$$[\mathbf{A}|\mathbf{I}] = \left(\begin{array}{cccc|c||cccc|c} \mathbf{a}_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & & & \mathbf{i}_{22} & & & \\ & \ddots & \ddots & \ddots & & & & \ddots & & \\ & & \mathbf{a}_{k,k-1} & \mathbf{a}_{kk} & \boldsymbol{\alpha}_{kn} & & & & \mathbf{i}_{kk} & \\ \hline & & & \boldsymbol{\alpha}_{nk} & \boldsymbol{\alpha}_{nn} & & & & & \boldsymbol{\iota}_{nn} \end{array}\right) \tag{4.38}$$

where the block $\boldsymbol{\alpha}_{nn}$ now represents the rest of the block tridiagonal matrix $\mathbf{A}$ for rows/columns $k+1, k+2, \ldots, n$, and $\boldsymbol{\alpha}_{kn}$ and $\boldsymbol{\alpha}_{nk}$ are elongated blocks containing $\mathbf{a}_{k,k+1}$ and $\mathbf{a}_{k+1,k}$, respectively. We then perform a downwards Gaussian elimination sweep, as done in Eq. (3.5), where we stop on reaching row k, getting

$$[\mathbf{A}^{\text{Schur}}|\mathbf{J}^{\text{Schur}}] = \left(\begin{array}{cccc|c||ccccc|c} \mathbf{a}_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & & \\ & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & \mathbf{l}_{21} & \mathbf{i}_{22} & & & & \\ & & \ddots & \ddots & & \circ & \ddots & \ddots & & & \\ & & & \mathbf{d}^L_{kk} & \boldsymbol{\alpha}_{kn} & \circ & \circ & \mathbf{l}_{k,k-1} & \mathbf{i}_{kk} & & \\ \hline & & & \boldsymbol{\alpha}_{nk} & \boldsymbol{\alpha}_{nn} & & & & & & \boldsymbol{\iota}_{nn} \end{array}\right) \tag{4.39}$$

where we have renamed the factors $\mathbf{c}^L_i = -\mathbf{a}_{i+1,i}\mathbf{d}^{-1}_{ii}$ in terms of the LU factors $\mathbf{l}_{i+1,i}$ and the $\circ$ symbols indicate fill–in on the right hand side of the augmented matrix. As we sweep down, we can at the same time calculate the LU factors $\mathbf{u}_{i-1,i} = -\mathbf{a}_{i-1,i}\mathbf{a}^{-1}_{ii}$. Thus we have calculated the Schur complement $\mathbf{d}^L_{kk}$ corresponding to the block matrix composed of the first k rows/columns of $\mathbf{A}$:

$$\left(\begin{array}{cccc|c}
\mathbf{a}_{11} & \mathbf{a}_{12} & & & \\
\mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & \\
 & \ddots & \ddots & \ddots & \\
 & & \mathbf{a}_{k-1,k-2} & \mathbf{a}_{k-1,k-1} & \mathbf{a}_{k-1,k} \\ \hline
 & & & \mathbf{a}_{k,k-1} & \mathbf{a}_{kk}
\end{array}\right). \tag{4.40}$$

The Schur reduction phase for the corner processes can be seen in Fig. 4.18. When the reduction phase is complete, the Hybrid method passes off $\mathbf{A}^{\text{Schur}}$ to BCR, in order to determine the block tridiagonal matrix elements corresponding to the reduced system

$$\left(\begin{array}{cc|cc}
\mathbf{d}^L_{kk} & \boldsymbol{\alpha}_{kn} & \mathbf{i}_{kk} & \\
\boldsymbol{\alpha}_{nk} & \boldsymbol{\alpha}_{nn} & & \boldsymbol{\iota}_{nn}
\end{array}\right). \tag{4.41}$$

In the Hybrid method, however, the block structure within $\boldsymbol{\alpha}_{nn}$ will be preserved, and will be solved for via BCR. In the next section, we solve the above reduced system differently, but only in order to yield the equations for how to reconstruct the rest of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ once BCR returns with its calculated portion of $\mathbf{G}$.

4.4.1.2 Equations for the Solution of the Reduced System

Starting with a row operation between rows k and $k+1 \equiv n$, using the LU factor $\boldsymbol{\lambda}_{nk} = -\boldsymbol{\alpha}_{nk}(\mathbf{d}^L_{kk})^{-1}$, we get

$$\left(\begin{array}{cccc|c||cccc|c}
\mathbf{a}_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & \\
 & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & \mathbf{l}_{21} & \mathbf{i}_{22} & & & \\
 & & \ddots & \ddots & & \circ & \ddots & \ddots & & \\
 & & & \mathbf{d}^L_{kk} & \boldsymbol{\alpha}_{kn} & \circ & \circ & \mathbf{l}_{k,k-1} & \mathbf{i}_{kk} & \\ \hline
 & & & \mathbf{0}_{nk} & \boldsymbol{\delta}^L_{nn} & \circ & \circ & \boldsymbol{\lambda}_{nk}\mathbf{l}_{k,k-1} & \boldsymbol{\lambda}_{nk} & \boldsymbol{\iota}_{nn}
\end{array}\right) \tag{4.42}$$

where $\boldsymbol{\delta}^L_{nn} = \boldsymbol{\alpha}_{nn} + \boldsymbol{\lambda}_{nk}\boldsymbol{\alpha}_{kn}$. At the same time, we can calculate the associated LU factor $\boldsymbol{\upsilon}_{kn} = -(\mathbf{d}^L_{kk})^{-1}\boldsymbol{\alpha}_{kn}$ for later use. We can now solve for the inverse on the bottom row by multiplying across it with $(\boldsymbol{\delta}^L_{nn})^{-1}$, obtaining

$$\left(\begin{array}{cccc|c||cccc|c}
\mathbf{a}_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & \\
 & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & \mathbf{l}_{21} & \mathbf{i}_{22} & & & \\
 & & \ddots & \ddots & & \circ & \ddots & \ddots & & \\
 & & & \mathbf{d}^L_{kk} & \boldsymbol{\alpha}_{kn} & \circ & \circ & \mathbf{l}_{k,k-1} & \mathbf{i}_{kk} & \\ \hline
 & & & & \boldsymbol{\iota}_{nn} & \bullet & \bullet & \boldsymbol{\gamma}_{n,k-1} & \boldsymbol{\gamma}_{nk} & \boldsymbol{\gamma}_{nn}
\end{array}\right) \tag{4.43}$$

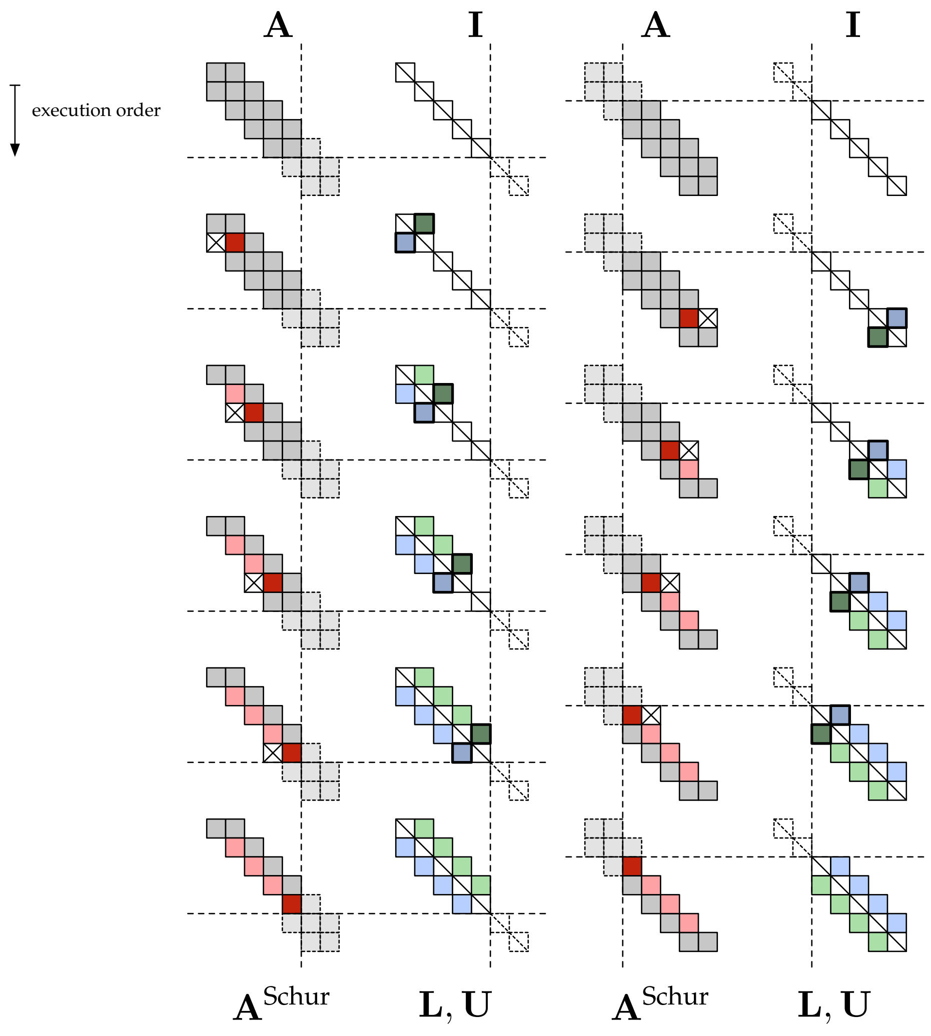


Figure 4.18: The Schur reduction phase for the corner processes, where a Gaussian elimination sweep will yield a Schur complement block, represented by the final bright red block. The LU factors $\mathbf{l}$ and $\mathbf{u}$ are stored, and their positions are given with blue and green blocks, respectively.

and where we write $\bullet$ to indicate blocks of the inverse in order to save space. We now multiply row $k+1 \equiv n$ with $-\boldsymbol{\alpha}_{kn}$ and add it to row k, and then multiply across row k with $(\mathbf{d}^L_{kk})^{-1}$ to obtain

$$\left(\begin{array}{cccc|c||cccc|c}
\mathbf{a}_{11} & \mathbf{a}_{12} & & & & \mathbf{i}_{11} & & & & \\
 & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & \mathbf{l}_{21} & \mathbf{i}_{22} & & & \\
 & & \ddots & \ddots & & \circ & \ddots & \ddots & & \\
 & & & \mathbf{i}_{kk} & \mathbf{0}_{kn} & \bullet & \bullet & \mathbf{g}_{k,k-1} & \mathbf{g}_{kk} & \boldsymbol{\gamma}_{kn} \\
\hline
 & & & & \boldsymbol{\iota}_{nn} & \bullet & \bullet & \boldsymbol{\gamma}_{n,k-1} & \boldsymbol{\gamma}_{nk} & \boldsymbol{\gamma}_{nn}
\end{array}\right) \tag{4.44}$$

where we have

$$\begin{aligned}
\mathbf{g}_{kk} &= (\mathbf{d}^L_{kk})^{-1}(\mathbf{i}_{kk} - \boldsymbol{\alpha}_{kn}\boldsymbol{\gamma}_{nk}) \\
&= (\mathbf{d}^L_{kk})^{-1} + \boldsymbol{\upsilon}_{kn}\boldsymbol{\gamma}_{nk}
\end{aligned} \tag{4.45}$$

and

$$\begin{aligned}
\mathbf{g}_{kn} &= -(\mathbf{d}^L_{kk})^{-1}\boldsymbol{\alpha}_{kn}\boldsymbol{\gamma}_{nn} \\
&= \boldsymbol{\upsilon}_{kn}\boldsymbol{\gamma}_{nn}
\end{aligned} \tag{4.46}$$

where the LU factor $\boldsymbol{\upsilon}_{kn} = -(\mathbf{d}^L_{kk})^{-1}\boldsymbol{\alpha}_{kn}$ is the related LU factor to $\boldsymbol{\lambda}_{nk}$, calculated earlier. To reiterate, the solution of these Green's function blocks is accomplished by a BCR solution phase in the Hybrid method, but we are otherwise free to choose a method. Once this is accomplished, we find ourselves in the situation of having to determine the block tridiagonal inverse for the first k rows/columns of $\mathbf{A}$ using the block $\mathbf{g}_{kk}$ returned via BCR, and the LU factors preserved during the Schur corner reduction phase.

4.4.1.3 Corner Schur Production

The corner Schur production phase seeks to determine the Green's function blocks $\mathbf{g}_{i+1,i}$, $\mathbf{g}_{i,i+1}$ and $\mathbf{g}_{ii}$ for the values $i = k-1, k-2, \ldots, 1$, since we assume the rest of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ has been determined. Looking now at $\mathbf{g}_{k,k-1}$, we have via the row operation from Eq. (4.43) leading to Eq. (4.44) that

$$\begin{aligned}
\mathbf{g}_{k,k-1} &= (\mathbf{d}^L_{kk})^{-1}(\mathbf{l}_{k,k-1} - \boldsymbol{\alpha}_{kn}\boldsymbol{\gamma}_{n,k-1}) \\
&= (\mathbf{d}^L_{kk})^{-1}(\mathbf{l}_{k,k-1} - \boldsymbol{\alpha}_{kn}\boldsymbol{\gamma}_{nk}\mathbf{l}_{k,k-1}) \\
&= (\mathbf{d}^L_{kk})^{-1}(\mathbf{i}_{k,k} - \boldsymbol{\alpha}_{kn}\boldsymbol{\gamma}_{nk})\mathbf{l}_{k,k-1} \\
&= ((\mathbf{d}^L_{kk})^{-1} + \boldsymbol{\upsilon}_{kn}\boldsymbol{\gamma}_{nk})\mathbf{l}_{k,k-1} \\
&= \mathbf{g}_{kk}\mathbf{l}_{k,k-1}
\end{aligned} \tag{4.47}$$

Thus we have defined $\mathbf{g}_{k,k-1}$ based only on the knowledge of what $\mathbf{g}_{kk}$ is, and an LU factor derived in the Schur decomposition of the 2×2 partitioned system given in Eq. (4.40).

To obtain $\mathbf{g}_{k-1,k}$ and $\mathbf{g}_{k-1,k-1}$, we perform a row operation upwards on Eq. (4.44) using the stored LU factor $\mathbf{u}_{k-1,k}$, and multiply across row $k-1$ with $(\mathbf{d}^L_{k-1,k-1})^{-1}$, obtaining

$$\left(\begin{array}{ccccc|c||cccccc|c}
\mathbf{a}_{11} & \mathbf{a}_{12} & & & & & \mathbf{i}_{11} & & & & & \\
 & \mathbf{d}^L_{22} & \mathbf{a}_{23} & & & & \mathbf{l}_{21} & \mathbf{i}_{22} & & & & \\
 & & \ddots & \ddots & & & \vdots & \vdots & \ddots & & & \\
 & & & \mathbf{i}_{k-1,k-1} & \mathbf{0}_{k-1,k} & & \bullet & \bullet & \bullet & \mathbf{g}_{k-1,k-1} & \mathbf{g}_{k-1,k} & \bullet \\
 & & & & \mathbf{i}_{kk} & & \bullet & \bullet & \bullet & \mathbf{g}_{k,k-1} & \mathbf{g}_{kk} & \bullet \\
\hline
 & & & & & \boldsymbol{\iota}_{nn} & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet
\end{array}\right) \tag{4.48}$$

where

$$\begin{aligned}
\mathbf{g}_{k-1,k} &= (\mathbf{d}^L_{k-1,k-1})^{-1}(-\mathbf{a}_{k-1,k}\mathbf{g}_{kk}) \\
&= \mathbf{u}_{k-1,k}\mathbf{g}_{kk}
\end{aligned} \tag{4.49}$$

and

$$\begin{aligned}
\mathbf{g}_{k-1,k-1} &= (\mathbf{d}^L_{k-1,k-1})^{-1}(\mathbf{i}_{k-1,k-1} - \mathbf{a}_{k-1,k}\mathbf{g}_{k,k-1}) \\
&= (\mathbf{d}^L_{k-1,k-1})^{-1} + \mathbf{u}_{k-1,k}\mathbf{g}_{k,k-1}
\end{aligned} \tag{4.50}$$

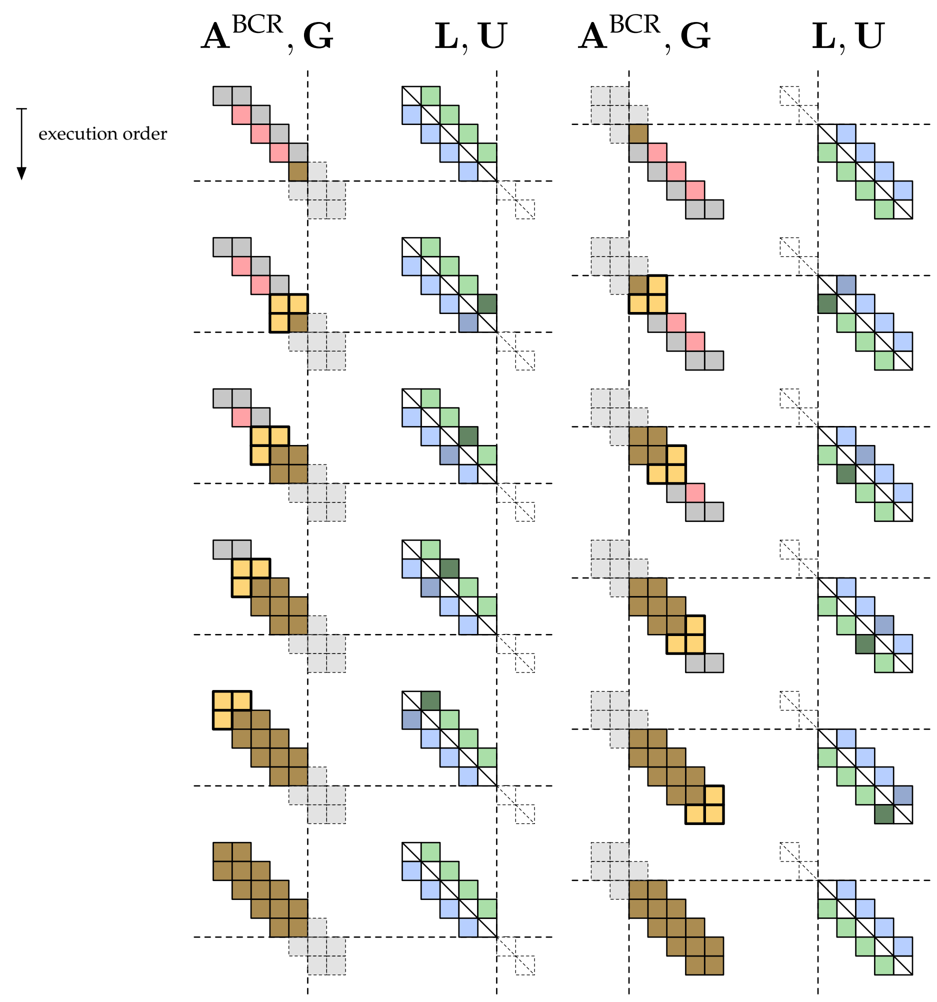
We can then use Eq. (4.47), Eq. (4.49) and Eq. (4.50) recursively in the production of our desired tridiagonal portion of $\mathbf{G}$ as they are independent of any relation to row $k+1 \equiv n$, producing the following equations for $i = k-1, k-1, \ldots, 1$.

$$\mathbf{g}_{i+1,i} = \mathbf{g}_{i+1,i+1}\mathbf{l}_{i+1,i} \tag{4.51}$$
$$\mathbf{g}_{i,i+1} = \mathbf{u}_{i,i+1}\mathbf{g}_{i+1,i+1} \tag{4.52}$$
$$\mathbf{g}_{ii} = (\mathbf{d}^L_{ii})^{-1} + \mathbf{u}_{i,i+1}\mathbf{g}_{i+1,i} \tag{4.53}$$

The derivation and formulas are mirrored for the case of the Schur reduction/production of the lower corner portion of $\mathbf{A}$ for process **myPID** $= \mathcal{P} - 1$. The corner Schur production phases are visualized in Fig. 4.19, for both the upper and lower corner processes of $\mathbf{A}$.

4.4.1.4 Center Schur Reduction

We now look at the Schur reduction that a process owning rows in $\mathbf{A}$ that are neither at the top or bottom. We again take our original augmented matrix



$\mathbf{A}^{\mathrm{BCR}}, \mathbf{G}$
$\mathbf{L}, \mathbf{U}$
$\mathbf{A}^{\mathrm{BCR}}, \mathbf{G}$
$\mathbf{L}, \mathbf{U}$
execution order

Figure 4.19: The Schur production phase for the corner processes, where the block tridiagonal portion of **G** is reconstructed on the corners once an initial inverse block is returned from the BCR phase of the Hybrid method. The used LU factors in constructing the new (brightened) inverse blocks is indicated by darkened LU blocks.

from Eq. (2.68), and now partition it at rows/columns j and k, such that we have

$$\left(\begin{array}{c|cccc|c||c|cccc|c}
\boldsymbol{\alpha}_{11} & \boldsymbol{\alpha}_{1j} & & & & & \boldsymbol{\iota}_{11} & & & & & \\
\hline
\boldsymbol{\alpha}_{j1} & \mathbf{a}_{jj} & \mathbf{a}_{j,j+1} & & & & & \mathbf{i}_{jj} & & & & \\
 & \mathbf{a}_{j+1,j} & \mathbf{a}_{j+1,j+1} & \ddots & & & & & \mathbf{i}_{j+1,j+1} & & & \\
 & & \ddots & \ddots & \mathbf{a}_{k-1,k} & & & & & \ddots & & \\
 & & & \mathbf{a}_{k,k-1} & \mathbf{a}_{kk} & \boldsymbol{\alpha}_{kn} & & & & & \mathbf{i}_{kk} & \\
\hline
 & & & & \boldsymbol{\alpha}_{nk} & \boldsymbol{\alpha}_{nn} & & & & & & \boldsymbol{\iota}_{nn}
\end{array}\right) \tag{4.54}$$

where $\boldsymbol{\alpha}_{11}$ now represents the block tridiagonal part of $\mathbf{A}$ for the first upper rows $1, 2, \ldots, j-1$, and $\boldsymbol{\alpha}_{nn}$ encompasses the block tridiagonal part of $\mathbf{A}$ for the last lower rows $k+1, k+2, \ldots, n$. The coupling matrices $\boldsymbol{\alpha}_{1j}$, $\boldsymbol{\alpha}_{j1}$, $\boldsymbol{\alpha}_{kn}$ and $\boldsymbol{\alpha}_{nk}$ are elongated, where their nontrivial content corresponds to what was $\mathbf{a}_{j-1,j}$, $\mathbf{a}_{j,j-1}$, $\mathbf{a}_{n-1,n}$ and $\mathbf{a}_{n,n-1}$, respectively. The identity matrices $\boldsymbol{\iota}_{11}$ and $\boldsymbol{\iota}_{nn}$ are likewise sized to include the identity matrices $\mathbf{i}_{11}, \mathbf{i}_{22}, \ldots, \mathbf{i}_{j-1,j-1}$ and $\mathbf{i}_{k+1,k+1}, \mathbf{i}_{k+2,k+2}, \ldots, \mathbf{i}_{nn}$, respectively.

Looking at Eq. (4.54), we start a series of steps that seek to eliminate all subdiagonal blocks $\mathbf{a}_{j+2,j+1}, \mathbf{a}_{j+3,j+2}, \ldots, \mathbf{a}_{k,k-1}$ causing fill–in along column j from row $j+2$ down to row k. At the same time, we seek to eliminate the superdiagonal block $\mathbf{a}_{j,j+1}$ with row $j+1$. This causes fill–in at position $(j, j+2)$, which we then eliminate with row $j+2$, and so forth, until we have created a final fill in block at position (jk). This procedure can be seen in Fig. 4.20,

If we perform the following permutation on $\mathbf{A}$, where $\mathbf{P}$ is a block permutation matrix, such that we have the following 3×3 block form,

$$\mathbf{PAP} =$$
$$\left(\begin{array}{cccc|cc|cc}
\mathbf{a}_{j+1,j+1} & \mathbf{a}_{j+1,j+2} & & & \mathbf{a}_{j+1,j} & & & \\
\mathbf{a}_{j+2,j+1} & \ddots & \ddots & & & & & \\
 & \ddots & \mathbf{a}_{k-2,k-2} & \mathbf{a}_{k-2,k-1} & & & & \\
 & & \mathbf{a}_{k-1,k-2} & \mathbf{a}_{k-1,k-1} & & \mathbf{a}_{k-1,k} & & \\
\hline
\mathbf{a}_{j,j+1} & & & & \mathbf{a}_{jj} & & \boldsymbol{\alpha}_{j1} & \\
 & & & \mathbf{a}_{k,k-1} & & \mathbf{a}_{kk} & & \boldsymbol{\alpha}_{kn} \\
\hline
 & & & & \boldsymbol{\alpha}_{1j} & & \boldsymbol{\alpha}_{11} & \\
 & & & & & \boldsymbol{\alpha}_{nk} & & \boldsymbol{\alpha}_{nn}
\end{array}\right), \tag{4.55}$$

we can then see that the series of block eliminations in Fig. 4.20 corresponds to the calculation of the Schur complement for the upper left 2×2 partition in

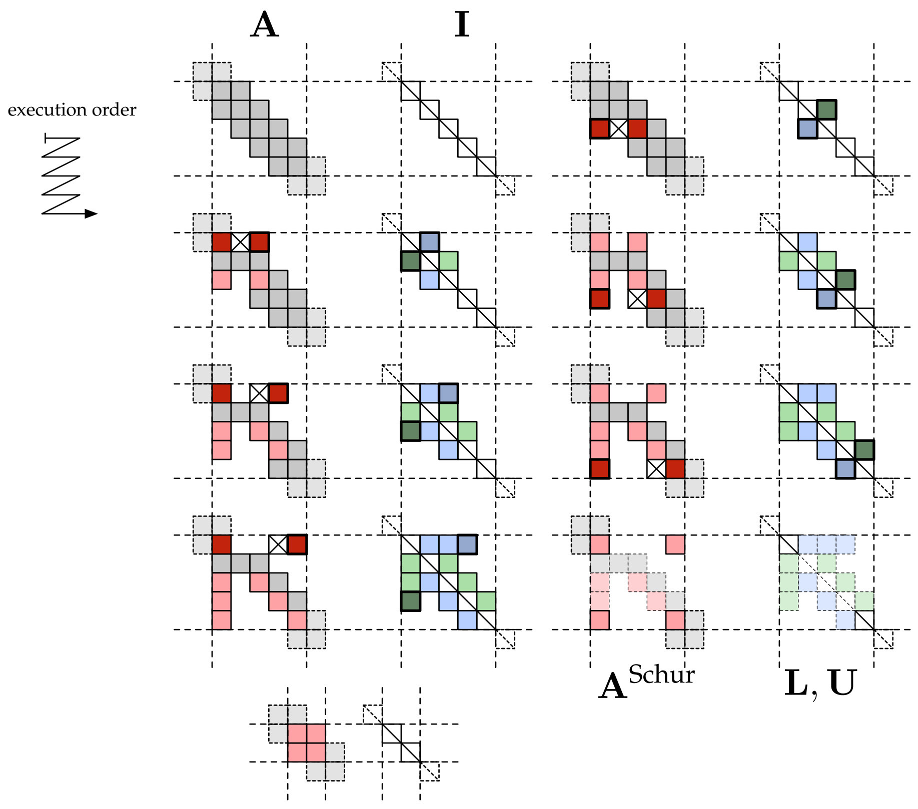


Figure 4.20: This figure shows the Schur reduction phase for a central process owning rows in **A**. As the elimination proceeds, the bright red blocks indicate updates in **A**, and when they have propagated to the corners of the section of **A** we have partitioned, we are finished, and remain with the "reduce" block tridiagonal system as the bottom.

Eq. (4.55), leaving us with

$$\mathbf{P}\mathbf{A}^{\text{Schur}}\mathbf{P} =$$

$$\left(\begin{array}{cccc|cc|cc||cccc|cc|cc}
\mathbf{a}_{j+1,j+1} & \mathbf{a}_{j+1,j+2} & & & \mathbf{a}_{j+1,j} & & & & \mathbf{i}_{j+1,j+1} & \square & & & \square & & & \\
 & \ddots & \ddots & & \vdots & & & & \mathbf{l}_{j+2,j+1} & \ddots & \square & & \square & & & \\
 & & \mathbf{d}_{k-2,k-2} & \mathbf{a}_{k-2,k-1} & \mathbf{f}_{k-2,j} & & & & \circ & \ddots & \mathbf{i}_{k-2,k-2} & \square & \square & & & \\
 & & & \mathbf{d}_{k-1,k-1} & \mathbf{f}_{k-1,j} & \mathbf{a}_{k-1,k} & & & \circ & \circ & \mathbf{l}_{k-1,k-2} & \mathbf{i}_{k-1,k-1} & \square & \square & & \\
\hline
 & & & & \mathbf{s}_{jj} & \mathbf{s}_{jk} & \boldsymbol{\alpha}_{j1} & & \mathbf{l}_{j,j+1} & \cdots & \mathbf{l}_{j,k-2} & \mathbf{l}_{j,k-1} & \mathbf{i}_{jj} & & & \\
 & & & & \mathbf{s}_{kj} & \mathbf{s}_{kk} & & \boldsymbol{\alpha}_{kn} & \circ & \cdots & \mathbf{l}_{k,k-1}\mathbf{l}_{k-1,k-2} & \mathbf{l}_{k,k-1} & & \mathbf{i}_{kk} & & \\
\hline
 & & & & \boldsymbol{\alpha}_{1j} & & \boldsymbol{\alpha}_{11} & & & & & & & & \boldsymbol{\iota}_{11} & \\
 & & & & & \boldsymbol{\alpha}_{nk} & & \boldsymbol{\alpha}_{nn} & & & & & & & & \boldsymbol{\iota}_{nn}
\end{array}\right) \quad (4.56)$$

where the 2×2 block composed of $\mathbf{s}_{jj}$, $\mathbf{s}_{jk}$, $\mathbf{s}_{kj}$ and $\mathbf{s}_{kk}$ is the calculated Schur complement, $\mathbf{f}_{ij}$ is fill–in in location (i, j) in $\mathbf{A}$, $\circ$ denotes fill–in and $\square$ denotes placeholder[1] locations of the LU $\mathbf{u}_{ji}$ factors corresponding to the $\mathbf{l}_{ij}$ factors. It is at this point that $\mathbf{A}^{\text{Schur}}$ is passed off to a BCR solution process in order to solve for the Green's function blocks on the reduced system corresponding to the lower 2×2 partition in Eq. (4.56).

4.4.1.5 Equations for the Solution of the Reduced System

Taking Eq. (4.56), we rename its larger partitions in the following manner

$$\mathbf{P}\mathbf{A}^{\text{Schur}}\mathbf{P} = \left(\begin{array}{c|c|c||c|c|c}
\mathbf{A}' & \mathbf{F} & & \mathbf{L}_0 & & \\
\hline
 & \mathbf{S} & \mathbf{Y} & \mathbf{L}_1 & \mathbf{I}_{\mathbf{S}} & \\
\hline
 & \mathbf{X} & \mathbf{Z} & & & \mathbf{I}_{\mathbf{Z}}
\end{array}\right) \quad (4.57)$$

and which we now seek to continue to manipulate further to obtain $\mathbf{G}$. This phase is handled in the Hybrid method via BCR, but we perform it here explicitly in order to yield the equations that will tell us how to conduct the

[1]The $\square$ symbol in this case only indicates where a LU factor $\mathbf{u}$ would be found, and does not represent actual content in the right hand side of the augmented matrix expression.

production phase to reconstruct $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ for rows $j, j+1, \ldots, k$ once BCR has finished with the reduced system corresponding to

$$\left(\begin{array}{cc|cc} \mathbf{S} & \mathbf{Y} & \mathbf{I_S} & \\ \mathbf{X} & \mathbf{Z} & & \mathbf{I_Z} \end{array}\right). \tag{4.58}$$

Thus continuing from Eq. (4.57), we eliminate $\mathbf{X}$ with a row operation to obtain

$$\left(\begin{array}{c|c|c||c|c|c} \mathbf{A}' & \mathbf{F} & & \mathbf{L}_0 & & \\ \hline & \mathbf{S} & \mathbf{Y} & \mathbf{L}_1 & \mathbf{I_S} & \\ \hline & & \mathbf{Z}+\mathbf{L_X}\mathbf{Y} & \mathbf{L_X}\mathbf{L}_1 & \mathbf{L_X} & \mathbf{I_Z} \end{array}\right). \tag{4.59}$$

This is followed by determining the inverse on the bottom row by multiplying across with $(\mathbf{Z}+\mathbf{L_X}\mathbf{Y})^{-1}$, getting

$$\left(\begin{array}{c|c|c||c|c|c} \mathbf{A}' & \mathbf{F} & & \mathbf{L}_0 & & \\ \hline & \mathbf{S} & \mathbf{Y} & \mathbf{L}_1 & \mathbf{I_S} & \\ \hline & & \mathbf{I_Z} & \mathbf{G_Z}\mathbf{L_X}\mathbf{L}_1 & \mathbf{G_Z}\mathbf{L_X} & \mathbf{G_Z} \end{array}\right) \tag{4.60}$$

where $\mathbf{G_Z} = (\mathbf{Z}+\mathbf{L_X}\mathbf{Y})^{-1}$. We then eliminate $\mathbf{Y}$ by multiplying the bottom row by $-\mathbf{Y}$ and adding it to the middle row to get

$$\left(\begin{array}{c|c|c||c|c|c} \mathbf{A}' & \mathbf{F} & & \mathbf{L}_0 & & \\ \hline & \mathbf{S} & & \mathbf{L}_1-\mathbf{Y}\mathbf{G_Z}\mathbf{L_X}\mathbf{L}_1 & \mathbf{I_S}-\mathbf{Y}\mathbf{G_Z}\mathbf{L_X} & -\mathbf{Y}\mathbf{G_Z} \\ \hline & & \mathbf{I_Z} & \mathbf{G_Z}\mathbf{L_X}\mathbf{L}_1 & \mathbf{G_Z}\mathbf{L_X} & \mathbf{G_Z} \end{array}\right) \tag{4.61}$$

where we can get the inverse on the second row by multiplying across with $\mathbf{S}^{-1}$, yielding

$$\left(\begin{array}{c|c|c||c|c|c} \mathbf{A}' & \mathbf{F} & & \mathbf{L}_0 & & \\ \hline & \mathbf{I_S} & & \mathbf{S}^{-1}(\mathbf{L}_1-\mathbf{Y}\mathbf{G_Z}\mathbf{L_X}\mathbf{L}_1) & \mathbf{S}^{-1}(\mathbf{I_S}-\mathbf{Y}\mathbf{G_Z}\mathbf{L_X}) & \mathbf{S}^{-1}(-\mathbf{Y}\mathbf{G_Z}) \\ \hline & & \mathbf{I_Z} & \mathbf{G_Z}\mathbf{L_X}\mathbf{L}_1 & \mathbf{G_Z}\mathbf{L_X} & \mathbf{G_Z} \end{array}\right). \tag{4.62}$$

Looking at the inverse blocks on the positions corresponding to the Schur complement $\mathbf{S}$, we see

$$\mathbf{G_S} = \mathbf{S}^{-1}(\mathbf{I_S}-\mathbf{Y}\mathbf{G_Z}\mathbf{L_X}) = \begin{pmatrix} \mathbf{g}_{jj} & \mathbf{g}_{jk} \\ \mathbf{g}_{kj} & \mathbf{g}_{kk} \end{pmatrix} \tag{4.63}$$

and thus we expect to return from the BCR phase in the Hybrid method with the inverse blocks of $\mathbf{G}$ determined in these positions.

Looking at the adjacent blocks of the inverse in the positions corresponding to $\mathbf{L}_1$ in Eq. (4.57), we see

$$
\begin{aligned}
\mathbf{G}_{\mathbf{L}_1} &= \mathbf{S}^{-1}(\mathbf{L}_1 - \mathbf{Y}\mathbf{G}_{\mathbf{Z}}\mathbf{L}_{\mathbf{X}}\mathbf{L}_1) && (4.64)\\
&= \mathbf{S}^{-1}(\mathbf{I}_{\mathbf{S}} - \mathbf{Y}\mathbf{G}_{\mathbf{Z}}\mathbf{L}_{\mathbf{X}})\mathbf{L}_1 && (4.65)\\
&= \mathbf{G}_{\mathbf{S}}\mathbf{L}_1 && (4.66)
\end{aligned}
$$

which we can expand to get

$$
\begin{aligned}
&\begin{pmatrix} \mathbf{g}_{j,j+1} & \cdots & \mathbf{g}_{j,k-2} & \mathbf{g}_{j,k-1} \\ \mathbf{g}_{k,j+1} & \cdots & \mathbf{g}_{k,k-2} & \mathbf{g}_{k,k-1} \end{pmatrix} \\
&\quad = \begin{pmatrix} \mathbf{g}_{jj} & \mathbf{g}_{jk} \\ \mathbf{g}_{kj} & \mathbf{g}_{kk} \end{pmatrix} \begin{pmatrix} \mathbf{l}_{j,j+1} & \cdots & \mathbf{l}_{j,k-2} & \mathbf{l}_{j,k-1} \\ \circ & \cdots & \mathbf{l}_{k,k-1}\mathbf{l}_{k-1,k-2} & \mathbf{l}_{k,k-1} \end{pmatrix} \\
&\quad = \begin{pmatrix} \bullet & \cdots & (\mathbf{g}_{jj}\mathbf{l}_{j,k-2} + \mathbf{g}_{jk}\mathbf{l}_{k,k-1}\mathbf{l}_{k-1,k-2}) & (\mathbf{g}_{jj}\mathbf{l}_{j,k-1} + \mathbf{g}_{jk}\mathbf{l}_{k,k-1}) \\ \bullet & \cdots & (\mathbf{g}_{kj}\mathbf{l}_{j,k-2} + \mathbf{g}_{kk}\mathbf{l}_{k,k-1}\mathbf{l}_{k-1,k-2}) & (\mathbf{g}_{kj}\mathbf{l}_{j,k-1} + \mathbf{g}_{kk}\mathbf{l}_{k,k-1}) \end{pmatrix} \\
&\quad = \begin{pmatrix} \bullet & \cdots & (\mathbf{g}_{jj}\mathbf{l}_{j,k-1}\mathbf{l}_{k-1,k-2} + \mathbf{g}_{jk}\mathbf{l}_{k,k-1}\mathbf{l}_{k-1,k-2}) & \mathbf{g}_{j,k-1} \\ \bullet & \cdots & (\mathbf{g}_{kj}\mathbf{l}_{j,k-1}\mathbf{l}_{k-1,k-2} + \mathbf{g}_{kk}\mathbf{l}_{k,k-1}\mathbf{l}_{k-1,k-2}) & \mathbf{g}_{k,k-1} \end{pmatrix} \\
&\quad = \begin{pmatrix} \bullet & \cdots & (\mathbf{g}_{jj}\mathbf{l}_{j,k-1} + \mathbf{g}_{jk}\mathbf{l}_{k,k-1})\mathbf{l}_{k-1,k-2} & \mathbf{g}_{j,k-1} \\ \bullet & \cdots & (\mathbf{g}_{kj}\mathbf{l}_{j,k-1} + \mathbf{g}_{kk}\mathbf{l}_{k,k-1})\mathbf{l}_{k-1,k-2} & \mathbf{g}_{k,k-1} \end{pmatrix} \\
&\quad = \begin{pmatrix} \bullet & \cdots & \mathbf{g}_{j,k-1}\mathbf{l}_{k-1,k-2} & \mathbf{g}_{j,k-1} \\ \bullet & \cdots & \mathbf{g}_{k,k-1}\mathbf{l}_{k-1,k-2} & \mathbf{g}_{k,k-1} \end{pmatrix} \\
&\quad = \begin{pmatrix} \bullet & \cdots & \mathbf{g}_{j,k-2} & \mathbf{g}_{j,k-1} \\ \bullet & \cdots & \mathbf{g}_{k,k-2} & \mathbf{g}_{k,k-1} \end{pmatrix}
\end{aligned}
\tag{4.67}
$$

and thus we can express these blocks in terms of the stored LU factors $\mathbf{l}$ and the inverse blocks $\mathbf{G}_{\mathbf{S}}$ returned by BCR. This leads us to obtaining our first desired block of the inverse

$$\mathbf{g}_{k,k-1} = \mathbf{g}_{kj}\mathbf{l}_{j,k-1} + \mathbf{g}_{kk}\mathbf{l}_{k,k-1} \tag{4.68}$$

and we can recognize the ability to generate recursively more blocks of the inverse in $\mathbf{G}_{\mathbf{L}_1}$ using stored LU factors by

$$
\begin{aligned}
\mathbf{g}_{ji} &= \mathbf{g}_{j,i+1}\mathbf{l}_{i+1,i} && (4.69)\\
\mathbf{g}_{ki} &= \mathbf{g}_{k,i+1}\mathbf{l}_{i+1,i} && (4.70)
\end{aligned}
$$

for $i = k-2, k-3, \ldots, j+1$. Thus we have back solved up to the following form:

$$
\left(\begin{array}{cccc|cc|cc}
\mathbf{a}_{j+1,j+1} & \mathbf{a}_{j+1,j+2} & & & \mathbf{a}_{j+1,j} & & & \\
 & \ddots & \ddots & & \vdots & & & \\
 & & \mathbf{d}_{k-2,k-2} & \mathbf{a}_{k-2,k-1} & \mathbf{f}_{k-2,j} & & & \\
 & & & \mathbf{d}_{k-1,k-1} & \mathbf{f}_{k-1,j} & \mathbf{a}_{k-1,k} & & \\
\hline
 & & & & \mathbf{i}_{jj} & & & \\
 & & & & & \mathbf{i}_{kk} & & \\
\hline
 & & & & & & \boldsymbol{\iota}_{11} & \\
 & & & & & & & \boldsymbol{\iota}_{nn}
\end{array}\right\|
\left\|\begin{array}{cccc|cc|cc}
\mathbf{i}_{j+1,j+1} & \circ & & & \circ & & & \\
\mathbf{l}_{j+2,j+1} & \ddots & \circ & & \circ & & & \\
\circ & \ddots & \mathbf{i}_{k-2,k-2} & \circ & \circ & & & \\
\circ & \circ & \mathbf{l}_{k-1,k-2} & \mathbf{i}_{k-1,k-1} & \circ & \circ & & \\
\hline
\mathbf{g}_{j,j+1} & \cdots & \mathbf{g}_{j,k-2} & \mathbf{g}_{j,k-1} & \mathbf{g}_{jj} & \mathbf{g}_{jk} & \bullet & \bullet \\
\mathbf{g}_{k,j+1} & \cdots & \mathbf{g}_{k,k-2} & \mathbf{g}_{k,k-1} & \mathbf{g}_{kj} & \mathbf{g}_{kk} & \bullet & \bullet \\
\hline
\bullet & \cdots & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \cdots & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet
\end{array}\right). \tag{4.71}
$$

4.4.1.6 Schur Center Production

We are now ready to look for a recursive method which can reconstruct the block tridiagonal inverse for the rows $j, j+1, \ldots, k$ involved in the earlier center Schur reduction phase. Working from Eq. (4.71), and looking first for $\mathbf{g}_{k-1,k}$, we attain it by first eliminating $\mathbf{a}_{k-1,k}$ and $\mathbf{f}_{k-1,j}$, and then multiplying across by $\mathbf{d}_{k-1,k-1}^{-1}$, obtaining the inverse on row $k-1$,

$$\left(\begin{array}{cccc|cc|cc}
\mathbf{a}_{j+1,j+1} & \mathbf{a}_{j+1,j+2} & & & \mathbf{a}_{j+1,j} & & & \\
& \ddots & \ddots & & \vdots & & & \\
& & \mathbf{d}_{k-2,k-2} & \mathbf{a}_{k-2,k-1} & \mathbf{f}_{k-2,j} & & & \\
& & & \mathbf{i}_{k-1,k-1} & & & & \\
\hline
& & & & \mathbf{i}_{jj} & & & \\
& & & & & \mathbf{i}_{kk} & & \\
\hline
& & & & & & \boldsymbol{\iota}_{11} & \\
& & & & & & & \boldsymbol{\iota}_{nn}
\end{array}\right\Vert$$

$$\left\Vert\begin{array}{cccc|cc|cc}
\mathbf{i}_{j+1,j+1} & \circ & & & \circ & & & \\
\mathbf{l}_{j+2,j+1} & \ddots & \circ & & \circ & & & \\
& \ddots & \mathbf{i}_{k-2,k-2} & \circ & \circ & & & \\
\bullet & \cdots & \mathbf{g}_{k-1,k-2} & \mathbf{g}_{k-1,k-1} & \mathbf{g}_{k-1,j} & \mathbf{g}_{k-1,k} & \bullet & \bullet \\
\hline
\mathbf{g}_{j,j+1} & \cdots & \mathbf{g}_{j,k-2} & \mathbf{g}_{j,k-1} & \mathbf{g}_{jj} & \mathbf{g}_{jk} & \bullet & \bullet \\
\mathbf{g}_{k,j+1} & \cdots & \mathbf{g}_{k,k-2} & \mathbf{g}_{k,k-1} & \mathbf{g}_{kj} & \mathbf{g}_{kk} & \bullet & \bullet \\
\hline
\bullet & \cdots & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \cdots & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet
\end{array}\right) \tag{4.72}$$

where

$$\begin{aligned}
\mathbf{g}_{k-1,k} &= \mathbf{d}_{k-1,k-1}^{-1}(-\mathbf{a}_{k-1,k}\mathbf{g}_{kk} - \mathbf{f}_{k-1,j}\mathbf{g}_{jk}) \\
&= \mathbf{u}_{k-1,k}\mathbf{g}_{kk} + \mathbf{u}_{k-1,j}\mathbf{g}_{jk} \qquad (4.73)
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{g}_{k-1,k-1} &= \mathbf{d}_{k-1,k-1}^{-1}(\mathbf{i}_{k-1,k-1} - \mathbf{a}_{k-1,k}\mathbf{g}_{k,k-1} - \mathbf{f}_{k-1,j}\mathbf{g}_{j,k-1}) \\
&= \mathbf{d}_{k-1,k-1}^{-1} + \mathbf{u}_{k-1,k}\mathbf{g}_{k,k-1} + \mathbf{u}_{k-1,j}\mathbf{g}_{j,k-1}. \qquad (4.74)
\end{aligned}$$

Continuing this process, we can now define a recursion for $i = k-1, k-2, \ldots, j+2$ for determining the block tridiagonal portion of $\mathbf{G}$ on rows $k-1, k-2, \ldots, j+1$:

$$\mathbf{g}_{ii} = \mathbf{d}_{ii}^{-1} + \mathbf{u}_{ij}\mathbf{g}_{ji} + \mathbf{u}_{i,i+1}\mathbf{g}_{i+1,i} \tag{4.75}$$

$$\mathbf{g}_{i-1,i} = \mathbf{u}_{i-1,j}\mathbf{g}_{ji} + \mathbf{u}_{i-1,i}\mathbf{g}_{ii} \tag{4.76}$$

$$\mathbf{g}_{i,i-1} = \mathbf{g}_{ij}\mathbf{l}_{j,i-1} + \mathbf{g}_{ii}\mathbf{l}_{i,i-1} \tag{4.77}$$

with the final diagonal block defined similar to Eq. (4.75) becoming

$$\mathbf{g}_{j+1,j+1} = \mathbf{d}_{i+1,i+1}^{-1} + \mathbf{u}_{j+1,j}\mathbf{g}_{j,j+1} + \mathbf{u}_{j+1,j+2}\mathbf{g}_{j+2,j+1}. \tag{4.78}$$

Since we have been given $\mathbf{g}_{jj}$ and $\mathbf{g}_{kk}$ from the BCR phase, and with formulas Eq. (4.68) and Eq. (4.73) for $\mathbf{g}_{k-1,k}$ and $\mathbf{g}_{k,k-1}$, we now have the full block tridiagonal portion of $\mathbf{G}$ between rows j and k, inclusive.

4.4.2 Algorithm

We now present the algorithms that perform the Schur reduction and production phases in the Hybrid method, along with the main algorithm for the Hybrid method that calls the Schur and BCR reduction and production phases correctly to determine $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ for a given $\mathbf{A}$.

4.4.2.1 Schur Reduction

The Schur reduction phase is handled by Alg. (4.14) REDUCESCHUR, which handles the three different cases of a process owning either an upper, lower, or middle section of $\mathbf{A}$.

For the case of a process owning the upper rows of $\mathbf{A}$, we enter the case on line 1. This section of the algorithm ensures that the process owning these upper rows commences a downwards elimination sweep, computing the Schur complement of the matrix defined in Eq. (4.40). As computation proceeds, we store the generated LU factors $\mathbf{l}_{i,i-1}$ and $\mathbf{u}_{i-1,i}$ on line 3 and line 4 for later use in reconstructing $\mathbf{G}$. The matrix $\mathbf{A}$ is updated on line 5. For the case of a process owning the bottom rows of $\mathbf{A}$, we execute the code handled by the case on line 6, whose behavior is mirrored by that of the case of a process owning the upper rows.

The case of a process owning some rows in the middle of $\mathbf{A}$ is handled by the case on line 11, where we are tasked with two eliminations per iteration. Thus a total of 4 LU factors are calculated on line 13, line 14, line 15, and line 16. Furthermore, as we are calculating a Schur complement which is a 2×2 block matrix (cf. Eq. (4.55) and Eq. (4.56)), we have 4 block updates per iteration, one for each block in the Schur complement. This is performed on line 17, line 18, line 19, and line 20.

In the case of a sequential case where $\mathcal{P} = 1$, and by looking at the conditionals for line 1, line 6 and line 11, we see that we execute the case for a process owning the lower rows of $\mathbf{A}$ given by line 6. For two processes, we execute the cases for line 1 and line 6, while the case for line 11 where a process might only own some middle rows in $\mathbf{A}$ is only executed when $\mathcal{P} \geq 3$.

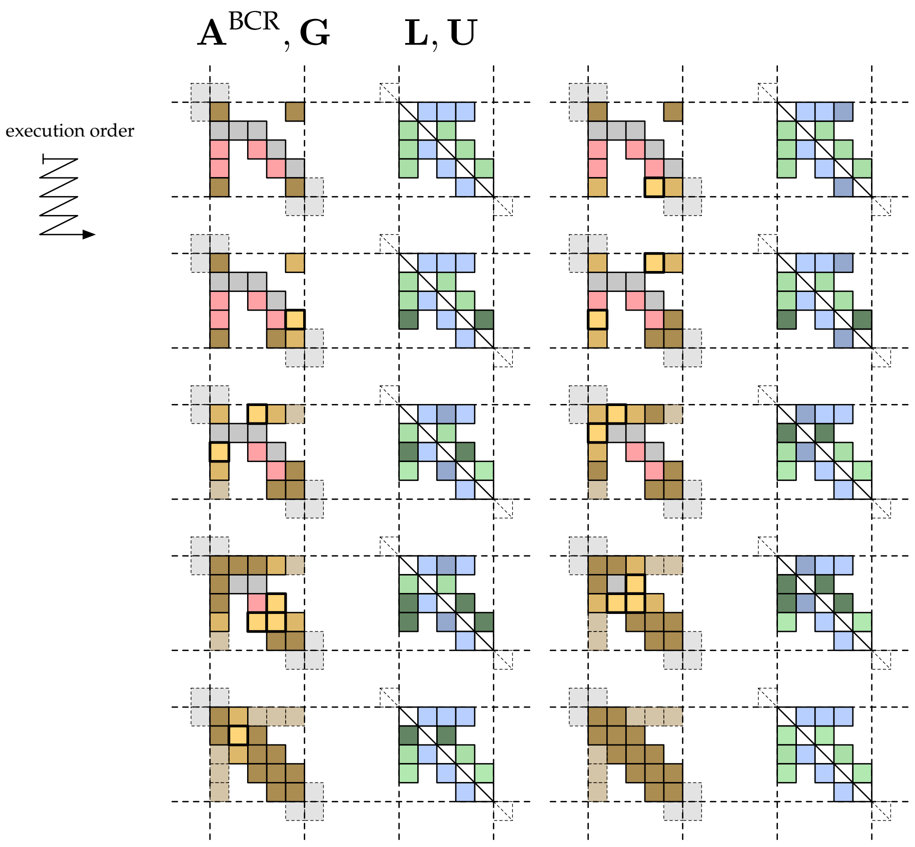


Figure 4.21: This figure shows the Schur production phase for a central process owning rows in $\mathbf{A}$. The process starts where we have been given the Green's function blocks in the corners $\mathbf{g}_{jj}$, $\mathbf{g}_{jk}$, $\mathbf{g}_{kj}$ and $\mathbf{g}_{kk}$, as defined by Eq. (4.63). As production proceeds, the bright yellow blocks indicate new computed elements of $\mathbf{G}$, while the inert blocks of $\mathbf{G}$ are a dark yellow, and involved blocks of $\mathbf{G}$ in the computation of new blocks are an intermediate yellow color. Inert blocks of the LU factors are a lighter color than the current, active blocks being used in the production of $\mathbf{G}$ blocks. We can finally see the production of the block tridiagonal portion of the inverse $\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{G}\right\}$, owned by the center process, at the bottom, where we have discarded off diagonal elements that were necessary for the production process.

Algorithm 4.14 REDUCESCHUR($\mathbf{A}$)

```
1:  if myPID $= 0$ and $\mathcal{P} > 1$ then                          corner eliminate downwards
2:      for $i = top + 1$ up to $bot$ do                          $\mathbf{d}_{top,top} = \mathbf{a}_{top,top}$
3:          $\mathbf{l}_{i,i-1} \leftarrow -\mathbf{a}_{i,i-1}\mathbf{d}_{i-1,i-1}^{-1}$          $1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
4:          $\mathbf{u}_{i-1,i} \leftarrow -\mathbf{a}_{i-1,i-1}^{-1}\mathbf{a}_{i-1,i}$          $1\mathsf{op}(\times)$
5:          $\mathbf{d}_{ii} \leftarrow \mathbf{a}_{ii} + \mathbf{l}_{i,i-1}\mathbf{a}_{i-1,i}$          $1\mathsf{op}(\times), 1\mathsf{op}(+)$
6:  if myPID $= \mathcal{P} - 1$ then                          corner eliminate upwards
7:      for $i = bot - 1$ down to $top$ do                          $\mathbf{d}_{bot,bot} = \mathbf{a}_{bot,bot}$
8:          $\mathbf{l}_{i,i+1} \leftarrow -\mathbf{a}_{i,i+1}\mathbf{d}_{i+1,i+1}^{-1}$          $1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
9:          $\mathbf{u}_{i+1,i} \leftarrow -\mathbf{a}_{i+1,i+1}^{-1}\mathbf{a}_{i+1,i}$          $1\mathsf{op}(\times)$
10:         $\mathbf{d}_{ii} \leftarrow \mathbf{a}_{ii} + \mathbf{l}_{i,i+1}\mathbf{a}_{i+1,i}$          $1\mathsf{op}(\times), 1\mathsf{op}(+)$
11: if myPID $\neq 0$ and myPID $\neq \mathcal{P} - 1$ and $\mathcal{P} > 1$ then          center eliminate down
12:     for $i = top + 2$ up to $bot$ do                          $\mathbf{d}_{top+1,top+1} = \mathbf{a}_{top+1,top+1}$
13:         $\mathbf{l}_{i,i-1} \leftarrow -\mathbf{a}_{i,i-1}\mathbf{d}_{i-1,i-1}^{-1}$          $1\mathsf{op}(\mathcal{LU}), 1\mathsf{op}(\times)$
14:         $\mathbf{l}_{top,i-1} \leftarrow -\mathbf{a}_{top,i-1}\mathbf{d}_{i-1,i-1}^{-1}$          $1\mathsf{op}(\times)$
15:         $\mathbf{u}_{i-1,i} \leftarrow -\mathbf{a}_{i-1,i-1}^{-1}\mathbf{a}_{i-1,i}$          $1\mathsf{op}(\times)$
16:         $\mathbf{u}_{i-1,top} \leftarrow -\mathbf{a}_{i-1,i-1}^{-1}\mathbf{a}_{i-1,top}$          $1\mathsf{op}(\times)$
17:         $\mathbf{d}_{ii} \leftarrow \mathbf{a}_{ii} + \mathbf{l}_{i,i-1}\mathbf{a}_{i-1,i}$          $1\mathsf{op}(\times), 1\mathsf{op}(+)$
18:         $\mathbf{a}_{top,top} \leftarrow \mathbf{a}_{top,top} + \mathbf{l}_{top,i-1}\mathbf{a}_{i-1,top}$          $1\mathsf{op}(\times), 1\mathsf{op}(+)$
19:         $\mathbf{a}_{i,top} \leftarrow \mathbf{l}_{i,i-1}\mathbf{a}_{i-1,top}$          $1\mathsf{op}(\times)$
20:         $\mathbf{a}_{top,i} \leftarrow \mathbf{l}_{top,i-1}\mathbf{a}_{i-1,i}$          $1\mathsf{op}(\times)$
21: return $\mathbf{A}, \mathbf{L}, \mathbf{U}$
```

4.4.2.2 Schur Production

When the results for the Schur reduction phase is obtained, it is passed off to a BCR method to solve the reduced system of equations. With these results, we then calculate the remaining desired blocks of the inverse using Alg. (4.15) PRODUCESCHUR.

This algorithm is also partitioned to handle the three different cases of a process owning some top rows of $\mathbf{A}$, bottom rows of $\mathbf{A}$, or some rows excluding the top or bottom ones, much as Alg. (4.14) REDUCESCHUR does. This is handled by the cases on line 1, line 6 and line 11, respectively.

For the case of a process owning some upper rows in $\mathbf{A}$, we know that a downwards sweeping corner reduction phase was performed, and we counter this with an upwards sweeping corner production phase, as provided by the recursion formulas Eqs. (4.51)–(4.53). We find these formulas implemented on line 3, line 4 and line 5. The derivation and formulas for case of a process owning lower rows in $\mathbf{A}$ is a mirror analogue to these.

The construction of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ becomes a little more complex for the case of a process owning a section of middle rows, however, the order of reconstruction mirrors the order of derivation of the equations used.

The formation of the first two blocks $\mathbf{g}_{k,k-1}$ and $\mathbf{g}_{k-1,k}$, as given by Eq. (4.68) and Eq. (4.73) respectively, is performed first on line 12 and line 13.

A for loop on line 14 generates a sequence of Green's function blocks corresponding to the top row of blocks in Eq. (4.67) and the associated back solved Green's function blocks using the calculated LU factors $\mathbf{u}$ (seen as placeholder $\circ$ symbols in Eq. (4.56)) obtained during the Schur reduction phase.

These off diagonal blocks can then be used in a second for loop on line 17 that takes care of generating most of the block tridiagonal of $\mathbf{G}$ belonging to the calling process. This is done via calculations on line 18, line 19 and line 20 that correspond to Eqs. (4.75)–(4.77).

A final diagonal block is calculated on line 21 that takes care of catching the only block missed by the for loop on line 17, corresponding to Eq. (4.78).

Finally, the algorithm that introduces the BCR method between the Schur reduction and production phases, and gives us a single procedure to call to determine $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ for some block tridiagonal matrix $\mathbf{A}$, is given in Alg. (4.16) INVERSEHYBRID.

Overall, the algorithm is characterized by two phases, the first being one of reduction of the full block tridiagonal system $\mathbf{A}$ of $n \times n$ blocks to one of a single block $\mathbf{a}_{kk}$, for which we can easily directly compute the corresponding inverse $\mathbf{g}_{kk}$. This is done in a single call on line 3.

Algorithm 4.15 ProduceSchur($\mathbf{A}, \mathbf{L}, \mathbf{U}, \mathbf{G}$)

```
1: if myPID $= 0$ and $\mathcal{P} > 1$ then                                    corner produce upwards
2:     for $i = bot$ down to $top + 1$ do
3:         $\mathbf{g}_{i,i-1} \leftarrow \mathbf{g}_{ii}\mathbf{l}_{i,i-1}$                                     1op(×)
4:         $\mathbf{g}_{i-1,i} \leftarrow \mathbf{u}_{i-1,i}\mathbf{g}_{ii}$                                     1op(×)
5:         $\mathbf{g}_{i-1,i-1} \leftarrow \mathbf{d}^{-1}_{i-1,i-1} + \mathbf{u}_{i-1,i}\mathbf{g}_{i,i-1}$              1op(×), 1op(+)
6: if myPID $= \mathcal{P} - 1$ then                                          corner produce downwards
7:     for $i = top$ up to $bot - 1$ do
8:         $\mathbf{g}_{i,i+1} \leftarrow \mathbf{g}_{ii}\mathbf{l}_{i,i+1}$                                     1op(×)
9:         $\mathbf{g}_{i+1,i} \leftarrow \mathbf{u}_{i+1,i}\mathbf{g}_{ii}$                                     1op(×)
10:        $\mathbf{g}_{i+1,i+1} \leftarrow \mathbf{d}^{-1}_{i+1,i+1} + \mathbf{u}_{i+1,i}\mathbf{g}_{i,i+1}$              1op(×), 1op(+)
11: if myPID $\neq 0$ and myPID $\neq \mathcal{P} - 1$ and $\mathcal{P} > 1$ then          center produce up
12:    $\mathbf{g}_{bot,bot-1} \leftarrow \mathbf{g}_{bot,top}\mathbf{l}_{top,bot-1} + \mathbf{g}_{bot,bot}\mathbf{l}_{bot,bot-1}$        2op(×), 1op(+)
13:    $\mathbf{g}_{bot-1,bot} \leftarrow \mathbf{u}_{bot-1,bot}\mathbf{g}_{bot,bot} + \mathbf{u}_{bot-1,top}\mathbf{g}_{top,bot}$        2op(×), 1op(+)
14:    for $i = bot - 1$ down to $top + 1$ do
15:        $\mathbf{g}_{top,i} \leftarrow \mathbf{g}_{top,top}\mathbf{l}_{top,i} + \mathbf{g}_{top,i+1}\mathbf{l}_{i+1,i}$              2op(×), 1op(+)
16:        $\mathbf{g}_{i,top} \leftarrow \mathbf{u}_{i,i+1}\mathbf{g}_{i+1,top} + \mathbf{u}_{i,top}\mathbf{g}_{top,top}$              2op(×), 1op(+)
17:    for $i = bot - 1$ down to $top + 2$ do
18:        $\mathbf{g}_{ii} \leftarrow \mathbf{d}^{-1}_{ii} + \mathbf{u}_{i,top}\mathbf{g}_{top,i} + \mathbf{u}_{i,i+1}\mathbf{g}_{i+1,i}$             2op(×), 2op(+)
19:        $\mathbf{g}_{i-1,i} \leftarrow \mathbf{u}_{i-1,top}\mathbf{g}_{top,i} + \mathbf{u}_{i-1,i}\mathbf{g}_{ii}$                2op(×), 1op(+)
20:        $\mathbf{g}_{i,i-1} \leftarrow \mathbf{g}_{i,top}\mathbf{l}_{top,i-1} + \mathbf{g}_{ii}\mathbf{l}_{i,i-1}$                2op(×), 1op(+)
21:    $\mathbf{g}_{top+1,top+1} \leftarrow \mathbf{d}^{-1}_{top+1,top+1} + \mathbf{u}_{top+1,top}\mathbf{g}_{top,top+1} + \mathbf{u}_{top+1,top+2}\mathbf{g}_{top+2,top+1}$
       2op(×), 2op(+)
22: return $\mathbf{G}$
```

The second main phase of INVERSEHYBRID is a production phase that takes the easily calculated single inverse block $\mathbf{g}_{kk}$ and generates the desired block tridiagonal inverse $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ using the LU factors saved during the reduction phase.

Algorithm 4.16 INVERSEHYBRID($\mathbf{A}, \mathbf{i}^{\text{BCR}}$)

```
Require: $\mathbf{A} \in \mathbb{B}^{n,n}$
Ensure: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} = \text{Trid}_{\mathbf{A}}\{\mathbf{A}^{-1}\} \in \mathbb{B}^{n,n}$
1: $\mathbf{A}^{\text{Schur}}, \mathbf{L}, \mathbf{U} \leftarrow$ REDUCESCHUR($\mathbf{A}$)
2: $\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U} \leftarrow$ REDUCEBCR($\mathbf{A}^{\text{Schur}}, \mathbf{L}, \mathbf{U}, \mathbf{i}^{\text{BCR}}$)
3: $\mathbf{g}_{kk}^{\text{BCR}} = (\mathbf{a}_{kk}^{\text{BCR}})^{-1}$                                   $1\text{op}(\mathcal{LU}), 1\text{op}(\times)$
4: $\mathbf{G}^{\text{BCR}} \leftarrow$ PRODUCEBCR($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}, \mathbf{i}^{\text{BCR}}$)
5: $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\} \subset \mathbf{G}^{\text{Schur}} \leftarrow$ PRODUCESCHUR($\mathbf{A}^{\text{BCR}}, \mathbf{L}, \mathbf{U}, \mathbf{G}^{\text{BCR}}$)
6: return $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$
```

4.4.3 Complexity

In analyzing the operation count for Alg. (4.16) INVERSEHYBRID, we can first notice that the core of it on line 2, line 3 and line 4 is precisely identical to that of Alg. (4.13) INVERSEBCR, although in this case, executed for a problem of size $(2\mathcal{P}-2) \times (2\mathcal{P}-2)$, rather than $n \times n$. Thus the earlier complexity analysis for BCR will serve us here. This leaves us with determining the complexity of the REDUCESCHUR and PRODUCESCHUR operations.

We will look at the case of a set of m rows in $\mathbf{A}$ being assigned to some process p, upon which REDUCESCHUR and PRODUCESCHUR is called. The operation count will differ for two distinct cases, namely on whether the m rows assigned to process p are either somewhere in the *middle* of $\mathbf{A}$, or if the rows are at the top or bottom of $\mathbf{A}$. In other words, that process p will be performing corner Schur reduction/productions or center Schur reduction/productions.

Starting with Alg. (4.14) REDUCESCHUR working on a section of $\mathbf{A}$ composed of m rows. In the case of a corner process, we see on line 2 and line 7 that a total of $m-1$ iterations are performed, each costing $1\text{op}(\mathcal{LU})$, $3\text{op}(\times)$ and $1\text{op}(+)$. With respect to a center process, we see the loop on line 12 executes a total of $m-2$ times, with each pass costing $1\text{op}(\mathcal{LU})$, $8\text{op}(\times)$ and $2\text{op}(+)$.

Next we look at Alg. (4.15) PRODUCESCHUR working on the same section of $\mathbf{A}$ with m rows. For a corner process, we see the loop on line 2 and line 7 execute a total of $m-1$ times, with each pass costing $3\text{op}(\times)$ and $1\text{op}(+)$. The case for the center process is slightly more complicated, but we see the loop on line 14 execute for a total of $m - 2$ times, each pass costing $4\mathsf{op}(\times)$ and $2\mathsf{op}(+)$. The loop on line 17 executes $m - 3$ times, with each pass costing $6\mathsf{op}(\times)$ and $4\mathsf{op}(+)$. Finally, single statements outside any loops cost the middle process $6\mathsf{op}(\times)$ and $4\mathsf{op}(+)$.

Complexity Analysis for Schur Reduction/Production

Calculation	LU–factorizations $\mathsf{op}(\mathcal{LU})$	Multiplications $\mathsf{op}(\times)$	Additions $\mathsf{op}(+)$
ReduceSchur (corner)	$m-1$	$3(m-1)$	$m-1$
ProduceSchur (corner)	0	$3(m-1)$	$m-1$
Total (corner)	$m-1$	$6(m-1)$	$2(m-1)$
ReduceSchur (center)	$m-2$	$8(m-2)$	$2(m-2)$
ProduceSchur (center)	0	$10(m-2)$	$6(m-2)$
Total (center)	$m-2$	$18(m-2)$	$8(m-2)$

Table 4.3: This table illustrates the amount of basic operations performed in calculating performing the Schur reduction/production procedures for a process owning m rows of $\mathbf{A}$. The second, third and fourth columns refer to the amount of basic matrix operations of LU–factorization, multiplication and addition involved in each algorithm. The term m is the total amount of diagonal blocks owned by the calling process, and may differ between processes.

We now generate the total operation count for performing the Schur reduction/production phases on a matrix $\mathbf{A}$ with n block rows, spread over a total of $\mathcal{P}$ processes, where each process owns $m_0, m_1, \ldots, m_{\mathcal{P}-1}$ rows. We will use Eq. (4.1) in deriving the total operation count.

In order to count the operations involved in total for the different operation types in Table 4.3, we recognize the fact that for corner processes we have a total of $\alpha(m-1)$ operations, and for center processes we have $\beta(m-2)$ operations in total, where α and β is some constant integer. This enables us to develop the following general formula for an operation count:

$$\begin{aligned}\mathsf{ops} &= \alpha(m_0 - 1) + \alpha(m_{\mathcal{P}-1} - 1) + \sum_{i=1}^{\mathcal{P}-2} \beta(m_i - 2) \\ &= \alpha(m_0 + m_{\mathcal{P}-1}) - 2\alpha - 2\beta(\mathcal{P} - 2) + \beta \sum_{i=1}^{\mathcal{P}-2} m_i \\ &= \beta n + (\alpha - \beta)(m_0 + m_{\mathcal{P}-1}) - 2\alpha - 2\beta(\mathcal{P} - 2) \qquad (4.79)\end{aligned}$$

where α is the factor multiplied on $m - 1$ for the number of operations for corner processes, and β is the factor multiplied on $m - 2$ for the middle processes. The formula takes its origin in the fact that the corner processes, with m_0 and $m_{\mathcal{P}-1}$ rows each, has a different operation count than the middle processes, and we seek to generate an expression involving the total number of block rows n using Eq. (4.1). Using Eq. (4.79), we can now generate the total operation count for the Schur reduction/production phases in the Hybrid method, and they are given below.

$$\begin{aligned} \mathsf{op}(\mathcal{LU}) &= n - 2\mathcal{P} + 2 && (4.80) \\ \mathsf{op}(\times) &= 18n - 12(m_0 + m_{\mathcal{P}-1}) - 36\mathcal{P} + 60 && (4.81) \\ \mathsf{op}(+) &= 8n - 6(m_0 + m_{\mathcal{P}-1}) - 16\mathcal{P} + 28 && (4.82) \end{aligned}$$

If we assume we are in the case Hybrid was designed for, namely that $n \gg \mathcal{P}$, then

$$\begin{aligned} \mathsf{op}(\mathcal{LU}) &\rightarrow n && (4.83) \\ \mathsf{op}(\times) &\rightarrow 18n && (4.84) \\ \mathsf{op}(+) &\rightarrow 8n, && (4.85) \end{aligned}$$

while assuming that $m_0 \ll n$ and $m_{\mathcal{P}-1} \ll n$.

With regards to the BCR phase, after the Schur reduction phase over $\mathcal{P}$ processes, then we have reduced $\mathbf{A}$ to a $(2\mathcal{P} - 2) \times (2\mathcal{P} - 2)$ system. Thus we can use Table 4.2, substituting n with $2\mathcal{P} - 2$. This leads to a total of $\mathcal{O}(2\mathcal{P})\mathsf{op}(\mathcal{LU})$, $\mathcal{O}(32\mathcal{P})\mathsf{op}(\times)$ and $\mathcal{O}(16\mathcal{P})\mathsf{op}(+)$ operations. Ultimately, if $n \gg \mathcal{P}$, then the operations contributed by BCR will become negligible in the overall behavior of the Hybrid method, and be dominated by the Schur phase costs. The BCR phase for the Hybrid method has an elimination tree of depth h, where

$$h = \lfloor \log_2(2\mathcal{P} - 2) \rfloor = 1 + \lfloor \log_2(\mathcal{P} - 1) \rfloor \qquad (4.86)$$

as given by Eq. (4.37). We can see the results of the complexity analysis for the Hybrid method in Table 4.4.

Complexity Analysis for the Hybrid Method

Calc.	LU–facts. $\mathsf{op}(\mathcal{LU})$	Multiplications $\mathsf{op}(\times)$	Additions $\mathsf{op}(+)$
Schur	$n - 2\mathcal{P} + 2$	$18n - 12(m_0 + m_{\mathcal{P}-1}) - 36\mathcal{P} + 60$	$8n - 6(m_0 + m_{\mathcal{P}-1}) - 16\mathcal{P} + 28$
BCR	$2\mathcal{P} - 2$	$32\mathcal{P} - 20h - 47$	$16\mathcal{P} - 12h - 24$
Total	n	$\mathcal{O}(18n)$	$\mathcal{O}(8n)$

Table 4.4: This table illustrates the amount of basic operations performed in calculating performing calculating the block tridiagonal inverse of $\mathbf{A}$ using the Hybrid method. The second, third and fourth columns refer to the amount of basic matrix operations of LU–factorization, multiplication and addition involved in each algorithm.

Chapter 5

Benchmarking

There are two possible outcomes: if the result confirms the hypothesis, then you've made a measurement. If the result is contrary to the hypothesis, then you've made a discovery.

Enrico Fermi

5.1 Benchmarking Serial Algorithms

The serial algorithms presented in this thesis have not been explicitly benchmarked. However, for the case of transmission calculations, a complexity analysis and execution time measurements have been carried out [1] on a symmetric multiprocessor[1], a Beowulf cluster[2] as well as a personal laptop[3].

It was shown that the Sweep based algorithm SWEEPINVERSETRI was a superior choice over standard Gaussian elimination for determining the block tridiagonal matrix $\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{G}\right\}$, but beyond this a series of benchmarks were carried out on a number of example block matrices $\mathbf{A}$ arising from relatively recent literature. The examples, however, barely exceed $n = 10$ block rows for $\mathbf{A}$, if at all, and are not suited for parallel benchmarking for processes $\mathcal{P} > 2$.

An issue to note is that the method employed in [1] was one consisting only of a reduction phase, since only a single block of the inverse $\mathbf{g}_{ij}$ is needed to

[1]Sun Microsystems SunFire E25K with 72 UltraSPARC-IV+ dual–core chips @ 1.35 GHz & 384 GB of RAM.

[2]Niflheim cluster at the Technical University of Denmark composed of 434 IBM ThinkCentre S50 (3.2 GHz P4, 2 GB) and 479 HP/Compaq EVO d510 (2.26 GHz P4, 1 GB) machines.

[3]Apple PowerBook with an IBM/Motorola PowerPC G4 @ 1.6 GHz & 2GB of RAM and an Apple MacBook with an Intel Core 2 Duo @ 2.2 GHz & 2GB of RAM.

calculate transmission. This feature renders BCR and the Hybrid method as possible candidates for this task also, and with greater possibility of speedup than SWEEPINVERSEPARALLEL.

The size of the examples from the literature in [1] goes to show how an efficient, scalable parallel algorithm for the treatment of block tridiagonal matrices may help model far larger systems, and thus obtain results that are more comparable with the real world.

5.2 Benchmarking Parallel Algorithms

5.2.1 Load Balancing

So far, we have not touched on the subject of load balancing. As we can more or less arbitrarily subdivide the matrix $\mathbf{A}$'s rows among several processes, we can use our earlier complexity analysis in determining a possible strategy.

It must be said, however, that the complexity analysis has been worked out assuming equal sized blocks of dimension d throughout $\mathbf{A}$. This is not usually the case, and very much dependent on the geometry of the two–probe system being modeled. Some two probe systems, though, such as elongated carbon nanotube systems or nanowire systems, may exhibit equal sized blocks throughout, and would be well suited for strategies developed here.

5.2.1.1 Parallel Sweeps

With respect to sweeps, the load balancing affects the Gaussian elimination portion called by Alg. (4.1) GAUSSELIMINATEPARALLEL differently than in the construction phase of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$, managed by Alg. (4.2) DIAGONALSPARALLEL and Alg. (4.3) OFFDIAGONALSPARALLEL.

This is due to the highly sequential dependency of GAUSSELIMINATEPARALLEL in passing the upwards and downwards sweeps on $\mathbf{A}$ along neighbor processes. A downward sweep is essentially a sequential process, despite being run on multiple machines, since a process can only begin its portion of the sweep when the sweep has passed through all processes above it. Likewise we have a similar situation for the upwards sweep. Thus we can essentially arbitrarily change the row distribution of $\mathbf{A}$ among processes, without affecting the Gaussian elimination phase running time of SWEEPINVERSEPARALLEL.

Furthermore, we can be reasonably assured that this stage will terminate at more or less the same time for all processes, allowing for the fact that the upwards/downwards sweeps will meet at a middle process p_i in $\mathbf{A}$, where one of the sweeps will have to wait on the other to relinquish p_i in order to be able to progress.

However, for the reconstruction of $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ the story is different regarding load balancing. Here, when started, each process will already have enough information in order to finish execution without communication. In other words, this stage of SWEEPINVERSEPARALLEL is embarrassingly parallel, and thus for all processes to terminate more or less at the same time, they will need to be assigned a number of rows that ensures a more or less equal workload.

Assuming equal block sizes throughout $\mathbf{G}$, the optimal workload will result in the same number of rows assigned to each process. This has to take into account the fact that the corner processes perform a single $\mathsf{op}(\times)$ less than a middle process, as seen in Alg. (4.3) OFFDIAGONALSPARALLEL, but this quickly becomes negligible as the ration of n to $\mathcal{P}$ increases.

5.2.1.2 Block Cyclic Reduction

Block cyclic reduction has the behavior of relatively finely grained parallelism, in the sense that each row update from an odd row to an even row, can be done in parallel, on every level of the elimination tree.

This leads for the ideal situation to be that we have as many processes as we have block rows $n = \mathcal{P}$. This, will quickly lead to idle processes as BCR moves up the elimination tree, losing half the available computing power at each level. Although perhaps squandering resources, this is expected to be the quickest execution profile for BCR. With respect to memory, this would lead to the finest subdivision of block rows among processes, affording the possibility of treating larger problems, i.e. larger block dimensions, though at the cost of having to provide more hardware.

However, $\mathcal{P} = n$ is not the usual case for the sort of situation we envision our algorithms to run under, as we expect the number of block rows n to be much larger than the number of processes available for computation.

This will lead to fewer processes being "lost" moving up each level in the reduction phase of BCR. However, BCR does have a great deal of redundancy[1], and we would lose efficiency as each process, having multiple rows m assigned to it, performs a large number of cyclic reduction steps largely in serial, before having climbed up far enough in the elimination tree for message passing to start happening. If this is the case, then we will see that it will be better to employ a uniform[2] distribution of rows among processes, such that all processes

[1]The redundancy is approximately 2.7 [32].

[2]This is assuming constant block dimension d throughout $\mathbf{A}$.

will reach the message passing levels of the elimination tree at more or less the same time. This will avoid the case of a node having a child belonging to a lightly loaded process eliminate far quicker than the other child belonging to an overburdened process. Similarly, the under–burdened processes would terminate their jobs sooner than other processes or have to wait on overburdened processes to receive data before being able to continue.

Ultimately, the desire to preserve the parallel capability of BCR without suffering too great a deal when processes own too many or an unequal amount of rows each is the main inspirational factor in the development of the Hybrid method.

5.2.1.3 Hybrid Method

For the Hybrid method, we assume the limit where the BCR phase of operations become negligible, and thus we can limit ourselves to considering the Schur phase costs. As seen in the complexity analysis for the Schur reduction/production phases, a different operation count exists for a process belonging either to a corner or a center process, where the significant difference lies in the fact that while corner processes execute a total of $\mathcal{O}(6m)$ matrix multiplications, the center proccesses will execute three times more such operations, on the order of $\mathcal{O}(18m)$.

In order to approximate the number of flops executed by a corner or a center process, we can take from [29], the fact that matrix multiplication[1] between two blocks of dimension d requires on the order of $\mathcal{O}(2d^3)$ flops, while LU factorization of a block of dimension d requires $\mathcal{O}(\frac{2}{3}d^3)$. We can then use the complexities as tabulated earlier for InverseHybrid, and for the corner process we obtain

$$\begin{aligned} 1\mathsf{op}(\mathcal{LU}) + 6\mathsf{op}(\times) &= 1\mathcal{O}\left(\frac{2}{3}d^3\right) + 6\mathcal{O}\left(2d^3\right) \\ &= \left(\frac{2}{3} + 12\right)\mathcal{O}(d^3) \\ &= \left(\frac{38}{3}\right)\mathcal{O}(d^3) \end{aligned} \tag{5.1}$$

[1]As done by BLAS.

while for the center process we have

$$\begin{aligned}
1\mathsf{op}(\mathcal{LU}) + 18\mathsf{op}(\times) &= 1\mathcal{O}\left(\frac{2}{3}d^3\right) + 18\mathcal{O}\left(2d^3\right) \\
&= \left(\frac{2}{3} + 36\right)\mathcal{O}(d^3) \\
&= \left(\frac{110}{3}\right)\mathcal{O}(d^3) \qquad (5.2)
\end{aligned}$$

and their ratio leads to

$$\alpha = \frac{110}{38} \approx 2.895, \qquad (5.3)$$

which tells us that in order for the corner processes to be equally burdened as a center process, they must have $2.895m$ rows assigned versus only m rows assigned for center processes. This leads us to a basic load balancing for the Hybrid method, although with the caveat that we have assumed equal block dimension throughout $\mathbf{A}$.

In the serial case of $\mathcal{P} = 1$ and even for $\mathcal{P} = 2$, the Hybrid method only uses corner Schur reduction/production operations, and the center variant only comes into play for $\mathcal{P} \geq 3$. We can then calculate the theoretical speedup s for $\mathcal{P} \geq 3$ processes by taking the ratio of the total cost if every block did corner Schur productions, to the cost of $\mathcal{P} - 2$ processes performing center Schur reduction/productions and only 2 processes performing corner Schur reduction/productions[1]. Thus the parallel efficiency becomes

$$\begin{aligned}
s &= \frac{\mathcal{P}c_{\text{corner}}}{(\mathcal{P}-2)c_{\text{center}} + 2c_{\text{corner}}} \\
&= \frac{\mathcal{P}}{(\mathcal{P}-2)\frac{c_{\text{center}}}{c_{\text{corner}}} + 2} \\
&= \frac{\mathcal{P}}{\alpha(\mathcal{P}-2) + 2}, \qquad (5.4)
\end{aligned}$$

where α is the ratio of cost between a Schur reduction/production of a center process c_{center} and the cost of a Schur reduction/production for a corner process c_{corner}, for the same number of owned block rows, as calculated in Eq. (5.3).

5.2.2 Implementation

Unfortunately, the pseudocode as presented in this thesis has not been implemented perfectly, and the operation counts differ compared to the complexity

[1] This is the implemented case for the Hybrid method

analysis in this thesis. This is due to improved readability of the code and quick implementation being prioritized at the cost of efficiency. Thus the operation count for the implemented code is the following for a corner process

$$\begin{aligned}
1\mathsf{op}(\mathcal{LU}) + 7\mathsf{op}(\times) &= \mathcal{O}\left(\frac{2}{3}d^3\right) + 7\mathcal{O}\left(2d^3\right) \\
&= \left(\frac{2}{3} + 14\right)\mathcal{O}(d^3) \\
&= \left(\frac{44}{3}\right)\mathcal{O}(d^3) \qquad (5.5)
\end{aligned}$$

while for the center process we have

$$\begin{aligned}
1\mathsf{op}(\mathcal{LU}) + 19\mathsf{op}(\times) &= \mathcal{O}\left(\frac{2}{3}d^3\right) + 19\mathcal{O}\left(2d^3\right) \\
&= \left(\frac{2}{3} + 38\right)\mathcal{O}(d^3) \\
&= \left(\frac{116}{3}\right)\mathcal{O}(d^3) \qquad (5.6)
\end{aligned}$$

and their ratio leads to

$$\alpha = \frac{116}{44} \approx 2.636, \qquad (5.7)$$

which is less of a difference than present for the pseudocode, but both processes now execute with higher prefactors compared to before in that the corner process is now about 16% costlier, and the center process is 5.5% costlier. The methods are qualitatively still linear in the execution time, with respect to the number of diagonal blocks n in $\mathbf{A}$, however.

5.2.3 Results

A set of general complex block tridiagonal $\mathbf{A}$ matrices were generated in the same manner as presented in [1] and inverted using the parallel methods described in the previous chapter, for 4 different parameters.

The first parameter we vary is the block dimension for every block in $\mathbf{A}$. This we assign up to 3 values to: $d = \{128, 256, 512\}$, and it is constant throughout $\mathbf{A}$, i.e. all blocks in $\mathbf{A}$ are of the same size. Next, we vary the block–dimension of $\mathbf{A}$, i.e. how many diagonal blocks it has. This we assign up to 4 values: $n = \{128, 256, 512, 1024\}$. The third parameter we vary is how many processes we have available to help determine the inverse. This varies as

$\mathcal{P} = \{1, 2, 4, 8, 16, 32, 64\}$. Finally, we vary the load balancing by how we choose to distribute our n available blocks over the $\mathcal{P}$ available processes, which we can represent by the calculated value $\alpha = \{1, 2, 2.636, 3\}$.

If $\alpha = 1$, then a corner block is predicted to cost as much as a center block to work with, and we would choose a uniform distribution of identical block sizes over all of $\mathbf{A}$. One of the available choices is $\alpha = 2$, where a center process is assumed to do double the work a corner process does, for an equal assignment of m rows. Thus in order to load balance, the corner processes are assigned twice the amount of rows that the center processes are allocated. In the theoretical case, we would choose $\alpha \approx 2.895$ and in the implemented case we would choose $\alpha \approx 2.636$. However, as we will see, even a rough estimate of $\alpha = 2$ will yield a strong improvement in speedup behavior over $\alpha = 1$. Ideally, corner processes should be assigned αm rows, as center processes are allocated m rows each, where α is the value attained via the complexity analysis carried out earlier. This has been done, however, under the assumption that the dimension of blocks in $\mathbf{A}$ is identical throughout.

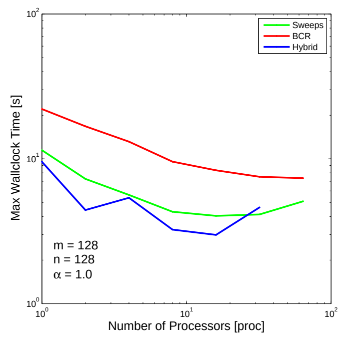
5.2.3.1 Load Distribution with $\alpha = 1$

For the uniform distribution of elements, the figures for block dimensions $d = \{128, 256, 512\}$ are presented in Fig. 5.1, Fig. 5.2 and Fig. 5.3, respectively. Each figure is arranged a in a 4×2 pattern, accounting for the 4 possible choices of the number of diagonal blocks in $\mathbf{A}$, namely $n = \{128, 256, 512, 1024\}$ and wether we measure walltime or speedup.

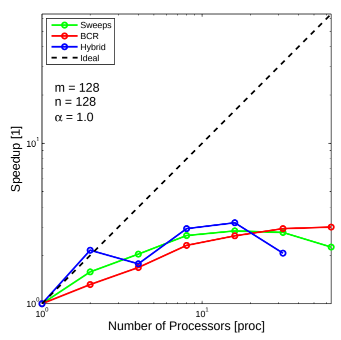
The walltime measurements are carried out by measuring the time it takes for all the participating processes to complete calculating $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ for a given algorithm. Speedup measurements are then made on the basis of these walltime measurements.

5.2.3.2 Load Distribution with $\alpha = 2$

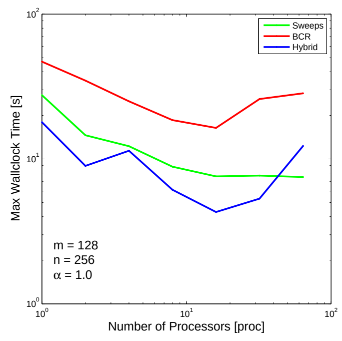
We have also tabulated the walltime and speedup results for an assumption that $\alpha = 2$, where $\mathbf{A}$ has been distributed such that corner processes are assigned $2m$ rows, while center processes are allocated m rows. For this basic distribution of rows, the figures for $m = \{128, 256, 512\}$ are presented in Fig. 5.4, Fig. 5.5 and Fig. 5.6, respectively. Each figure is arranged in the same 4×2 pattern, accounting for the 4 possible choices $n = \{128, 256, 512, 1024\}$ and wether we illustrate walltime or speedup results.



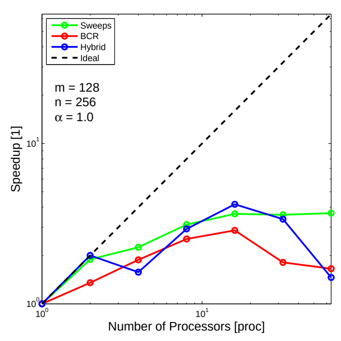
(a) Walltime for $n = 128$



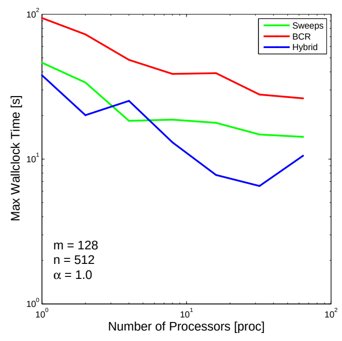
(b) Speedup for $n = 128$



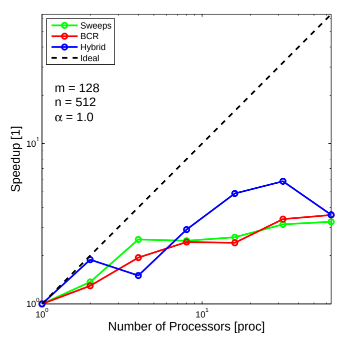
(c) Walltime for $n = 256$



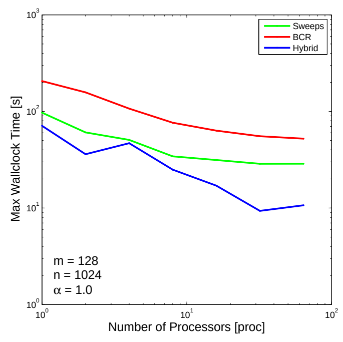
(d) Speedup for $n = 256$



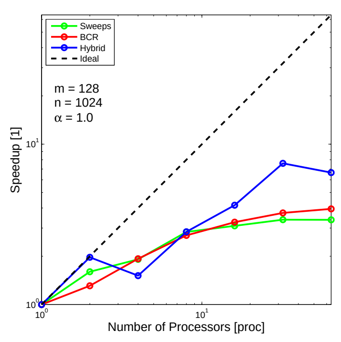
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

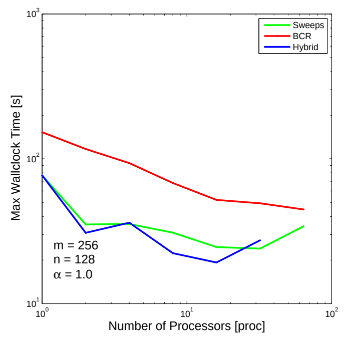


(g) Walltime for $n = 1024$

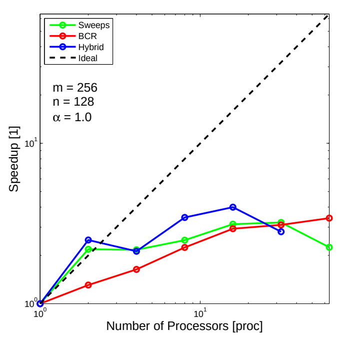


(h) Speedup for $n = 1024$

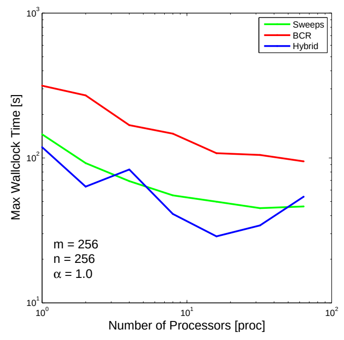
Figure 5.1: Timing figures for $\alpha = 1$ and block dimension $d = 128$.



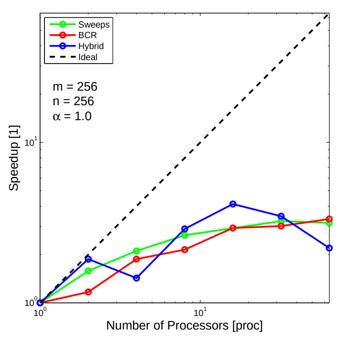
(a) Walltime for $n = 128$



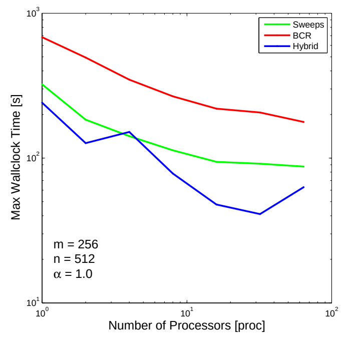
(b) Speedup for $n = 128$



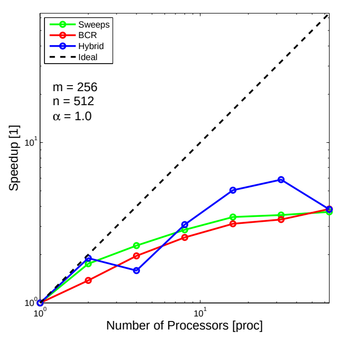
(c) Walltime for $n = 256$



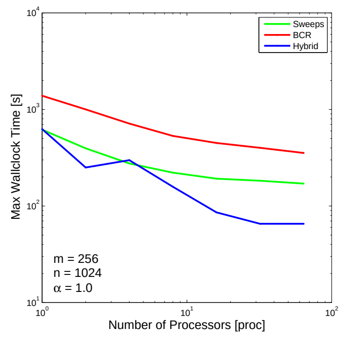
(d) Speedup for $n = 256$



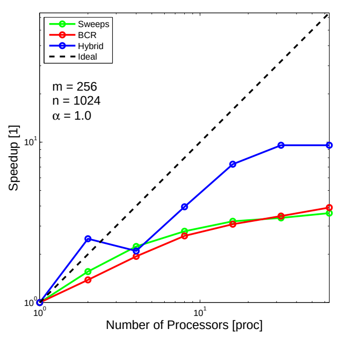
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

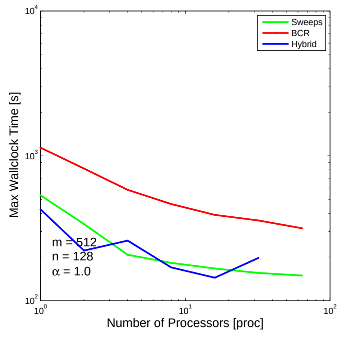


(g) Walltime for $n = 1024$

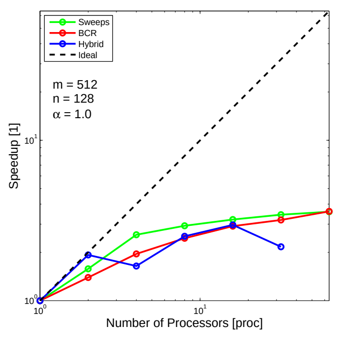


(h) Speedup for $n = 1024$

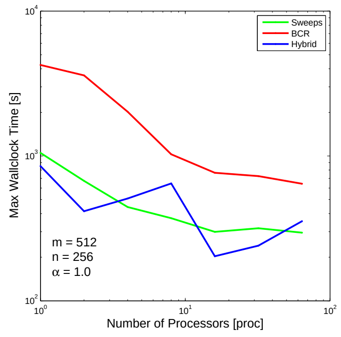
Figure 5.2: Timing figures for $\alpha = 1$ and block dimension $d = 256$.



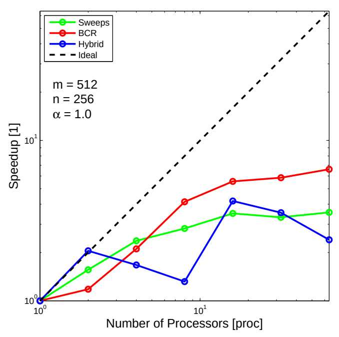
(a) Walltime for $n = 128$



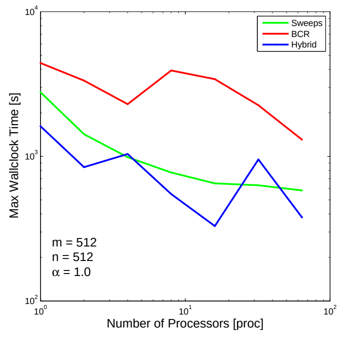
(b) Speedup for $n = 128$



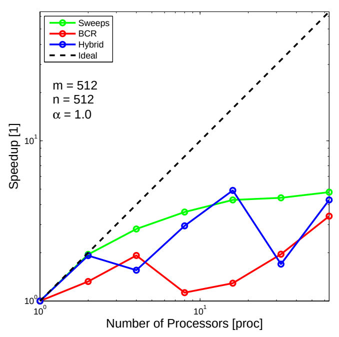
(c) Walltime for $n = 256$



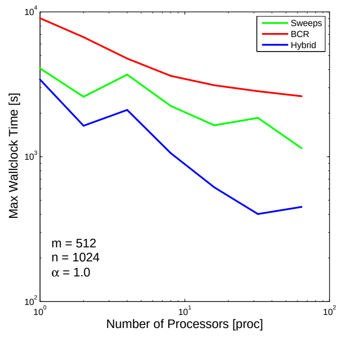
(d) Speedup for $n = 256$



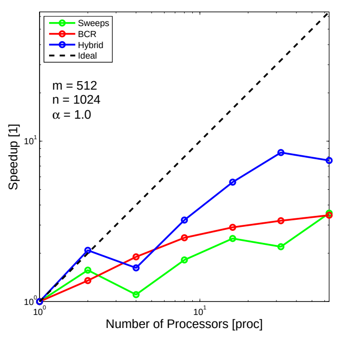
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

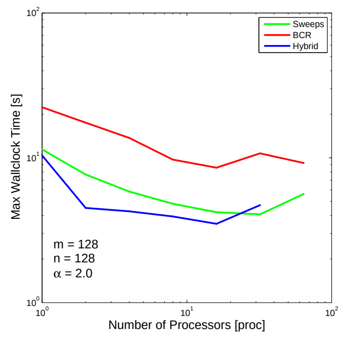


(g) Walltime for $n = 1024$

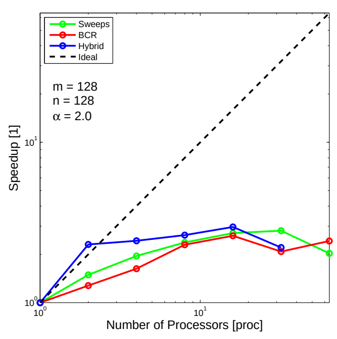


(h) Speedup for $n = 1024$

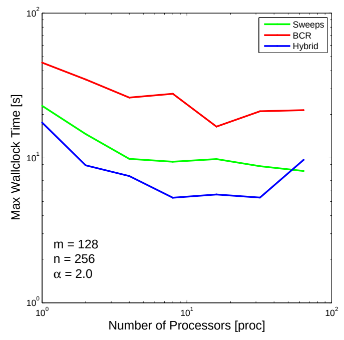
Figure 5.3: Timing figures for $\alpha = 1$ and block dimension $d = 512$.



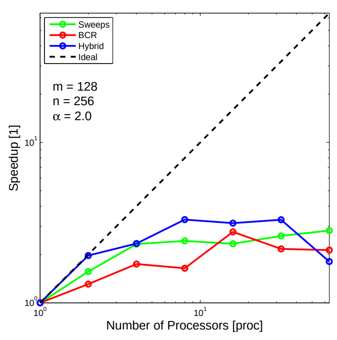
(a) Walltime for $n = 128$



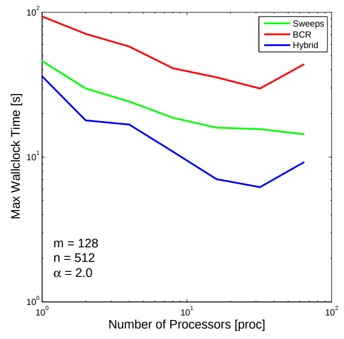
(b) Speedup for $n = 128$



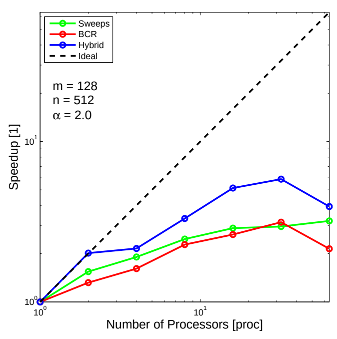
(c) Walltime for $n = 256$



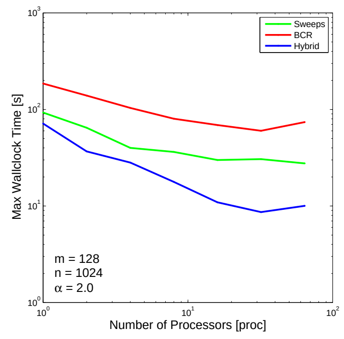
(d) Speedup for $n = 256$



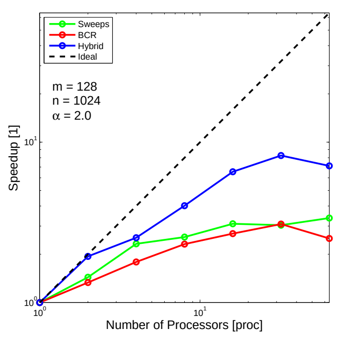
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

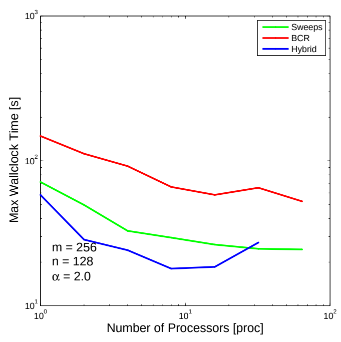


(g) Walltime for $n = 1024$

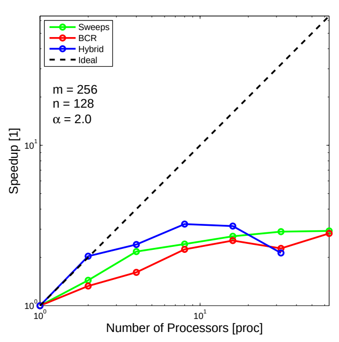


(h) Speedup for $n = 1024$

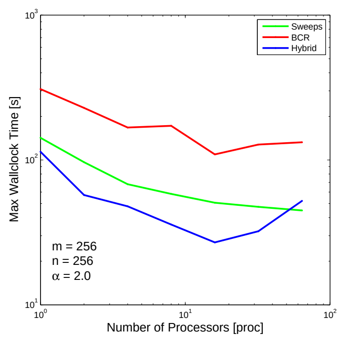
Figure 5.4: Timing figures for $\alpha = 2$ and block dimension $d = 128$.



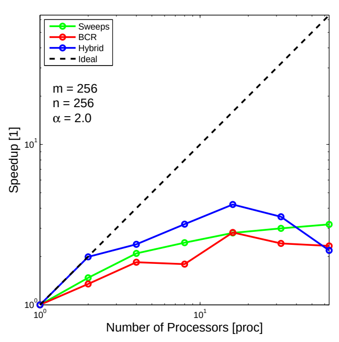
(a) Walltime for $n = 128$



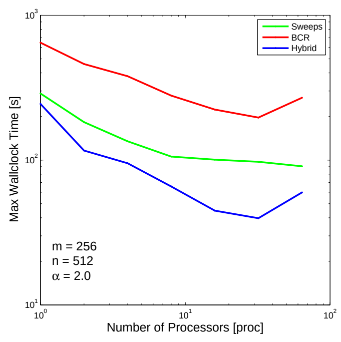
(b) Speedup for $n = 128$



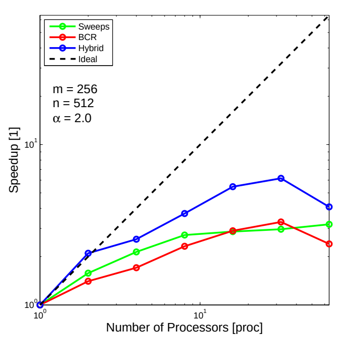
(c) Walltime for $n = 256$



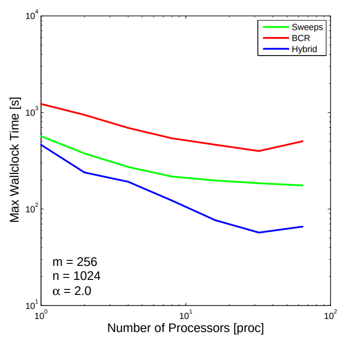
(d) Speedup for $n = 256$



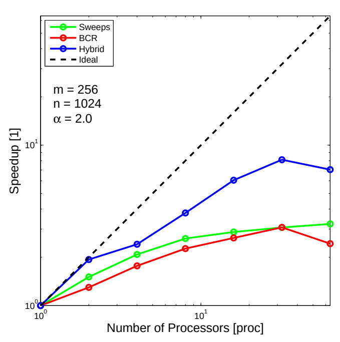
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

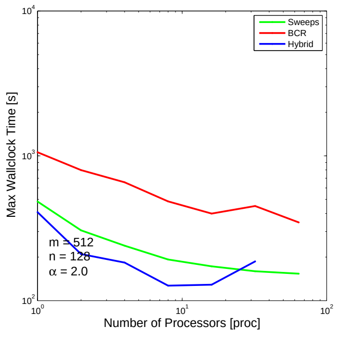


(g) Walltime for $n = 1024$

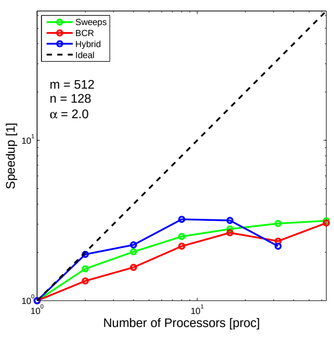


(h) Speedup for $n = 1024$

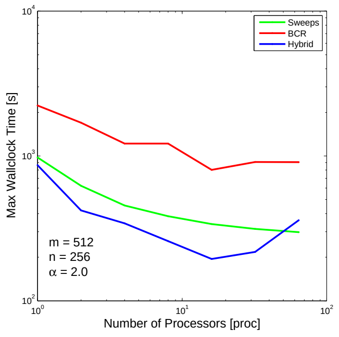
Figure 5.5: Timing figures for $\alpha = 2$ and block dimension $d = 256$.



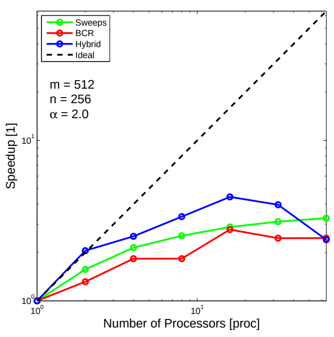
(a) Walltime for $n = 128$



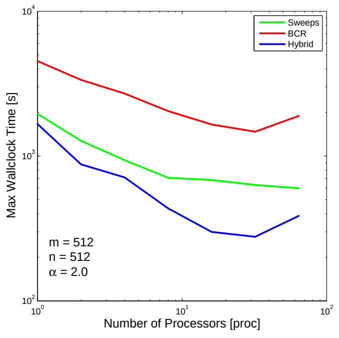
(b) Speedup for $n = 128$



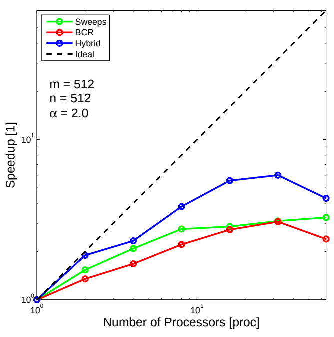
(c) Walltime for $n = 256$



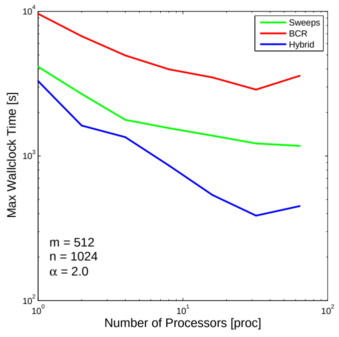
(d) Speedup for $n = 256$



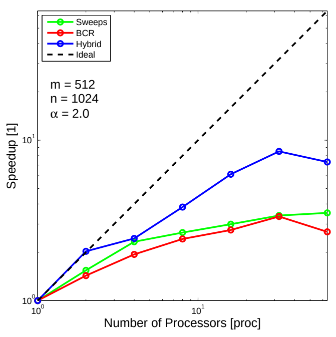
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$



(g) Walltime for $n = 1024$



(h) Speedup for $n = 1024$

Figure 5.6: Timing figures for $\alpha = 2$ and block dimension $d = 512$.

5.2.3.3 Load Distribution with $\alpha = 2.636$

We here tabulate the walltime and speedup results for an assumption that $\alpha = 2.636$, which should be the case as our complexity analysis for our implemented method has shown. Thus $\mathbf{A}$ has been distributed such that corner processes are assigned approximately $2.636m$ rows, while center processes are allocated m rows. As we can only distribute an integer value of rows among processes, this distribution is performed such that we approximate $\alpha = 2.636$ as much as possible.

For this distribution of rows, the figures for $m = \{128, 256, 512\}$ are presented in Fig. 5.7, Fig. 5.8 and Fig. 5.8, respectively. Each figure is arranged in the now familiar 4×2 pattern.

5.2.3.4 Load Distribution with $\alpha = 3$

Finally, the speedup results for an assumption that $\alpha = 3$ are given here, where $\mathbf{A}$ has been distributed such that corner processes are assigned $3m$ rows, while center processes are allocated m rows. For this distribution of rows, the figures for $m = \{128, 256, 512\}$ are presented in Fig. 5.10, Fig. 5.11 and Fig. 5.12, respectively. Each figure is arranged in the familiar 4×2 pattern.

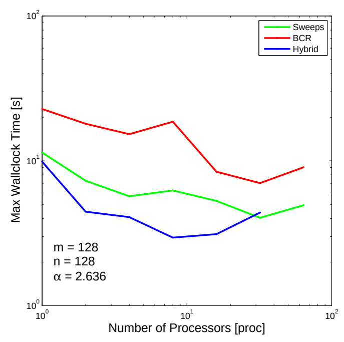
5.2.4 Remarks

Overall, the figures do well in qualitatively representing the behavior of the Sweeps, BCR and Hybrid algorithms. Some figures do exhibit an occasional spurious result, and this is due to using measurements from a single run, rather than over an ensemble of runs. Given more available computing time, these averaging runs could be carried out, and the figures attained would be less prone to such artifacts.

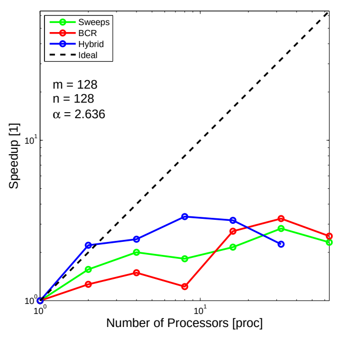
Notably, in all generated numerical cases and examples from the literature [1], inversions were stable, with relative norm errors on the block tridiagonal $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ being on the order of about 10^{-10} or less. This lends support to the notion that physical systems of interest will generally yield well–behaved matrices prior to and during execution of the presented algorithms.

5.2.4.1 Sweeps

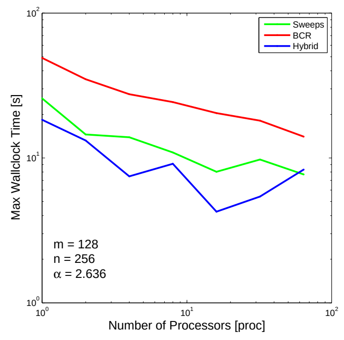
Unfortunately, the Sweeps method implemented has generally subpar behavior for $\mathcal{P} = 2$ processes compared to what is expected. Ideally, we expect near perfect speedup from one to two processes, since the Gaussian elimination



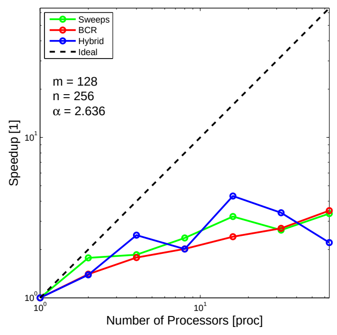
(a) Walltime for $n = 128$



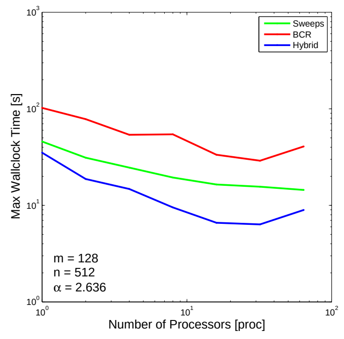
(b) Speedup for $n = 128$



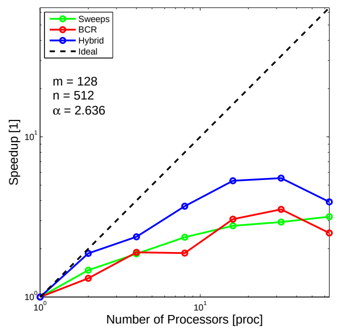
(c) Walltime for $n = 256$



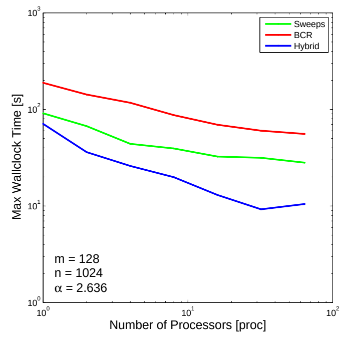
(d) Speedup for $n = 256$



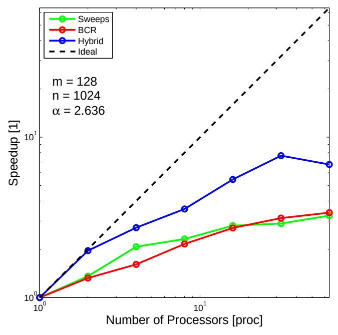
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

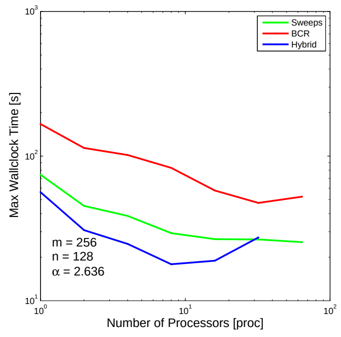


(g) Walltime for $n = 1024$

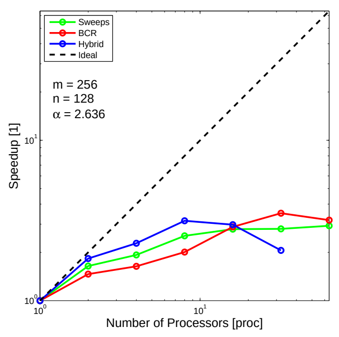


(h) Speedup for $n = 1024$

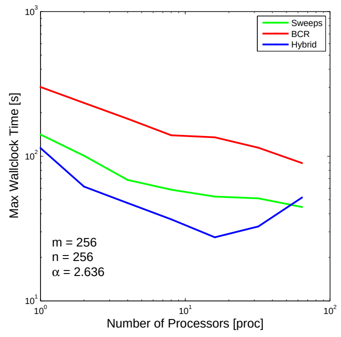
Figure 5.7: Timing figures for $\alpha = 2.636$ and block dimension $d = 128$.



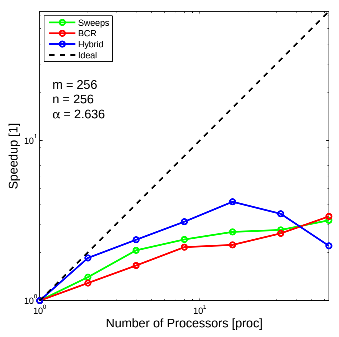
(a) Walltime for $n = 128$



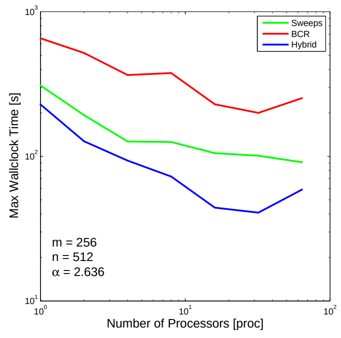
(b) Speedup for $n = 128$



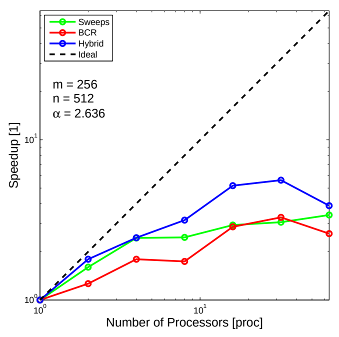
(c) Walltime for $n = 256$



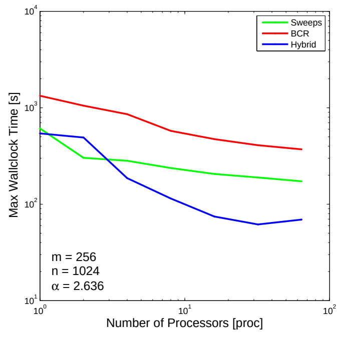
(d) Speedup for $n = 256$



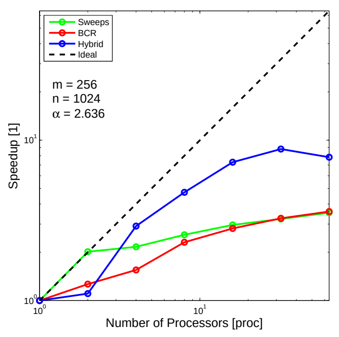
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

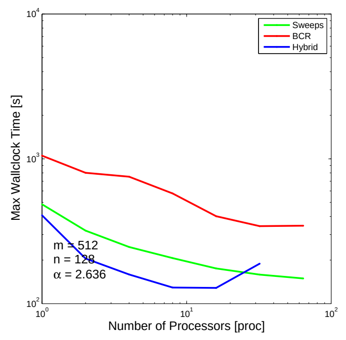


(g) Walltime for $n = 1024$

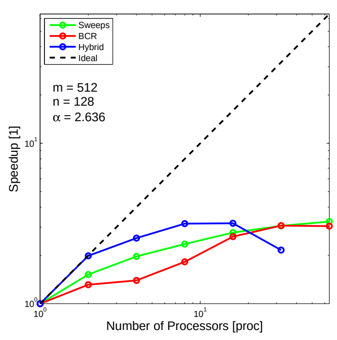


(h) Speedup for $n = 1024$

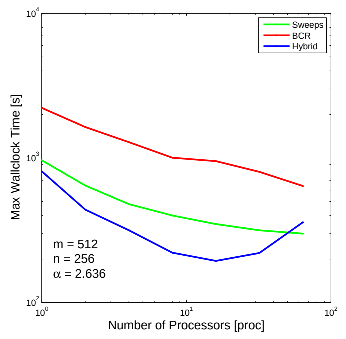
Figure 5.8: Timing figures for $\alpha = 2.636$ and block dimension $d = 256$.



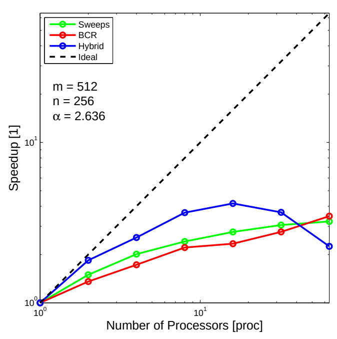
(a) Walltime for $n = 128$



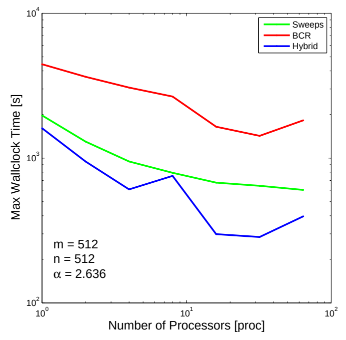
(b) Speedup for $n = 128$



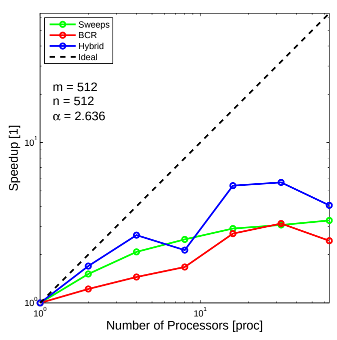
(c) Walltime for $n = 256$



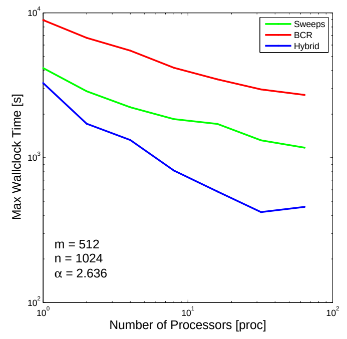
(d) Speedup for $n = 256$



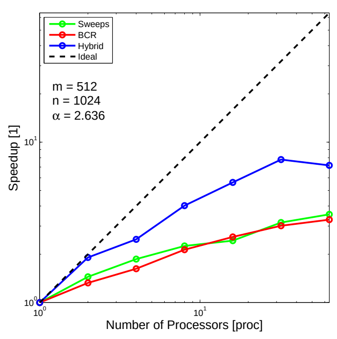
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

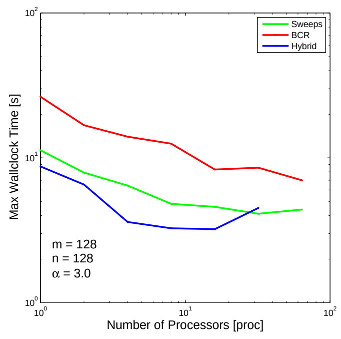


(g) Walltime for $n = 1024$

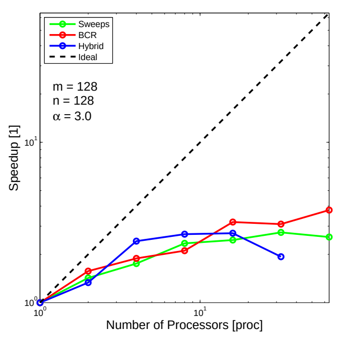


(h) Speedup for $n = 1024$

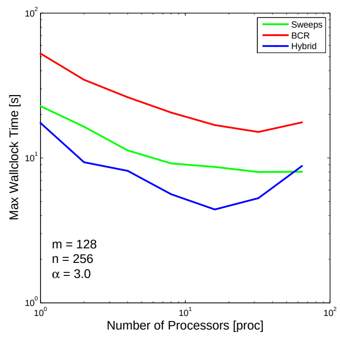
Figure 5.9: Timing figures for $\alpha = 2.636$ and block dimension $d = 512$.



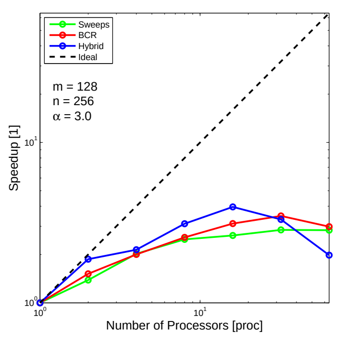
(a) Walltime for $n = 128$



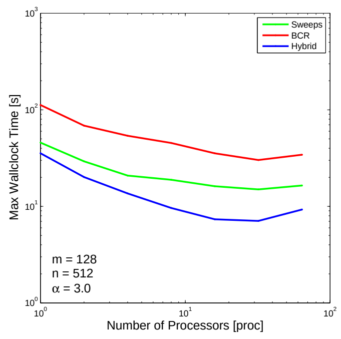
(b) Speedup for $n = 128$



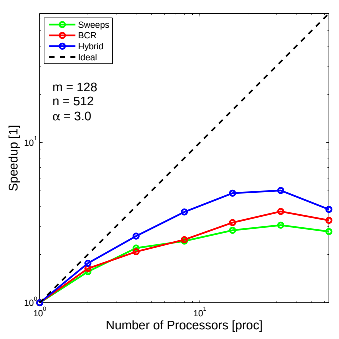
(c) Walltime for $n = 256$



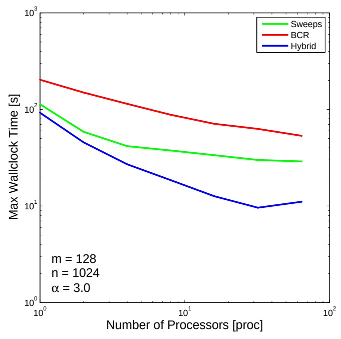
(d) Speedup for $n = 256$



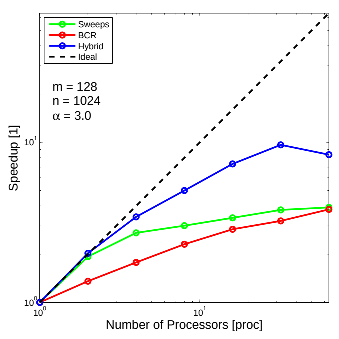
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

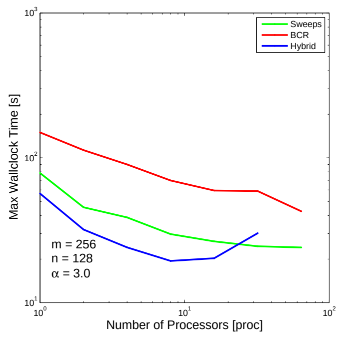


(g) Walltime for $n = 1024$

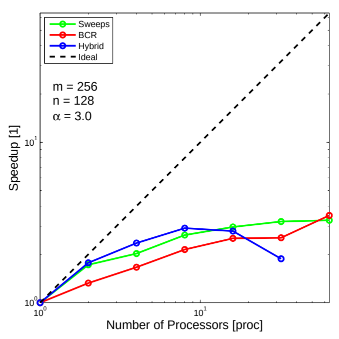


(h) Speedup for $n = 1024$

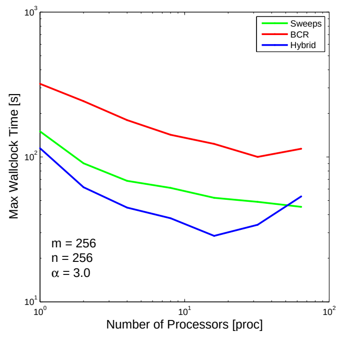
Figure 5.10: Timing figures for $\alpha = 3$ and block dimension $d = 128$.



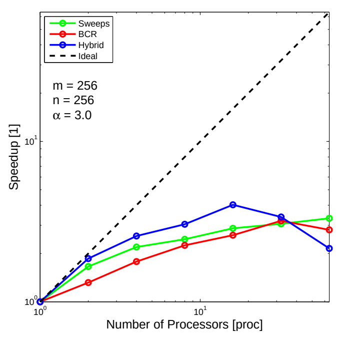
(a) Walltime for $n = 128$



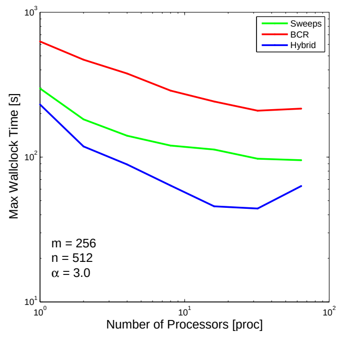
(b) Speedup for $n = 128$



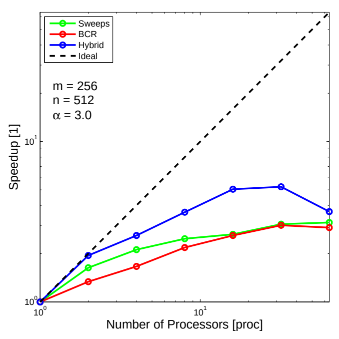
(c) Walltime for $n = 256$



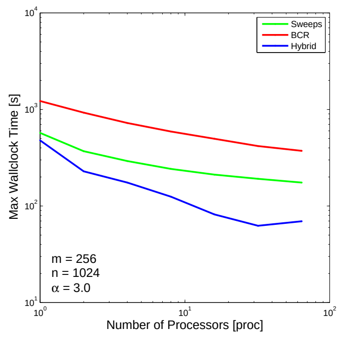
(d) Speedup for $n = 256$



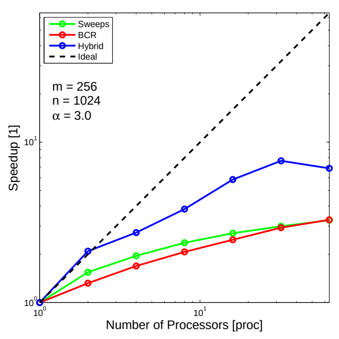
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$

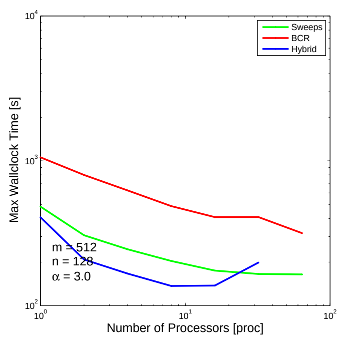


(g) Walltime for $n = 1024$

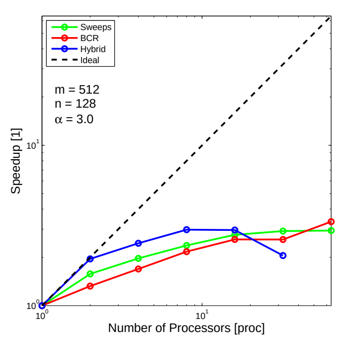


(h) Speedup for $n = 1024$

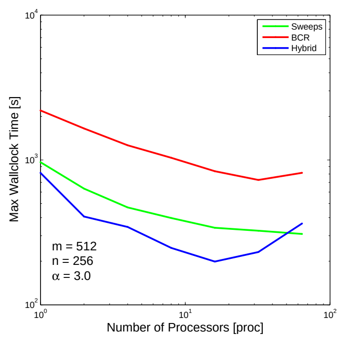
Figure 5.11: Timing figures for $\alpha = 3$ and block dimension $d = 256$.



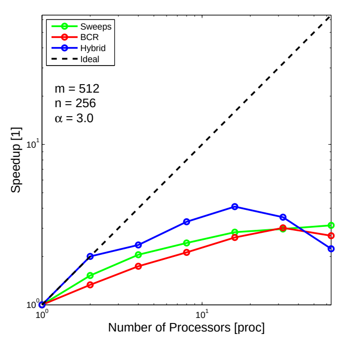
(a) Walltime for $n = 128$



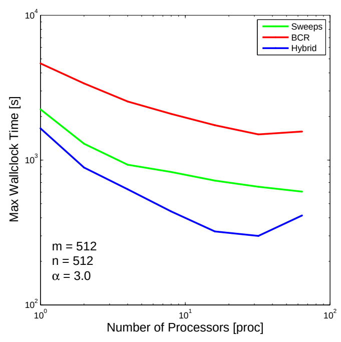
(b) Speedup for $n = 128$



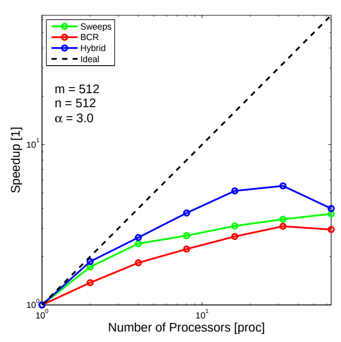
(c) Walltime for $n = 256$



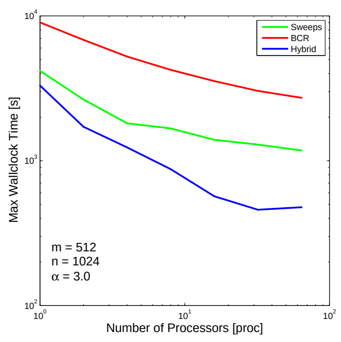
(d) Speedup for $n = 256$



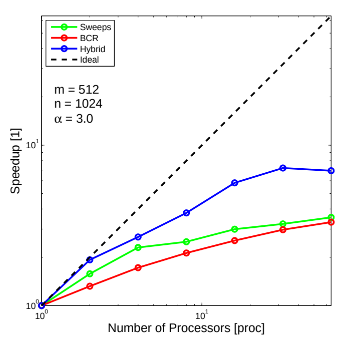
(e) Walltime for $n = 512$



(f) Speedup for $n = 512$



(g) Walltime for $n = 1024$



(h) Speedup for $n = 1024$

Figure 5.12: Timing figures for $\alpha = 3$ and block dimension $d = 512$.

sweep phase is composed of two independent elimination sweeps, where communication is performed only in order to “pass the torch” among processes.

It is true that for multiple processes a center process may stall this behavior, as e.g. an upwards elimination pass meets a process currently engaged in processing the downwards elimination pass. However, in the case $\mathcal{P} = 2$, both processes should terminate their first passes at approximately the same time, so time spent waiting should be minimal.

It turns out on closer inspection that one process takes twice the amount of time to complete the GAUSSELIMINATEPARALLEL phase of SWEEPINVERSEPARALLEL, than the other process, and this is indicative that one process manages to pass the torch and receive data from the other process in order to continue work. The second process, however, seems to stall, and is behavior that should be correctable.

One immediate feature that comes across in all speedup figures is that the Sweeps method does scale to speedup values that is greater than two, something unexpected considering the fact that the method performs two distinct elimination passes on $\mathbf{A}$, and at first sight would seem only to scale accordingly. The reason speedup progresses beyond this is that the complete inversion algorithm also incorporates an embarrassingly parallel construction step in generating $\mathbf{A}$ based on the results from the elimination sweep. This step can take place for a process immediately after it has completed both the upward and downward elimination passes, regardless of the status of its peers[1].

Ultimately, we can use SWEEPINVERSEPARALLEL relatively flexibly in order to distribute memory, without affecting speedup behavior or execution times much. In the light of differing block dimensions throughout $\mathbf{A}$, we can arbitrarily distribute rows among processes, and still expect the Gaussian elimination phase to terminate more or less at the same time for both corner processes. The reconstruction phase, independent as it is, will then depend cubically on the dimension of the blocks owned by each process. Thus with differing block dimensions and differing assigned block rows, the reconstruction phase execution times will differ among processes.

However, as center processes proceed on to construction of $\mathbf{G}$ before corner processes (since the elimination sweeps complete first for them), they will likely have terminated construction earlier than other processes that start construction later. One can then imagine assigning larger amounts of rows to them, and thus in taking longer to construct $\mathbf{G}$, they may terminate recon-

[1]As long as the neighbors have accepted any elimination sweep handoff the process has to do.

struction at the same time as other processes owning rows closer to the top and bottom that proceed to the reconstruction phase at later times.

5.2.4.2 Block Cyclic Reduction

For BCR, and in all cases where we have employed a uniform distribution, the speedup follows the same curve with speedup trailing off and becoming less and less efficient. However, as BCR should ideally be quickest to execute with $n = \mathcal{P}$, we are limited by the maximum of $\mathcal{P} = 64$ in these figures, where we have employed a minimum of $n = 128$. Thus in Fig. 5.13 is given a small example, for $n = 15$ and $n = 16$, in order to see speedup behavior as n approaches $\mathcal{P}$.

The reason to use the form $n = 2^4 - 1 = 15$ means that the elimination tree becomes complete and balanced, as seen in earlier figures in this paper, while for $n = 16$, a single extra corner reduction/production takes place, and the elimination tree appears as having an extra edge/node on top of the balanced case for $n = 15$.

The results from Fig. 5.13 do not show any qualitative differences, and while speedup is not very close to the ideal line, it seems to at least be consistently rising with more and more processes added to the job. It would be desirable, however, to have replicated this experiment for $n = 128$ or greater, in order to have a more valid comparison to the speedup runs provided.

In the case of basic distribution, we do see a consistent break in the speedup curve for $n = 256$ at $\mathcal{P} = 8$. This may be related to the unfortunate use of a non–uniform distribution. The reason this is unfortunate is that each elimination of a leaf node in the elimination tree for BCR can be done in parallel, however the parent of a leaf node, in the next level of elimination, needs to have had both its children nodes eliminated first before it can be eliminated itself. As we are using a non uniform distribution, the corner processes will have more nodes to eliminate than a center process, and we may end up in the situation that a node to be eliminated has to wait on a child node owned by a corner process, while the other child node has already been eliminated as it belonged to a center process that has less of a workload (due to fewer assigned rows).

Thus in the case of using BCR, the ideal is to use as many processes as possible[1] up to $\mathcal{P} = n$, and in the case of $\mathcal{P} < n$, with a uniform distribution of $\alpha = 1$. Again, this is all under the assumption of equal block dimensions throughout $\mathbf{A}$, and the situation may vary for unequal block dimensions.

[1]Providing communication costs eventually do not surpass computation costs.

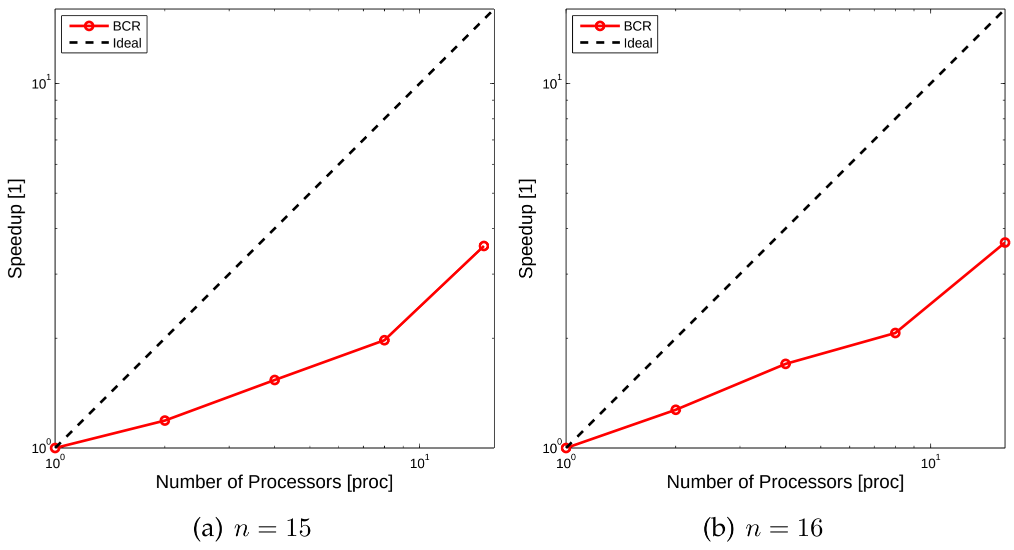


Figure 5.13: These two figures show that BCR does not suffer from qualitative differences in speedup depending on whether the elimination tree becomes a complete binary tree for $n = 15$, on the left, or an unbalanced tree and $n = 16$ on the right. In both cases, block dimension $d = 512$, and for $n = 15$, up to $\mathcal{P} = 15$ processes were used.

5.2.4.3 Hybrid

In nearly all cases[1], the Hybrid method performs exceedingly well for the speedup hop from $\mathcal{P} = 1$ to $\mathcal{P} = 2$. This is possible as the full $n \times n$ block **A** reduces to a tiny 2×2 block system embarrassingly parallel, and the BCR phase costs are negligible in the face of the Schur reduction/production costs. This leads to ideal speedup.

A strong qualitative difference visible among all speedup figures, is the poor performance the uniform distribution $\alpha = 1$ case suffers from going from $\mathcal{P} = 2$ to $\mathcal{P} = 4$. However, once this "hop" has happened, the uniformly distributed cases have relatively similar speedup to other cases of α. For the other cases, we observe the elimination of poor speedup behavior present in the case of $\alpha = 1$, and have smooth speedup curves.

Another feature present in the speedup results for the Hybrid method is that the curves stagnate and begin to turn downwards for high values of $\mathcal{P}$. This happens at lower values of $\mathcal{P}$ for lower values of n. This occurs since the BCR phase of execution begins to become significant as the ratio of $\mathcal{P}$ to n

[1]There are a couple of spurious cases which would likely disappear if the timings used were an average over an ensemble of benchmarks.

approaches 1.

This effect could be postponed by improving the relative efficiency of running the BCR algorithm over the Hybrid method, as we can see that BCR is an order of magnitude slower than the Hybrid method. Improving BCR would then lead to having BCR phase running times first begin to dominate for larger values of $\mathcal{P}$ (assuming constant n) than the case is now. One could slow down and spoil the performance of the Hybrid method in order to force this improvement on the ratio, and though the speedup curves may improve in shape, total running times would suffer.

It may also be noted that the Hybrid implementation is quicker than both BCR and Sweeps as long as the BCR phase of execution remains relatively modest. As it begins to dominate for certain cases and when $\mathcal{P}$ is large, the Hybrid method may run slower and terminate later than Sweeps.

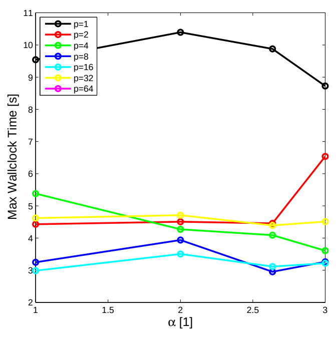
One question we seek to answer is for what choice of α the implemented Hybrid method performs best. To help answer this, we present comments now on both the given speedup figures for the Hybrid method, as well as introduce the new figures given in Fig. 5.14, Fig. 5.15, and Fig. 5.16. These figures all present the measured walltime for the Hybrid algorithm for differing block dimension d, number n of block rows in $\mathbf{A}$, and the parameter of interest α.

Each figure presents the measured walltime as a function of α on the left column, and as a function of the number of participating processes $\mathcal{P}$ on the right column. Both columns present the same data, but from differing perspectives in an effort to answer our question.

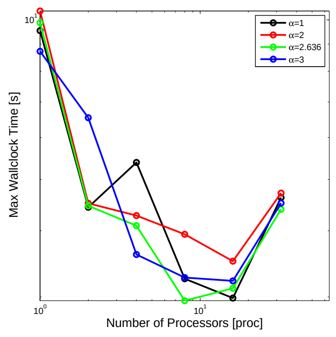
Looking at the right column in all three figures, we can see how increasing the number of participating processes generally decreases computation time, up to a certain point where the BCR phase of the algorithm likely begins to dominate. We can see that this point is reached for greater and greater values of $\mathcal{P}$ as the size of $\mathbf{A}$ represented by n increases.

These curves also show how for $\alpha = 1$, moving from 2 to 4 processes generally leads to an increase in computation time, which is also evidenced in the speedup curves presented earlier. This feature is then largely eliminated by choosing one of the other three tested values of α. As for these values, it is not uniquely clear which value is preferable. Thus we can choose a lower value of α, e.g. 2, for the sake of memory allocation concerns[1] while still satisfying the requirement for adequate speedup behavior.

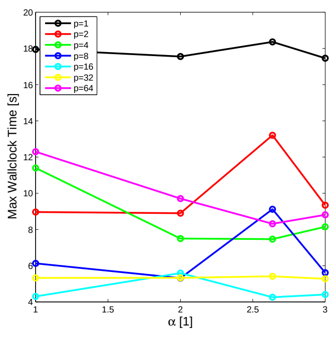
[1]Assuming all participating processes have the same memory available and that we're not dealing with a symmetric multiprocessing architecture, a corner process would be using twice as much memory than a central process for storing $\mathbf{A}$, and would in turn limit our treatable problem size.



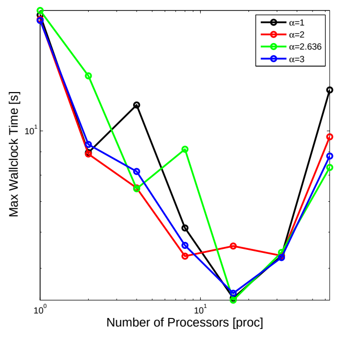
(a) $n = 128$



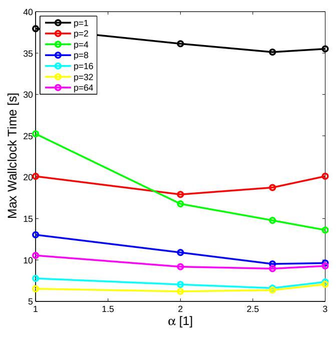
(b) $n = 128$



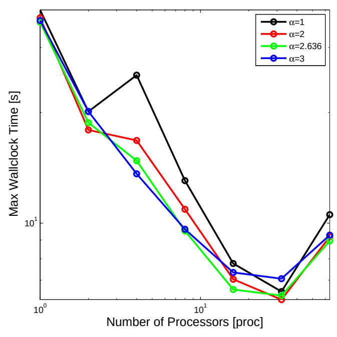
(c) $n = 256$



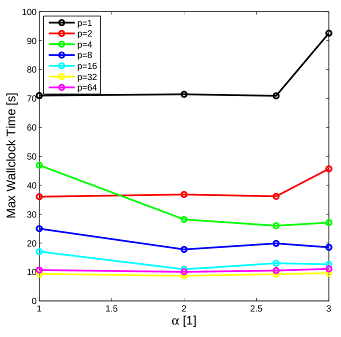
(d) $n = 256$



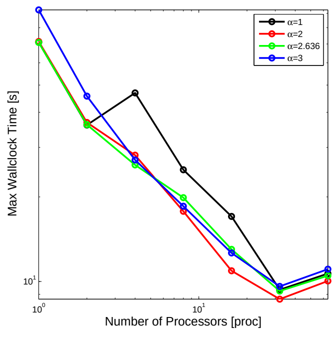
(e) $n = 512$



(f) $n = 512$

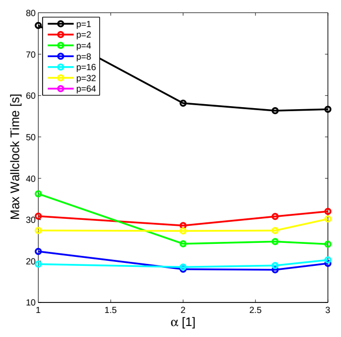


(g) $n = 1024$

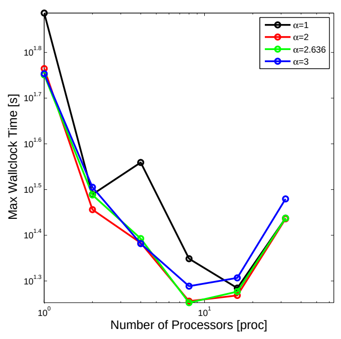


(h) $n = 1024$

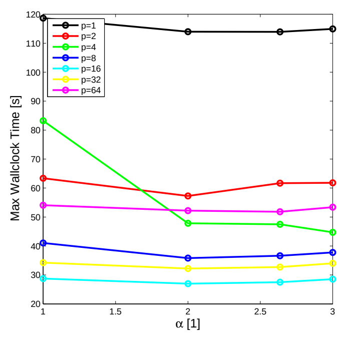
Figure 5.14: Choosing α for block dimension $d = 128$.



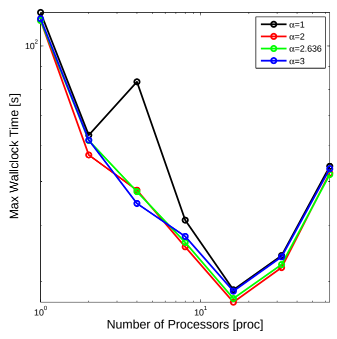
(a) $n = 128$



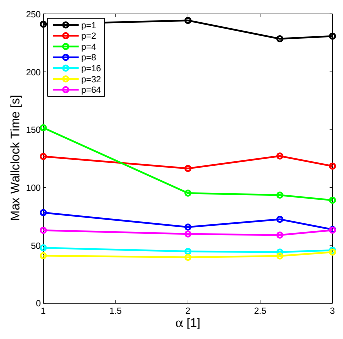
(b) $n = 128$



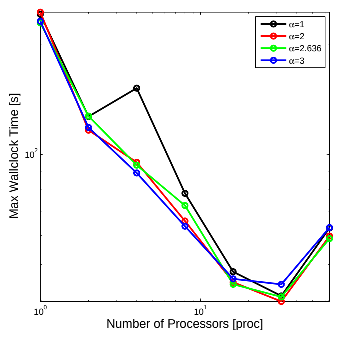
(c) $n = 256$



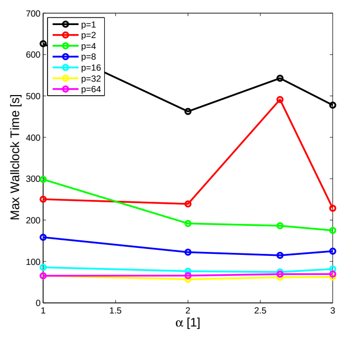
(d) $n = 256$



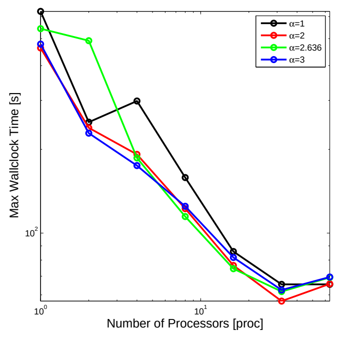
(e) $n = 512$



(f) $n = 512$

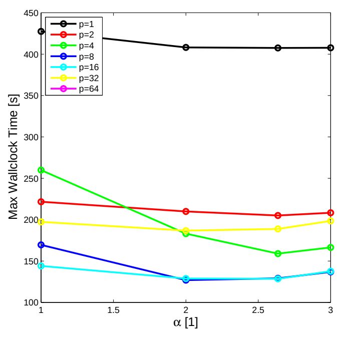


(g) $n = 1024$

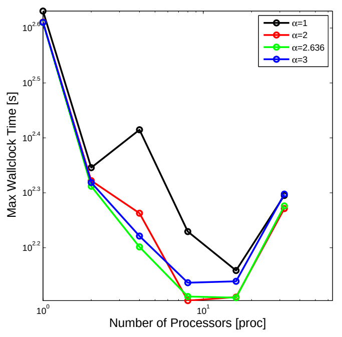


(h) $n = 1024$

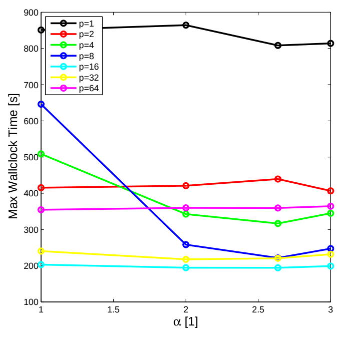
Figure 5.15: Choosing α for block dimension $d = 256$.



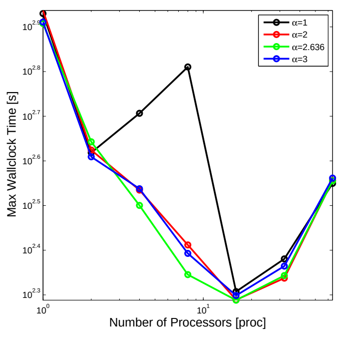
(a) $n = 128$



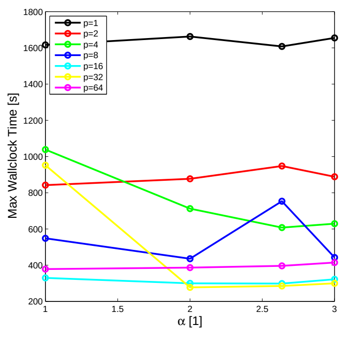
(b) $n = 128$



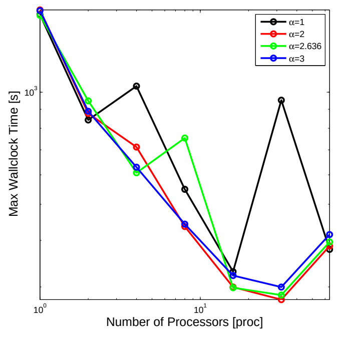
(c) $n = 256$



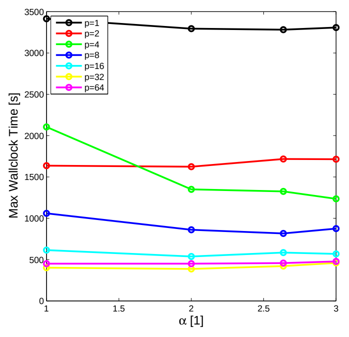
(d) $n = 256$



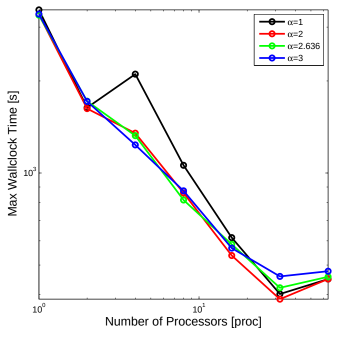
(e) $n = 512$



(f) $n = 512$



(g) $n = 1024$



(h) $n = 1024$

Figure 5.16: Choosing α for block dimension $d = 512$.

Looking on the left column of figures Fig. 5.14, Fig. 5.15, and Fig. 5.16, we would expect for $\alpha = 2.636$ to have a minimum value for each curve, if this value of α were to be universally optimal. This is not the case. However, what we do find is that in many cases the value for $\alpha = 1$ tends to be a maximum. This can also be interpreted by looking at the right column, and seeing how the black curves for $\alpha = 1$ tend to be above all other choices of α. Thus $\alpha = 1$ seems almost to be universally a poor choice.

Chapter 6

Conclusion

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

6.1 Results

A Sweep method has been developed that delivers a linear complexity algorithm for the determination of the block tridiagonal elements of **G**. It has been shown that this method is far more preferable to standard Gaussian elimination, and it has furthermore been parallelized, although with limited speedup.

A Hybrid method has been developed that provides good speedup figures for system sizes that are typical for common applications of NEGF simulations. Although speedup figures have only been tabulated for up to $\mathcal{P} = 64$ processes, they have been promising and $\mathcal{P} = 64$ more than covers what a current personal workstation would have available.

The tridiagonal optimized methods presented in this thesis, with the exception of GEINVERSETRI, have all been implemented within an application based on the semi–empirical extended Hückel framework for electron structure calculations. The code is proving itself already, delivering reliable results for a range of situations, but the real task is for it to tackle systems as of yet unprecedented size. The code is being used by research groups abroad.

An obvious immediate application for the Hybrid method is the simulation of an experiment of a 70nm long carbon nanotube field–effect transistor [40]. It is an ideal candidate for study as preliminary simulations on reduced size systems have shown promising results that measurements achieved in experiment can be numerically verified, and thus lend credibility to the program and algorithms involved.

6.2 Transmission

A result not explicitly treated in this thesis, is the case of determining a portion of the Green's function matrix $\mathbf{G}$ for use in transmission calculations. In [1][1], work was done in order to both prove the ability to use diagonal blocks of $\mathbf{G}$ for transmission calculations, and that a method based on tridiagonal optimized Sweeps was especially well suited for the task, being generally quicker, in all tested cases. The method further enables the possibility of improved cache usage, leading to impressive performance gains.

The task of transmission calculation should be equally solvable using the parallelized SWEEPINVERSEPARALLEL, however BCR or the Hybrid method may prove far more efficient, as they are capable of speedups greater than 2 and only a reduction phase is required in order to provide a single, sufficient block $\mathbf{g}_{kk}$ for which to determine transmission with.

6.3 Future Work

Further work can be done in the generation of speedup figures. One desire is to run the parallel algorithms for greater process counts, in order to clarify improving speedup behavior as seen earlier, e.g. for BCR. Furthermore, it is also of interest to see how the different portions of code contributes to overall running times. Rudimentary visualization confirms a maximum speedup of 2 for the Sweeps algorithm when considering only the elimination phase, and may reveal characteristics for the BCR and Hybrid algorithms.

A more detailed complexity analysis of the algorithms, allowing for arbitrary variation on the block sizes of $\mathbf{A}$ is also of interest. This would potentially allow for an improved deterministic method of assigning row possessions to participating processes, and which can be decided prior to $\mathbf{A}$ being inverted.

Furthermore, the ability to tune between a generating a load balance more optimized for efficiency in memory distribution and one optimized for execution speed is a desirable feature, especially as problem sizes approach hardware limits. This, I suspect, may be a nontrivial problem, however, in the sense

[1]Included in appendix A.

of the traveling salesperson problem[1].

An analysis of the stability of the Hybrid method is also of interest, and has not been rigorously analysed. In all generated numerical cases and examples from the literature [1], inversions were stable, with relative norm errors on the block tridiagonal $\mathrm{Trid}_{\mathbf{A}}\left\{\mathbf{G}\right\}$ being on the order of about 10^{-10} or less. Furthermore, transmission calculations were found to be attained with higher accuracy [1].

Finally, further optimization can be performed on the implemented version of Hybrid to bring it more closely in line with the theoretical operation count as described in this thesis. This may possibly worsen the speedup curves slightly, but should improve the algorithm's prefactor cost in execution with respect to problem size n and lead to quicker running times overall. A communications issue affecting speedup for $\mathcal{P} = 2$ for the Sweeps algorithm also needs clarification.

[1]This class of problems belong to the class of NP–complete problems. They are problems of "size" n for which no known algorithm exists that can solve them with a time written as a polynomial of n, though a solution can be checked in polynomial time.

Appendix A

Article: Block Tridiagonal Matrix Inversion and Fast Transmission Calculations

Accepted at the Journal of Computational Physics

Listed in references as [1].

Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Physics 227 (2008) 3174–3190

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Block tridiagonal matrix inversion and fast transmission calculations

Dan Erik Petersen[a,*], Hans Henrik B. Sørensen[b], Per Christian Hansen[b], Stig Skelboe[a], Kurt Stokbro[c]

[a] *Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark*
[b] *Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads, Bldg. 321, DK-2800 Lyngby, Denmark*
[c] *Nanoscience Center, University of Copenhagen, Universitetsparken 5d, DK-2100 Copenhagen, Denmark*

Received 24 April 2007; received in revised form 15 November 2007; accepted 19 November 2007
Available online 8 December 2007

Abstract

A method for the inversion of block tridiagonal matrices encountered in electronic structure calculations is developed, with the goal of efficiently determining the matrices involved in the Fisher–Lee relation for the calculation of electron transmission coefficients. The new method leads to faster transmission calculations compared to traditional methods, as well as freedom in choosing alternate Green's function matrix blocks for transmission calculations. The new method also lends itself to calculation of the tridiagonal part of the Green's function matrix. The effect of inaccuracies in the electrode self-energies on the transmission coefficient is analyzed and reveals that the new algorithm is potentially more stable towards such inaccuracies.
© 2007 Elsevier Inc. All rights reserved.

PACS: 71.15.−m; 02.70.−c

Keywords: Matrix inversion; Electron transport; Transmission; Density functional theory

1. Introduction

Quantum transport simulations have become an important theoretical tool for investigating the electrical properties of nanoscale systems, both in the semi-empirical approach [1–4] and full ab initio approach [5–8]. The basis for the approach is the Landauer–Büttiker model of coherent transport, where the electrical properties of a nanoscale constriction is described by the transmission coefficients of a number of one-electron states propagating coherently through the constriction. The approach has been used successfully to describe the electrical properties of a wide range of nanoscale systems, including atomic wires, molecules and interfaces

* Corresponding author. Tel.: +45 35 32 14 00; fax: +45 35 32 14 01.
E-mail address: danerik@diku.dk (D.E. Petersen).

0021-9991/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2007.11.035

[9–18]. In order to apply the method to semiconductor device simulation, it is necessary to handle systems comprising millions of atoms, and this will require new, efficient algorithms for calculating the transmission coefficient.

In this paper, ideas and calculations behind an algorithm that provides an improvement over a widely popular technique employed in the calculation of transmission coefficient of so-called two-probe systems [15] is presented. A two-probe system consists of three regions: a left electrode region, a central scattering region and a right electrode region. The electrode regions are semi-infinite periodic systems, and the scattering region connects the two electrode regions. A one-electron tight-binding Hamiltonian is used to describe the electronic structure of the system. The tight-binding Hamiltonian can be obtained from a semi-empirical tight-binding description as obtained from an extended Hückel model [19] or through a first-principles approach as obtained when using a self-consistent density-functional Kohn–Sham Hamiltonian [20].

In the pursuit of determining the electronic structure of molecules, bulk crystals and two-probe systems, associated self-consistent DFT calculations, relevant Green's functions and ultimately calculation of the transmission of two-probe systems all involve the problem of matrix inversion in some form or another. This paper deals with matrices of a *block tridiagonal* form, which lie at the center of the problems to be solved. Block matrices will be denoted with uppercase bold letters, while lower case bold letters refer to sub-block matrices of their uppercase counterparts.

Throughout this paper, it is assumed that block tridiagonal matrix, $\mathbf{A}$, is dealt with and that it is to be inverted in order to obtain the Green's function matrix (or a part thereof). In the process of finding the Green's function matrix $\mathbf{G} = \mathbf{A}^{-1}$ that enters in DFT theory, the following equation sets up the problem [21]:

$$\mathbf{A} = \varepsilon\mathbf{S} - \mathbf{H} - \mathbf{\Sigma}^{\mathrm{L}} - \mathbf{\Sigma}^{\mathrm{R}}. \tag{1}$$

In the above expression $\mathbf{S}$ is an overlap matrix, $\mathbf{H}$ is the Hamiltonian of the system and $\mathbf{\Sigma}^{\mathrm{L}}$ and $\mathbf{\Sigma}^{\mathrm{R}}$ are the self-energies from the *left* and *right* semi-infinite electrodes, respectively. Furthermore, the matrix $\mathbf{G}$ depends on the variable ε that dictates the energy of an incoming one-electron coherent wave for which it is desired to investigate the transmission through the system. The methods developed in this paper are designed for a fixed value of ε.

The individual blocks of the matrix $\mathbf{A}$ are denoted $\mathbf{a}_{ij}$ and are assumed to be dense, complex matrices along the tridiagonal. The diagonal blocks are square matrices, while the off-diagonal blocks are typically rectangular. The structure of $\mathbf{A}$ for two relevant cases is shown in Figs. 1 and 2.

A method to obtain the Green's function matrix $\mathbf{G}$ is now devised, much in the same spirit as [22]. In order to do so, the matrix to be inverted, $\mathbf{A}$, is augmented with the identity matrix, $\mathbf{I}$.

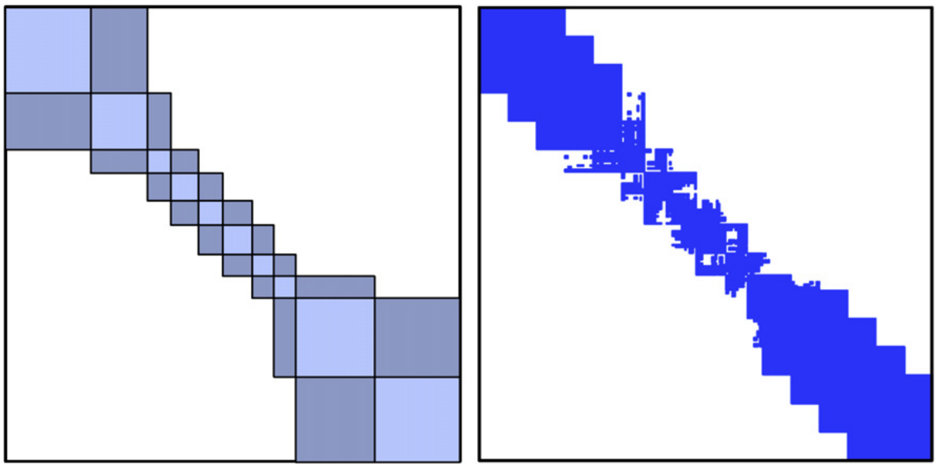


Fig. 1. The block-tridiagonal and sparsity structure for the Au111–AR example [17]. The matrix is of dimension 1295 × 1295, split up along the diagonal in blocks of order 243, 162, 66, 79, 69, 84, 62, 62, 225, and 243 from upper left to lower right, along with corresponding off-diagonal blocks.

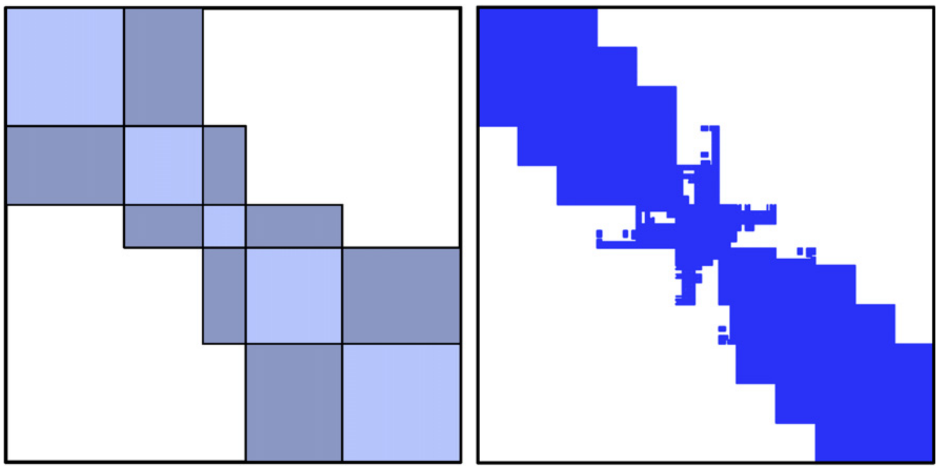


Fig. 2. The block-tridiagonal and sparsity structure for the Au111–DTB example [18]. The matrix is of dimension 943 × 943, split up along the diagonal in blocks of order 243, 162, 88, 198 and 243 from upper left to lower right, along with corresponding off-diagonal blocks.

$$\left[\mathbf{A}\,\middle|\,\mathbf{I}\right] = \left(\begin{array}{cccc|cccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & & \mathbf{i}_{22} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & & \mathbf{i}_{33} & \\ & & \ddots & \ddots\ \ddots & & & & \ddots \end{array}\right) \tag{2}$$

Each diagonal block of the identity matrix, $\mathbf{i}_{ii}$ has the same square block size of the corresponding block $\mathbf{a}_{ii}$ of the matrix **A**, and are themselves identity matrices.

The organization and shape of the matrix blocks in **A** are related to the topology of the two probe system. Looking at, e.g. Fig. 1, portions of the electrodes can be identified as the regions comprised of larger blocks towards the corner of the matrix, while the more sparsely populated central region of the system is identified as a series of smaller matrix blocks in the center of **A**. The top left corner of **A** attaches to the left electrode, while the lower right corner attaches to the right electrode.

The expression *augmented matrix* $[\mathbf{A}|\mathbf{I}]$ is equivalent to the equation $\mathbf{AG} = \mathbf{I}$ (cf. [23]). By manipulating the augmented matrix through a series of operations such that the left side, **A**, is reduced to the identity matrix **I**, we will obtain the augmented matrix $[\mathbf{I}|\mathbf{G}]$ where the inverse of **A**, namely $\mathbf{G} = \mathbf{A}^{-1}$, can be read on the right. This is done by illustrating the forward and backward block Gaussian elimination steps, and then combining the results.

Calculating all of **G** is ultimately not of interest. Only a block $\mathbf{g}_{ij}$ of **G** to be used in further transmission calculations will be determined. It is the particular choice of $\mathbf{g}_{ij}$ and the procedure for its calculation that separates the new transmission calculation method from previous algorithms.

This paper is organized as follows. The notation and block Gaussian elimination technique on which the methods used in this paper is based on is described in Section 2. Section 3 shows how the result of block Gaussian elimination is used to generate the Green's function matrix **G**. In Section 4, the calculation of transmission values via a traditional method and a new method is explained. The new method is then benchmarked against the traditional, baseline method, via a consideration of computational complexity, as well as measured speedup times in Section 5. The effects of perturbed surface Green's function matrices on the transmission accuracy, and conclusions on which portions of **G** would lead to more accurate transmission calculations is considered in Section 6. Conclusions are finally presented in Section 7.

2. Forward and backward block Gaussian elimination

The forward procedure is characterized with the superscript L since the elimination procedure proceeds from the *left* electrode towards the right.

A block Gaussian elimination step is performed on the matrix given in Eq. (2) by multiplying the first block row by the matrix $\mathbf{c}_1^{\mathrm{L}} = -\mathbf{a}_{21}\mathbf{a}_{11}^{-1}$ and subsequently adding it to the second block row. This produces a zero block in the (2,1) position:

$$\left(\begin{array}{cccc|cccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{a}_{21}+\mathbf{c}_1^{\mathrm{L}}\mathbf{a}_{11} & \mathbf{a}_{22}+\mathbf{c}_1^{\mathrm{L}}\mathbf{a}_{12} & \mathbf{a}_{23} & & \mathbf{c}_1^{\mathrm{L}}\mathbf{i}_{11} & \mathbf{i}_{22} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & & \mathbf{i}_{33} & \\ & & \ddots & \ddots \ \ddots & & & & \ddots \end{array}\right)$$
$$= \left(\begin{array}{cccc|cccc} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{0} & \mathbf{a}_{22}-\mathbf{a}_{21}\mathbf{a}_{11}^{-1}\mathbf{a}_{12} & \mathbf{a}_{23} & & \mathbf{a}_{21}\mathbf{a}_{11}^{-1}\mathbf{i}_{11} & \mathbf{i}_{22} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & & \mathbf{i}_{33} & \\ & & \ddots & \ddots \ \ddots & & & & \ddots \end{array}\right). \quad (3)$$

Next, a block Gaussian elimination step is performed by multiplying the second row by the factor $\mathbf{c}_2^{\mathrm{L}} = -\mathbf{a}_{32}(\mathbf{a}_{22} - \mathbf{a}_{21}\mathbf{a}_{11}^{-1}\mathbf{a}_{12})^{-1}$ and subsequently adding it to the third row. This produces a zero block in the (3,2) position. A recursive routine that will complete a full forward block Gaussian elimination is now defined.

$$\begin{array}{ll} \mathbf{d}_{11}^{\mathrm{L}} = \mathbf{a}_{11} & \mathbf{c}_1^{\mathrm{L}} = -\mathbf{a}_{21}(\mathbf{d}_{11}^{\mathrm{L}})^{-1} \\ \mathbf{d}_{22}^{\mathrm{L}} = \mathbf{a}_{22} - \mathbf{a}_{21}(\mathbf{d}_{11}^{\mathrm{L}})^{-1}\mathbf{a}_{12} & \mathbf{c}_2^{\mathrm{L}} = -\mathbf{a}_{32}(\mathbf{d}_{22}^{\mathrm{L}})^{-1} \\ \mathbf{d}_{33}^{\mathrm{L}} = \mathbf{a}_{33} - \mathbf{a}_{32}(\mathbf{d}_{22}^{\mathrm{L}})^{-1}\mathbf{a}_{23} & \mathbf{c}_3^{\mathrm{L}} = -\mathbf{a}_{43}(\mathbf{d}_{33}^{\mathrm{L}})^{-1} \\ \quad\vdots & \quad\vdots \\ \mathbf{d}_{ii}^{\mathrm{L}} = \mathbf{a}_{ii} - \mathbf{a}_{i,i-1}(\mathbf{d}_{i-1,i-1}^{\mathrm{L}})^{-1}\mathbf{a}_{i-1,i} & \mathbf{c}_i^{\mathrm{L}} = -\mathbf{a}_{i+1,i}(\mathbf{d}_{ii}^{\mathrm{L}})^{-1} \\ \quad\vdots & \quad\vdots \\ \mathbf{d}_{nn}^{\mathrm{L}} = \mathbf{a}_{nn} - \mathbf{a}_{n,n-1}(\mathbf{d}_{n-1,n-1}^{\mathrm{L}})^{-1}\mathbf{a}_{n-1,n} & \mathbf{c}_{n-1}^{\mathrm{L}} = -\mathbf{a}_{n,n-1}(\mathbf{d}_{n-1,n-1}^{\mathrm{L}})^{-1} \end{array}$$

The matrices $\mathbf{d}_{ii}^{\mathrm{L}}$ are the diagonal blocks of the resulting matrix on the left. It can be seen that each diagonal block is calculated from the following relation:

$$\mathbf{d}_{ii}^{\mathrm{L}} = \mathbf{a}_{ii} + \mathbf{c}_{i-1}^{\mathrm{L}}\mathbf{a}_{i-1,i}, \quad \text{where } i = 2,3,\ldots,n \text{ and } \mathbf{d}_{11}^{\mathrm{L}} = \mathbf{a}_{11}, \quad (4)$$

and each row multiplication factor is:

$$\mathbf{c}_i^{\mathrm{L}} = -\mathbf{a}_{i+1,i}(\mathbf{d}_{ii}^{\mathrm{L}})^{-1}, \quad \text{where } i = 1,2,\ldots,n-1. \quad (5)$$

The similar backward procedure is characterized with the superscript R since the elimination procedure moves from the *right* electrode towards the left. The derivation of the backwards recursive expressions follows that of the forward elimination. Each diagonal block can be calculated from the following relation:

$$\mathbf{d}_{ii}^{\mathrm{R}} = \mathbf{a}_{ii} + \mathbf{c}_{i+1}^{\mathrm{R}}\mathbf{a}_{i+1,i}, \quad \text{where } i = n-1,\ldots,2,1 \text{ and } \mathbf{d}_{nn}^{\mathrm{R}} = \mathbf{a}_{nn}, \quad (6)$$

and each row multiplication factor is:

$$\mathbf{c}_i^{\mathrm{R}} = -\mathbf{a}_{i-1,i}(\mathbf{d}_{ii}^{\mathrm{R}})^{-1}, \quad \text{where } i = n, \ldots, 3, 2. \tag{7}$$

3. Combining the two procedures

After a complete forward and backward block Gaussian elimination sweep, the augmented matrices, named $[\mathbf{D}^{\mathrm{L}}|\mathbf{J}^{\mathrm{L}}]$ and $[\mathbf{D}^{\mathrm{R}}|\mathbf{J}^{\mathrm{R}}]$, respectively, will look as follows where the matrices $\mathbf{J}^{\mathrm{L}}$ and $\mathbf{J}^{\mathrm{R}}$ are lower and upper block triangular, respectively:

$$[\mathbf{D}^{\mathrm{L}} \mid \mathbf{J}^{\mathrm{L}}] = \left(\begin{array}{cccc|cccc} \mathbf{d}_{11}^{\mathrm{L}} & \mathbf{a}_{12} & & & \mathbf{i}_{11} & & & \\ \mathbf{0} & \mathbf{d}_{22}^{\mathrm{L}} & \mathbf{a}_{23} & & \mathbf{c}_1^{\mathrm{L}}\mathbf{i}_{11} & \mathbf{i}_{22} & & \\ & \mathbf{0} & \mathbf{d}_{33}^{\mathrm{L}} & \mathbf{a}_{34} & \mathbf{c}_{2,1}^{\mathrm{L}}\mathbf{i}_{11} & \mathbf{c}_2^{\mathrm{L}}\mathbf{i}_{22} & \mathbf{i}_{33} & \\ & & \ddots & \ddots\ \ddots & \vdots & \vdots & & \ddots \end{array}\right), \tag{8}$$

$$[\mathbf{D}^{\mathrm{R}} \mid \mathbf{J}^{\mathrm{R}}] = \left(\begin{array}{cccc|cccc} \mathbf{d}_{11}^{\mathrm{R}} & \mathbf{0} & & & \mathbf{i}_{11} & \mathbf{c}_2^{\mathrm{R}}\mathbf{i}_{22} & \mathbf{c}_{2,3}^{\mathrm{R}}\mathbf{i}_{33} & \cdots \\ \mathbf{a}_{21} & \mathbf{d}_{22}^{\mathrm{R}} & \mathbf{0} & & & \mathbf{i}_{22} & \mathbf{c}_3^{\mathrm{R}}\mathbf{i}_{33} & \cdots \\ & \mathbf{a}_{32} & \mathbf{d}_{33}^{\mathrm{R}} & \mathbf{0} & & & \mathbf{i}_{33} & \cdots \\ & & \ddots & \ddots\ \ddots & & & & \ddots \end{array}\right). \tag{9}$$

Here, the following notation was introduced:

$$\left.\begin{array}{l} \mathbf{c}_1^{\mathrm{R}}\mathbf{c}_2^{\mathrm{R}}\cdots\mathbf{c}_i^{\mathrm{R}} = \mathbf{c}_{1,2,\ldots,i}^{\mathrm{R}} \\ \mathbf{c}_i^{\mathrm{L}}\mathbf{c}_{i-1}^{\mathrm{L}}\cdots\mathbf{c}_1^{\mathrm{L}} = \mathbf{c}_{i,i-1,\ldots,1}^{\mathrm{L}} \end{array}\right\} \text{ where } i = 1, 2, \ldots, n. \tag{10}$$

Combining the results obtained from Eqs. (2), (8), and (9) by employing the fact that

$$\mathbf{AG} = \mathbf{I}, \quad \mathbf{D}^{\mathrm{L}}\mathbf{G} = \mathbf{J}^{\mathrm{L}}, \quad \mathbf{D}^{\mathrm{R}}\mathbf{G} = \mathbf{J}^{\mathrm{R}}, \tag{11}$$

the expression

$$(\mathbf{A} - \mathbf{D}^{\mathrm{L}} - \mathbf{D}^{\mathrm{R}})\mathbf{G} = \mathbf{I} - \mathbf{J}^{\mathrm{L}} - \mathbf{J}^{\mathrm{R}} \tag{12}$$

is examined, which can be viewed as the following augmented matrix expression:

$$\left[\begin{array}{c|c} \mathbf{B} & \mathbf{F} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{I} \end{array}\right] - \left[\begin{array}{c|c} \mathbf{D}^{\mathrm{L}} & \mathbf{J}^{\mathrm{L}} \end{array}\right] - \left[\begin{array}{c|c} \mathbf{D}^{\mathrm{R}} & \mathbf{J}^{\mathrm{R}} \end{array}\right], \tag{13}$$

where

$$\mathbf{B} = \begin{pmatrix} \mathbf{a}_{11} - \mathbf{d}_{11}^{\mathrm{L}} - \mathbf{d}_{11}^{\mathrm{R}} & & & \\ & \mathbf{a}_{22} - \mathbf{d}_{22}^{\mathrm{L}} - \mathbf{d}_{22}^{\mathrm{R}} & & \\ & & \mathbf{a}_{33} - \mathbf{d}_{33}^{\mathrm{L}} - \mathbf{d}_{33}^{\mathrm{R}} & \\ & & & \ddots \end{pmatrix} \tag{14}$$

and

$$\mathbf{F}=\begin{pmatrix} -\mathbf{i}_{11} & -\mathbf{c}_2^{\mathrm{R}} & -\mathbf{c}_{2,3}^{\mathrm{R}} & -\mathbf{c}_{2,3,4}^{\mathrm{R}} & \cdots \\ -\mathbf{c}_1^{\mathrm{L}} & -\mathbf{i}_{22} & -\mathbf{c}_3^{\mathrm{R}} & -\mathbf{c}_{3,4}^{\mathrm{R}} & \cdots \\ -\mathbf{c}_{2,1}^{\mathrm{L}} & -\mathbf{c}_2^{\mathrm{L}} & -\mathbf{i}_{33} & -\mathbf{c}_4^{\mathrm{R}} & \cdots \\ -\mathbf{c}_{3,2,1}^{\mathrm{L}} & -\mathbf{c}_{3,2}^{\mathrm{L}} & -\mathbf{c}_3^{\mathrm{L}} & -\mathbf{i}_{44} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{15}$$

When **B** is subsequently reduced to the identity matrix **I**, **F** will simultaneously be transformed into the Green's function matrix **G**. In other words, the Green's function matrix sought for can be expressed as $\mathbf{G}=\mathbf{B}^{-1}\mathbf{F}$. The Green's function matrix is:

$$\mathbf{G}=\begin{pmatrix} \mathbf{g}_{11} & \mathbf{g}_{11}\mathbf{c}_2^{\mathrm{R}} & \mathbf{g}_{11}\mathbf{c}_{2,3}^{\mathrm{R}} & \cdots & \mathbf{g}_{11}\mathbf{c}_{2,\dots,n}^{\mathrm{R}} \\ \mathbf{g}_{22}\mathbf{c}_1^{\mathrm{L}} & \mathbf{g}_{22} & \mathbf{g}_{22}\mathbf{c}_3^{\mathrm{R}} & \cdots & \mathbf{g}_{22}\mathbf{c}_{3,\dots,n}^{\mathrm{R}} \\ \mathbf{g}_{33}\mathbf{c}_{2,1}^{\mathrm{L}} & \mathbf{g}_{33}\mathbf{c}_2^{\mathrm{L}} & \mathbf{g}_{33} & \cdots & \mathbf{g}_{33}\mathbf{c}_{4,\dots,n}^{\mathrm{R}} \\ \mathbf{g}_{44}\mathbf{c}_{3,2,1}^{\mathrm{L}} & \mathbf{g}_{44}\mathbf{c}_{3,2}^{\mathrm{L}} & \mathbf{g}_{44}\mathbf{c}_3^{\mathrm{L}} & \cdots & \mathbf{g}_{44}\mathbf{c}_{5,\dots,n}^{\mathrm{R}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{g}_{nn}\mathbf{c}_{n-1,\dots,1}^{\mathrm{L}} & \mathbf{g}_{nn}\mathbf{c}_{n-1,\dots,2}^{\mathrm{L}} & \mathbf{g}_{nn}\mathbf{c}_{n-1,\dots,3}^{\mathrm{L}} & \cdots & \mathbf{g}_{nn} \end{pmatrix}, \tag{16}$$

where the following expression for the diagonal blocks of the Green's function matrix is introduced:

$$\mathbf{g}_{ii}=-\mathbf{b}_{ii}^{-1}=\left(-\mathbf{a}_{ii}+\mathbf{d}_{ii}^{\mathrm{L}}+\mathbf{d}_{ii}^{\mathrm{R}}\right)^{-1} \quad \text{where } i=1,2,\dots,n. \tag{17}$$

Off-diagonal entries are then calculated via appropriate multiplications with calculated diagonal block matrices and factors obtained during block Gaussian elimination as follows using the notation given in Eq. (10):

$$\mathbf{g}_{ij}=\mathbf{g}_{ii}\mathbf{c}_{i+1,i+2,\dots,j-1,j}^{\mathrm{R}} \quad \text{for } i<j \tag{18}$$

$$\mathbf{g}_{ij}=\mathbf{g}_{ii}\mathbf{c}_{i-1,i-2,\dots,j+1,j}^{\mathrm{L}} \quad \text{for } i>j. \tag{19}$$

4. Computation of transmission

The calculation of transmission t, given by the following Fisher–Lee [24] relation obtained in non–equilibrium Green's function theory, can be expressed as (cf. [21,25]):

$$t=\mathrm{Tr}\{\mathbf{G}\boldsymbol{\Gamma}^{\mathrm{L}}\mathbf{G}^{\dagger}\boldsymbol{\Gamma}^{\mathrm{R}}\}. \tag{20}$$

Here 'Tr' denotes a matrix *trace* operation, and the dagger denotes Hermitian conjugation. Regarding $\boldsymbol{\Gamma}^{\mathrm{L}}$ and $\boldsymbol{\Gamma}^{\mathrm{R}}$, the superscripts indicate *left* and *right* electrode contact leads. These matrices are defined from the electrode self-energy [21]:

$$\boldsymbol{\Gamma}^{\mathrm{L}}=\hat{\imath}\left(\boldsymbol{\Sigma}^{\mathrm{L}}-\left(\boldsymbol{\Sigma}^{\mathrm{L}}\right)^{\dagger}\right),\quad \boldsymbol{\Gamma}^{\mathrm{R}}=\hat{\imath}\left(\boldsymbol{\Sigma}^{\mathrm{R}}-\left(\boldsymbol{\Sigma}^{\mathrm{R}}\right)^{\dagger}\right), \tag{21}$$

where $\hat{\imath}$ is the imaginary unit. These matrices are only non–zero in the $(1,1)$ block for $\boldsymbol{\Sigma}^{\mathrm{L}}$ and $\boldsymbol{\Gamma}^{\mathrm{L}}$, and in the (n,n) block for the case $\boldsymbol{\Sigma}^{\mathrm{R}}$ and $\boldsymbol{\Gamma}^{\mathrm{R}}$ (cf. [26–28]).

Two methods are now presented that can be used to calculate the transmission t given in Eq. (20).

4.1. Coupling method

The coupling method is by far the popular method of choice in the literature when transmission is to be calculated via the Green's function formalism (see [26–28]). The method is introduced here, and regarded as the *baseline* method to compare the new transmission calculation method to later in the paper.

In this method, the coupling between the left and right leads is calculated, and the transmission computed accordingly. This coupling is denoted as $\mathbf{g}_{n1}$, and it resides as the lowest left corner of the Green's function matrix $\mathbf{G}$. The calculation of transmission for a particular energy ε then becomes (cf. [26]):

$$t = \mathrm{Tr}\{\mathbf{g}_{n1}\gamma^{\mathrm{L}}_{11}\mathbf{g}^{\dagger}_{n1}\gamma^{\mathrm{R}}_{nn}\}, \tag{22}$$

where $\gamma^{\mathrm{L}}_{11} = [\mathbf{\Gamma}^{\mathrm{L}}]_{11}$ and $\gamma^{\mathrm{R}}_{nn} = [\mathbf{\Gamma}^{\mathrm{R}}]_{nn}$. Thus we introduce the notation $[\cdot]_{ij}$ which delivers the (i,j)-block, with respect to $\mathbf{A}$'s block structure, of the bracketed expression. The main task is to find $\mathbf{g}_{n1}$. From Eq. (16) it is seen that the expression for this matrix is:

$$\mathbf{g}_{n1} = \mathbf{g}_{nn}\mathbf{c}^{\mathrm{L}}_{n-1}\mathbf{c}^{\mathrm{L}}_{n-2}\cdots\mathbf{c}^{\mathrm{L}}_{2}\mathbf{c}^{\mathrm{L}}_{1}, \tag{23}$$

and we see that the only factors $\mathbf{c}^{\mathrm{L}}_{i}$ involved are all those computed in a downwards block Gaussian elimination sweep. The matrix $\mathbf{g}_{nn}$ in Eq. (23) can be obtained by considering the nth block from Eq. (17):

$$\mathbf{g}_{nn} = (-\mathbf{a}_{nn} + \mathbf{d}^{\mathrm{L}}_{nn} + \mathbf{d}^{\mathrm{R}}_{nn})^{-1} = (\mathbf{d}^{\mathrm{L}}_{nn})^{-1}, \tag{24}$$

since $\mathbf{d}^{\mathrm{R}}_{nn} = \mathbf{a}_{nn}$. This holds similarly for the first row of the Green's function matrix. From this, it is seen that the first and last diagonal blocks of the Green's function matrix correspond to the final blocks of upwards and downwards sweeps of block Gaussian elimination, respectively, in the following manner:

$$\mathbf{g}_{11} = (\mathbf{d}^{\mathrm{R}}_{11})^{-1} \quad \text{and} \quad \mathbf{g}_{nn} = (\mathbf{d}^{\mathrm{L}}_{nn})^{-1}. \tag{25}$$

4.2. Overlap method

A new method that seeks to compute the transmission much like the baseline coupling method, however via a different part of the Green's function matrix, is now introduced.

Here, the idea is again based on the transmission formula Eq. (20), however the matrices dealt with change from being a coupling between the leads to that of a coupling between two adjacent blocks somewhere in the center of the system. This corresponds to centering calculations around a diagonal block of $\mathbf{A}$. This will require us to calculate the Green's function for the kth block of interest, $\mathbf{g}_{kk}$.

The name of the method arises from the fact that calculation of a diagonal block involves a *sweep* of block Gauss elimination from both the upper left and lower right of $\mathbf{A}$ which will *overlap* on the block of interest.

The motivation behind this approach is to avoid the work in having to calculate an off-diagonal block of the Green's function matrix after a series of block Gaussian elimination sweeps. This amounts to $n-1$ matrix multiplications. In the new method, overhead will arise due to calculations involving the self-energies $\mathbf{\Sigma}^{\mathrm{L}}$ and $\mathbf{\Sigma}^{\mathrm{R}}$, and the corresponding matrices $\mathbf{\Gamma}^{\mathrm{L}}$ and $\mathbf{\Gamma}^{\mathrm{R}}$. However, these computations are less expensive matrix addition operations, and they are negligible with increasing number of matrix blocks and block sizes.

As we shall demonstrate below, it is advantageous to choose k corresponding to the smallest diagonal block inside the block tridiagonal matrix $\mathbf{A}$. Although this approach involves some additional computations with the self-energy matrices and their corresponding coupling matrices, this overhead is acceptable due to the savings involved in the cheaper matrix computations for the overlap method.

Choosing an arbitrary kth diagonal block, the transmission is given in the following expression, derived in the appendix:

$$t = \mathrm{Tr}\{\mathbf{g}_{kk}[\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow k}]_{kk}\mathbf{g}^{\dagger}_{kk}[\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow k}]_{kk}\}, \tag{26}$$

where the new self-energy related terms are given by Eqs. (48) and (49) in the Appendix. Using the nonzero structure of the respective self-energies $\sum^{\mathrm{L}}$ and $\sum^{\mathrm{R}}$ we obtain the simpler relations

$$\left[\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow 1}\right]_{11} = \gamma^{\mathrm{L}}_{11}, \qquad \left[\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow 1}\right]_{11} = \imath((\mathbf{d}^{\mathrm{R}}_{11})^{\dagger} - \mathbf{d}^{\mathrm{R}}_{11}) \tag{27}$$

$$\left[\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow n}\right]_{nn} = \gamma^{\mathrm{R}}_{nn}, \qquad \left[\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow n}\right]_{nn} = \imath((\mathbf{d}^{\mathrm{L}}_{nn})^{\dagger} - \mathbf{d}^{\mathrm{L}}_{nn}) \tag{28}$$

and for $k = 2,\ldots,n-1$

$$[\boldsymbol{\Gamma}^{\mathrm{L}}_{\downarrow k}]_{kk} = \imath((\mathbf{d}^{\mathrm{L}}_{kk})^{\dagger} - \mathbf{d}^{\mathrm{L}}_{kk}) - \gamma^{\mathrm{R}}_{kk}, \quad [\boldsymbol{\Gamma}^{\mathrm{R}}_{\uparrow k}]_{kk} = \imath((\mathbf{d}^{\mathrm{R}}_{kk})^{\dagger} - \mathbf{d}^{\mathrm{R}}_{kk}) - \gamma^{\mathrm{L}}_{kk}. \tag{29}$$

5. Benchmark results

The methods introduced here were implemented in C++ within Atomistix's Atomistix ToolKit, and computing times were obtained for calculating the transmission for 10 different energies ε for several different systems. These systems have been taken from the literature, and an overview of selected examples is presented in Table 1.

5.1. Operation count

In order to determine which transmission method may be algorithmically more efficient, the quantity of matrix factorizations, multiplications and additions related to the three different methods available is recorded in Table 2.

In Table 3 operation counts for the calculations of the full inverse of **A** as well as calculation of only the block tridiagonal part of the inverse is included. This is done for both a Gauss elimination (GE) algorithm, as well as the new method presented in this paper.

The block tridiagonal part of the inverse is of interest for further calculations carried out in Density Functional Theory (DFT) via the Green's function formalism, and results for the full inverse are included in order to show how the new method in this paper, though suited for the block tridiagonal calculation, is ill-suited to calculate the entire inverse, compared to traditional methods.

Looking at operation counts in Table 2 on obtaining various parts of the Green's function matrix **G**, it is seen that all choices require n LU factorizations, where n is the number of diagonal blocks in **A**.

Table 1
An overview of the test examples examined in this paper

Example systems

System	Article	Order	n	Block order
`Al100+C7`	[15]	444	5	128, 72, 16, 100, 128
`AlLead+C7`	[15]	296	5	72, 72, 20, 60, 72
`Au111–AR`	[17]	1295	10	243, 162, 66, 79, 69, 84, 62, 62, 225, 243
`Au111–TW`	[16]	1155	8	243, 162, 62, 70, 53, 70, 252, 243
`Au111–DTB`	[18]	928	5	243, 162, 88, 198, 243
`Fe–MgO–Fe`	[13,14]	228	5	54, 45, 30, 45, 54
`nanotube4_4`	–	576	4	128, 128, 192, 128

For each example the original paper related to the system, the dimension of the overall matrix **A**, the number of diagonal blocks n, and finally the size of each of the diagonal blocks, from the upper left of **A** down to the lower right is listed.

Table 2
This table illustrates the amount of basic operations performed in calculating different blocks of **G** via either block Gauss elimination (GE), the coupling method or overlap method

Green's function sub-block operation count

Block	Method	LU-factorizations	Multiplications	Additions
$\mathbf{g}_{n1}$	GE	n	$3(n-1)$	$n-1$
$\mathbf{g}_{nn}$	GE	n	$2n-1$	$n-1$
$\mathbf{g}_{kk}$	GE	n	$4n-2k-1$	$2n-k-1$
$\mathbf{g}_{n1}$	Coupling	n	$3(n-1)$	$n-1$
$\mathbf{g}_{nn}$	Overlap	n	$2n-1$	$n-1$
$\mathbf{g}_{kk}$	Overlap	n	$2n-1$	$n+1$

The third, fourth and fifth columns refer to the basic matrix operations of LU-factorization, multiplication and addition. The term n is the total amount of diagonal blocks in **A**, and k indicates which diagonal block in the Green's function matrix **G** is used for transmission calculations.

Table 3
This table illustrates the amount of basic operations performed in calculating either the full inverse **G** of **A**, or only the block tridiagonal part of it, using different methods

Green's function operation count

Calculation	Method	LU-factorizations	Multiplications	Additions
Full inv	GE	n	$2n^2 + n - 2$	$\frac{1}{2}(n^2 + n - 2)$
Trid inv	GE	n	$\frac{1}{2}(3n^2 + 5n - 6)$	$\frac{1}{2}(n^2 + n - 2)$
Full inv	Paper	$3n - 2$	$n^2 + 4n - 4$	$4n - 6$
Trid inv	Paper	$3n - 2$	$7n - 6$	$4n - 6$

The methods employed are block Gauss Elimination (*GE*), and the new method incorporating forward and backward Gaussian elimination sweeps (*paper*), as presented in Eq. (16). The third, fourth and fifth columns refer to the basic matrix operations of LU–factorization, multiplication and addition. The term *full inv* refers to calculating the full inverse, while *trid inv* refers to obtaining only the block tridiagonal part of the inverse. The term n is the total amount of diagonal blocks in **A**.

In obtaining $\mathbf{g}_{n1}$, block Gauss elimination and the coupling method both require the same amount of operations to complete, and there is no advantage either way. Again, in obtaining the lower diagonal block $\mathbf{g}_{nn}$, both block Gauss elimination and the overlap method require the same amount of matrix–matrix calculations.

The advantage of the overlap method over block Gauss elimination occurs when a central diagonal block $\mathbf{g}_{kk}$ is required. Here, only two more matrix–matrix additions over the overlap method for $\mathbf{g}_{nn}$ is needed, while for block Gauss elimination, a series of matrix–matrix multiplies and additions add up in order to back–solve up towards the desired diagonal block. Thus the overlap method is better suited for determining diagonal blocks than block Gauss elimination.

Looking at which block of the matrix **G** is cheapest to compute on the basis of Table 2, one would apparently choose $\mathbf{g}_{nn}$. This, however, may not be the case since the table does not take into account differing block sizes among the different sub-blocks in **A** and **G**. These differing sizes can lead to substantial changes in costs regarding the basic operations of LU-factorization, matrix multiplication and matrix addition in the table. The speedup results presented later in Section 5.2 and Table 4 will verify this.

With regard to the cost of the basic operations on a matrix block of order n_i, then the amount of work for each LU-factorization, multiplication and addition is on the order of $2/3n_i^3$, $2n_i^3$ and $2n_i^2$, respectively.

5.1.1. Transmission calculation

To finally calculate transmission after successfully obtaining a sub-block of **G**, the Fisher–Lee relation (cf. Eq. (20)) is invoked, and thus three matrix–matrix multiplications are incurred, as well as a matrix trace operation. However, the significant factor here among the different methods reviewed is that the final matrix block dimensions in the Fisher–Lee relation may be different. Typically, due to the topology of the two-probe system, the central region, and thus the kth diagonal block $\mathbf{g}_{kk}$, will be of smaller size than the corner blocks $\mathbf{g}_{nn}$ or $\mathbf{g}_{n1}$. Thus a significant prefactor cost in execution time can be saved by selecting the transmission method centered around the smallest Green's function diagonal matrix block.

Table 4
This table illustrates the speedup achieved by using the new methods centered around diagonal blocks, relative to the baseline coupling method using the off-diagonal block $\mathbf{g}_{n1}$

Speedup measurements

System	Coupling – $\mathbf{g}_{n1}$	Overlap – $\mathbf{g}_{nn}$	Overlap – $\mathbf{g}_{kk}$
`Al100+C7`	1.0000	1.2099	2.6557
`AlLead+C7`	1.0000	1.1916	2.0092
`Au111–AR`	1.0000	1.4211	3.2121
`Au111–TW`	1.0000	1.3721	2.8994
`Au111–DTB`	1.0000	1.3675	3.2537
`Fe–MgO–Fe`	1.0000	1.3064	1.8001
`nanotube4_4`	1.0000	1.2261	1.2477

The expression $\mathbf{g}_{n1}$ refers to the coupling method, while $\mathbf{g}_{nn}$ and $\mathbf{g}_{kk}$ refer to the overlap method performed on the nth and smallest diagonal block, respectively. The overlap methods are always faster, and in particular those centred on the smallest, kth, diagonal block.

Some overhead arises in choosing a central diagonal block in the shape of recalculating new matrices $[\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow k}]_{kk}$ and $[\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow k}]_{kk}$ for the transmission function Eq. (26) via Eqs. (27)–(29), but as these operations are cheaper matrix–matrix addition operations on small matrices, this overhead is offset by the gains in being able to employ smaller matrices in the more expensive matrix–matrix multiplication operations in the Fisher–Lee relation in Eq. (26).

5.1.2. Full inversion

With regard to determining the full inverse **G** from **A**, it is seen in Table 3 how block Gauss elimination excels over the method in this paper in terms of costly LU factorizations. Although Gauss elimination has about twice the number of matrix multiplies than the new method, Gauss elimination is still preferable when taking into account that it only requires about a third LU factorizations compared to the new method. Thus the new method is not suited for determining the full matrix **G**.

However, when requiring only the tridiagonal part of the inverse, as is the case for some DFT applications, the new method is a better choice since it only requires on the order of n matrix–matrix multiplications, while block Gauss elimination still requires on the order of n^2 matrix–matrix multiplications.

5.2. Speedup results

For an overview of the speedup of the new methods relative to the baseline coupling method, see Table 4. Overall, speedup improves in every case when moving from the coupling method to the overlap method. This is not surprising, seeing how the main difference between these two methods, operation count–wise, is the lack of extra matrix multiplications in order to obtain an off-diagonal Green's function matrix block. Eliminating this task will always lead to a faster method.

Performing calculations using the smallest diagonal block k over the first or nth block can also yield significant improvements in execution time, depending on the topology of the two-probe system, and the subsequent block structure in **A**. The difference here is that it is no longer possible to 'recycle' one of the self-energy terms that is assumed to be available from the outset, as well as different block size between $\mathbf{g}_{nn}$ and $\mathbf{g}_{kk}$. Thus in seeking a smaller diagonal matrix block to work with, appropriate self-energy terms must be determined once again, and this leads to extra overhead.

However, it may pay off to select some central diagonal block over a corner diagonal block in order to calculate transmission. This comes in the form of being able to work with smaller matrices, and thus matrix operation costs decrease. Crucially, matrix sizes may decrease such that memory requirements for matrix operations can be fulfilled by lower level hardware caches, leading to significant speedup in execution time. This effect is visible in Table 5, where significant speedup is achieved in the matrix–matrix operations involved in the Fisher–Lee calculation.

Table 5
This table illustrates the speedup in the calculation of solely the Fisher–Lee relation (see Eq. (26)) achieved by using the new methods centered around diagonal blocks, relative to the baseline coupling method using the off-diagonal block $\mathbf{g}_{n1}$

Speedup measurements – Fisher–Lee				
System	Coupling – $\mathbf{g}_{n1}$	Overlap – $\mathbf{g}_{nn}$	Overlap – $\mathbf{g}_{kk}$	Theoretical – $\frac{n^3}{m^3}$
`Al100+C7`	1.0000	1.0567	548.4500	512.000
`AlLead+C7`	1.0000	0.9546	47.4516	46.656
`Au111–AR`	1.0000	1.2654	170.6912	60.207
`Au111–TW`	1.0000	1.2788	275.6198	96.381
`Au111–DTB`	1.0000	1.2716	59.8502	21.056
`Fe–MgO–Fe`	1.0000	1.3186	7.1354	5.832
`nanotube4_4`	1.0000	0.9940	1.0178	1.000

The expression $\mathbf{g}_{n1}$ refers to the coupling method, while $\mathbf{g}_{nn}$ and $\mathbf{g}_{kk}$ refer to the overlap method performed on the nth and smallest diagonal block, respectively. The final column indicates the theoretical speedup based on the $\mathcal{O}(n^3)$ cost of evaluating Eq. (26). The reason for better speedup over theoretical prediction is due to improved cache usage by the smaller matrices dealt with when using $\mathbf{g}_{kk}$.

Furthermore, as will be explored in Section 6 concerning transmission accuracy, depending on the system, central matrix blocks may be less prone to perturbation from inaccurately calculated electrode surface Green's function matrices. This, however, varies from system to system, as well as incoming electron wave energies ε.

6. Transmission accuracy

It has been shown that any block of the Green's function matrix can be used in order to calculate transmission and a new strategy employing diagonal blocks of $\mathbf{G}$ was developed. The question now is which part of $\mathbf{G}$ might be used in order to achieve best accuracy in determining transmission. This section suggests that an investigation of the accuracy achieved for a given block may lead to informed choices. The problem of the selection of which matrix block is best concerning accuracy comes from the fact that in practice the self-energies of the electrodes, $\boldsymbol{\sigma}^{\mathrm{L}}_{11}$ and $\boldsymbol{\sigma}^{\mathrm{R}}_{nn}$, are not computed exactly. This is because in the Green's function formalism approach, the surface Green's function matrices for the electrodes (and hence their corresponding self-energies) are typically determined through an iterative procedure [29] that only converges to the correct retarded Green's function matrix when a small positive imaginary perturbation is applied. This means that transmissions are calculated for a slightly perturbed matrix $\widetilde{\mathbf{A}}$, where the corner blocks $\mathbf{a}_{11}$ and $\mathbf{a}_{nn}$ are perturbed to some degree through the inexact self-energies.

The matrix $\mathbf{A}$ here will denote the case when no imaginary perturbation is used and this can be done by employing a different manner to converge the surface Green's function matrices, such as a wave function matching [30–32] approach. To investigate how this imaginary perturbation ultimately affects the Green's function matrix that transmissions are calculated with, the inverses of an unperturbed case and a perturbed case are compared. The perturbation on $\mathbf{A}$ is described as the added matrix $\mathbf{P}$, defined as zero everywhere, except the corner blocks $\mathbf{p}_{11}$ and $\mathbf{p}_{nn}$, that correspond to the corner blocks $\mathbf{a}_{11}$ and $\mathbf{a}_{nn}$, both in size and location.

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{P}, \quad \text{where} \quad \mathbf{P} = \left(\begin{array}{c|c|c} \mathbf{p}_{11} & & \\ \hline & & \\ \hline & & \mathbf{p}_{nn} \end{array}\right) \tag{30}$$

The perturbation matrix, as seen in Eq. (30), is divided into 9 blocks, where the empty space denotes areas with elements equal to zero. In a similar manner, the inverse $\mathbf{G} = \mathbf{A}^{-1}$ is subdivided into the same block sizes.

$$\mathbf{A} = \left(\begin{array}{c|cccc|c} \mathbf{a}_{11} & \mathbf{a}_{12} & & & & \\ \hline \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & & \\ & & \ddots & \ddots & \ddots & \\ & & & & & \mathbf{a}_{n-1,n} \\ \hline & & & & \mathbf{a}_{n,n-1} & \mathbf{a}_{nn} \end{array}\right), \quad \mathbf{G} = \left(\begin{array}{c|ccc|c} \mathbf{g}_{11} & \bullet & \bullet & \bullet & \mathbf{g}_{1n} \\ \hline \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \mathbf{g}_{n1} & \bullet & \bullet & \bullet & \mathbf{g}_{nn} \end{array}\right) \tag{31}$$

To investigate the effect of the perturbation $\mathbf{P}$ the derivation of $\widetilde{\mathbf{G}} = \widetilde{\mathbf{A}}^{-1}$ is carried out:

$$\widetilde{\mathbf{G}} = [\mathbf{A}(\mathbf{I} + \mathbf{GP})]^{-1} = (\mathbf{I} + \mathbf{GP})^{-1}\mathbf{G}. \tag{32}$$

If the perturbation is assumed to be small, such that the spectral radius satisfies $\rho(\mathbf{GP}) < 1$, then the first inverse term can be expressed via a geometric series.

$$\widetilde{\mathbf{G}} = (\mathbf{I} + \mathbf{GP})^{-1}\mathbf{G} = \mathbf{G} - \mathbf{GPG} + \mathbf{GPGPG} - \cdots \tag{33}$$

Thus it can be seen that the difference in the perturbed and unperturbed inverses should be dominated by the term **GPG**. If **G** is subdivided into row and column blocks, as follows, it will be possible to proceed and derive a relatively compact expression for the structure of this first order correction term.

$$\mathbf{G} = \left(\begin{array}{c|c|c} \mathbf{g}_{11} & \cdots & \mathbf{g}_{1n} \\ \hline \vdots & \ddots & \vdots \\ \hline \mathbf{g}_{n1} & \cdots & \mathbf{g}_{nn} \end{array}\right) = \left(\begin{array}{c|c|c} \mathbf{b}_1 & \cdots & \mathbf{b}_n \end{array}\right) = \left(\begin{array}{c} \mathbf{c}_1 \\ \hline \vdots \\ \hline \mathbf{c}_n \end{array}\right) \tag{34}$$

such that

$$\mathbf{b}_1 = \begin{pmatrix} \mathbf{g}_{11} \\ \vdots \\ \mathbf{g}_{n1} \end{pmatrix}, \quad \mathbf{b}_n = \begin{pmatrix} \mathbf{g}_{n1} \\ \vdots \\ \mathbf{g}_{nn} \end{pmatrix} \tag{35}$$

and

$$\mathbf{c}_1 = (\mathbf{g}_{11} \quad \cdots \quad \mathbf{g}_{1n}), \quad \mathbf{c}_n = (\mathbf{g}_{n1} \quad \cdots \quad \mathbf{g}_{nn}). \tag{36}$$

With this notation, the first order perturbation term is written as follows:

$$\mathbf{GPG} = \mathbf{b}_1\mathbf{p}_{11}\mathbf{c}_1 + \mathbf{b}_n\mathbf{p}_{nn}\mathbf{c}_n. \tag{37}$$

It can be seen how the outer-product form of this expression will yield a dense matrix **GPG**, since **G** can generally be assumed to be dense. This indicates that the correction term's effect will depend directly on the full structure of **G**, and thus no prediction can be made about the effect of the perturbation on **G**, without calculating **G** itself.

We look at the first order perturbation at block (i,j):

$$[\mathbf{GPG}]_{ij} = [\mathbf{b}_1\mathbf{p}_{11}\mathbf{c}_1 + \mathbf{b}_n\mathbf{p}_{nn}\mathbf{c}_n]_{ij} = \mathbf{g}_{i1}\mathbf{p}_{11}\mathbf{g}_{1j} + \mathbf{g}_{in}\mathbf{p}_{nn}\mathbf{g}_{nj},$$

where the element $\mathbf{g}_{in}$ describes the amplitude of an electron propagating from site i to site n in the system. For most systems, this will decay as a function of the distance between orbitals at sites i and n, and thus the error should be smallest for Green's function blocks in the center of the cell, i.e., as far as possible from the electrodes. Thus we can expect choosing central blocks in **G** should lead to more accurate transmission calculations for most systems.

6.1. Numerical example with random perturbation

The effect of a perturbation of the electrode's surface Green's function matrices on the Green's function **G** itself is here illustrated by a numerical example. The Hamiltonian matrix **H** and overlap matrix **S** associated with Au111–AR is taken, and the matrix to be inverted is constructed as

$$\mathbf{A} = \mathbf{H} - \varepsilon\mathbf{S}, \quad \text{where } \varepsilon = 1.0. \tag{38}$$

The corner blocks of **A**, namely $\mathbf{a}_{11}$ and $\mathbf{a}_{nn}$, are then perturbed with matrices $\mathbf{p}_{11}$ and $\mathbf{p}_{nn}$. The elements of $\mathbf{p}_{11}$ and $\mathbf{p}_{nn}$ are computed as:

$$\mathbf{p}_{11} \leftarrow p_{ij} = \alpha_{ij}a_{ij}, \quad \text{where } a_{ij} \in \mathbf{a}_{11}, \text{ and} \tag{39}$$

$$\mathbf{p}_{nn} \leftarrow p_{kl} = \alpha_{kl}a_{kl}, \quad \text{where } a_{kl} \in \mathbf{a}_{nn}. \tag{40}$$

where the factors α_{ij} and α_{kl} are normally distributed with zero mean and standard deviation 10^{-5}.

Fig. 3 shows the results of the average difference of 100 perturbed inversions $\widetilde{\mathbf{G}}$ compared to **G**. From this figure, it is seen that for this particular choice of system (**H** and **S**) and energy (ε), the perturbation from the iterated self-energies cause the inverse to be most inaccurate at the corner diagonal blocks. Thus, choosing the

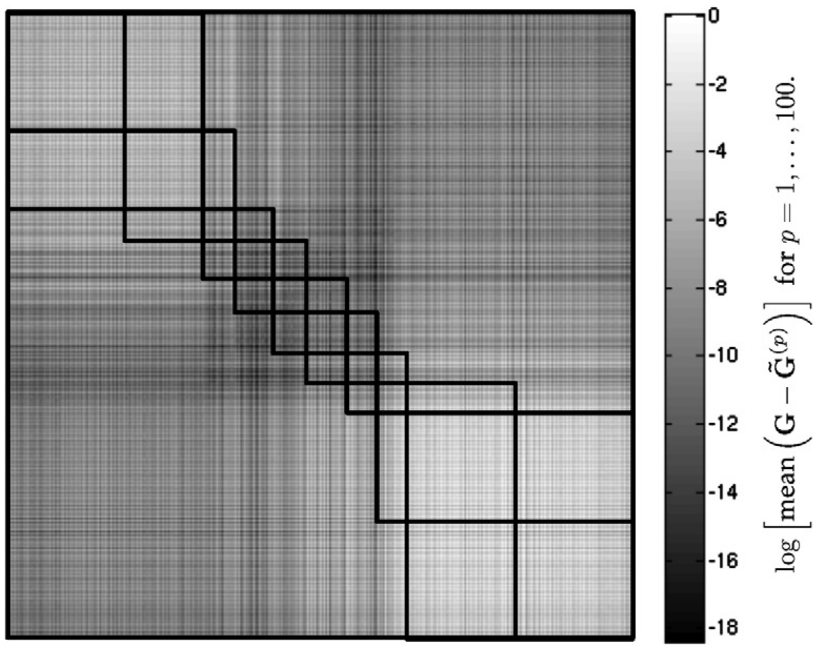


Fig. 3. The figure above illustrates the average element-wise difference expressed as $\log[\mathrm{mean}(\mathbf{G} - \widetilde{\mathbf{G}}^{(p)})]$ for $p = 1, \ldots, 100$. The matrix $\mathbf{G}$ corresponds to the Au111–AR example [17]. Element-wise differences range from about the same order down to about 18 orders of magnitude smaller. The dark lines outline the original block tridiagonal structure of the original matrix $\mathbf{A}$. The logarithm employed is the base 10 logarithm. In this particular example for choice of electron energy ε and $\mathbf{A}$, the diagonal blocks in the center of the matrix suffer least in terms of accuracy.

overlap method as the transmission calculation method would be on average best served by choosing a block towards the center of the matrix, where the perturbation has the least effect. This choice is further motivated by the fact that the center blocks typically are of smaller size, and matrix operations would be faster than operations with the corner diagonal blocks.

A problem with this analogy lies in the fact that one can not predict which Green's function matrix block would provide more accurate transmission results (see Eq. (37)), without calculating the Green's function matrix in the first place. This lends prediction to be prohibitive in general, when computing transmissions. The best choice of action is then relying on the usual behavior of most two-probe systems as well as choosing the fastest calculation method, leading us to pick a diagonal block towards the center of the system, which are typically the least affected by the electrodes as well as the smallest in size.

7. Conclusion

This paper developed and introduced a new, faster method of calculating transmission for two-probe systems by using diagonal block matrices from the Green's function matrix, $\mathbf{g}_{ii}$, rather than the coupling method found extensively in the literature that uses the corner off-diagonal block $\mathbf{g}_{n1}$.

This is done by developing a method for calculating any block matrix from the Green's function matrix $\mathbf{G}$ based on a series of Gauss eliminations carried out on the original matrix $\mathbf{A}$.

To calculate transmission via a diagonal block of the Green's function matrix $\mathbf{G}$, upwards and downwards block Gaussian elimination is performed that terminates overlapping over $\mathbf{a}_{kk}$, and $\mathbf{g}_{kk}$ is calculated (cf. Eq. (17)).

Furthermore, the related coupling matrices (usually obtained via self-energy) used in the transmission formula Eq. (26) are calculated via Eqs. (27)–(29), for the new, extended electrodes. This approach dispenses with the need of a series of matrix–matrix multiplications compared to the coupling method (cf. Eq. (23)) in exchange for cheaper matrix–matrix addition operations.

Execution time measurements indicated that centering transmission calculations on the Green's function matrix's diagonal blocks was preferable, in that a series of matrix–matrix multiplications would be saved as well as centering on smaller diagonal matrix blocks offset the cost of re-calculating self-energy matrices. Furthermore, the ability to choose smaller block matrices lends itself to the possibility of far better cache usage, and hence greater performance gains.

Perturbation analysis revealed that it is not possible to determine the effect of perturbation in the electrode self-energy matrices on the accuracy of the Green's function, without explicitly calculating the Green's function matrix. This eliminates the ability to predict which Green's function matrix block would be an ideal choice for the calculation of a two-probe system's transmission with respect to accuracy. However, due to the behavior of most two-probe systems, a central diagonal block choice is expected to yield more accurate results.

Acknowledgments

This work was supported by Grant No. 2106-04-0017, "Parallel Algorithms for Computational Nano-Science", under the NABIIT program from the Danish Council for Strategic Research.

Appendix. Derivation of Eq. (26) for the Transmission

We commence with the expression in Eq. (1). As shown in (e.g., Golub and Van Loan [33]) Section 3.2.1, we can represent a Gauss-elimination step as a matrix multiplication with a "Gauss transformation". The same is true for the block Gauss-elimination steps we use here, and thus we express a series of downwards Gauss-eliminations that terminate on row k by $\mathbf{E}_{\downarrow k}$. Similarly, a series of upwards Gauss-eliminations terminating on row k is denoted by $\mathbf{E}_{\uparrow k}$. We then write the combination of Gauss-elimination sweeps that produce a matrix $\mathbf{Z}_k$ as follows:

$$\mathbf{Z}_k = \mathbf{E}_{\downarrow k}\mathbf{A}\mathbf{E}_{\uparrow k}. \tag{41}$$

Due to the structure of $\mathbf{A}$, the matrix $\mathbf{Z}_k$ is block diagonal as shown in Fig. 4. Given $\mathbf{Z}_k$, we can write the Green's function matrix as

$$\mathbf{G} = \mathbf{A}^{-1} = \mathbf{E}_{\uparrow k}\mathbf{Z}_k^{-1}\mathbf{E}_{\downarrow k}. \tag{42}$$

We can then insert this expression into the Fisher–Lee relation from Eq. (20), to obtain

$$\begin{aligned} t &= \mathrm{Tr}\{(\mathbf{E}_{\uparrow k}\mathbf{Z}_k^{-1}\mathbf{E}_{\downarrow k})\mathbf{\Gamma}^{\mathrm{L}}(\mathbf{E}_{\uparrow k}\mathbf{Z}_k^{-1}\mathbf{E}_{\downarrow k})^{\dagger}\mathbf{\Gamma}^{\mathrm{R}}\} = \mathrm{Tr}\{\mathbf{E}_{\uparrow k}\mathbf{Z}_k^{-1}\mathbf{E}_{\downarrow k}\mathbf{\Gamma}^{\mathrm{L}}\mathbf{E}_{\downarrow k}^{\dagger}(\mathbf{Z}_k^{-1})^{\dagger}\mathbf{E}_{\uparrow k}^{\dagger}\mathbf{\Gamma}^{\mathrm{R}}\} \\ &= \mathrm{Tr}\{\mathbf{Z}_k^{-1}\mathbf{E}_{\downarrow k}\mathbf{\Gamma}^{\mathrm{L}}\mathbf{E}_{\downarrow k}^{\dagger}(\mathbf{Z}_k^{-1})^{\dagger}\mathbf{E}_{\uparrow k}^{\dagger}\mathbf{\Gamma}^{\mathrm{R}}\mathbf{E}_{\uparrow k}\} = \mathrm{Tr}\{\mathbf{Z}_k^{-1}\mathbf{\Gamma}_{\downarrow k}^{\mathrm{L}}(\mathbf{Z}_k^{-1})^{\dagger}\mathbf{\Gamma}_{\uparrow k}^{\mathrm{R}}\} \end{aligned} \tag{43}$$

where we have introduced

$$\mathbf{\Gamma}_{\downarrow k}^{\mathrm{L}} = \mathbf{E}_{\downarrow k}\mathbf{\Gamma}^{\mathrm{L}}\mathbf{E}_{\downarrow k}^{\dagger} \quad \text{and} \quad \mathbf{\Gamma}_{\uparrow k}^{\mathrm{R}} = \mathbf{E}_{\uparrow k}^{\dagger}\mathbf{\Gamma}^{\mathrm{R}}\mathbf{E}_{\uparrow k}. \tag{44}$$

To derive Eq. (43) we used that the trace is invariant under matrix commutation [34].

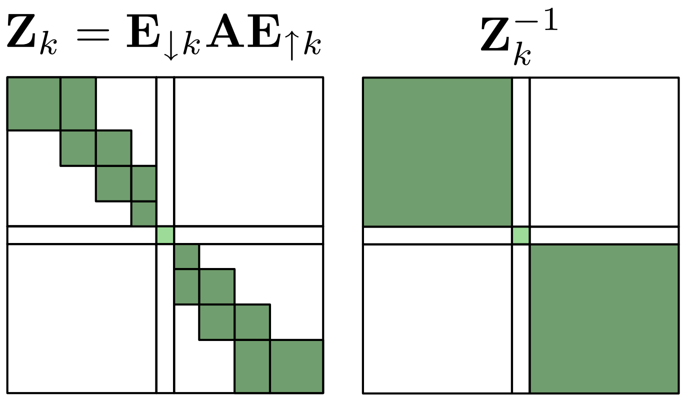

$\mathbf{Z}_k = \mathbf{E}_{\downarrow k}\mathbf{A}\mathbf{E}_{\uparrow k}$
$\mathbf{Z}_k^{-1}$

Fig. 4. The zero/nonzero structure of $\mathbf{Z}_k$ and $\mathbf{Z}_k^{-1}$.

Eq. (43) can be further simplified. First note that both $\mathbf{Z}_k$ and $\mathbf{Z}_k^{-1}$ have the special zero/nonzero structure shown in Fig. 4. Next, note that $\mathbf{\Gamma}^{\mathrm{L}}$ has nonzero elements in its (1,1)-block only, and hence the nonzeros in $\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow k}$ are confined to upper left blocks, as shown in Fig. 5. Similarly, the nonzeros of $\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow k}$ are confined to the bottom right blocks. Using the zero/nonzero structure of these matrices, it follows from the derivation illustrated in Fig. 6 that:

$$t = \mathrm{Tr}\{\mathbf{Z}_k^{-1}\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow k}(\mathbf{Z}_k^{-1})^{\dagger}\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow k}\} = \mathrm{Tr}\{[\mathbf{Z}_k^{-1}]_{kk}[\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow k}]_{kk}[\mathbf{Z}_k^{-1}]^{\dagger}_{kk}[\mathbf{\Gamma}^{\mathrm{R}}_{\uparrow k}]_{kk}\}. \tag{45}$$

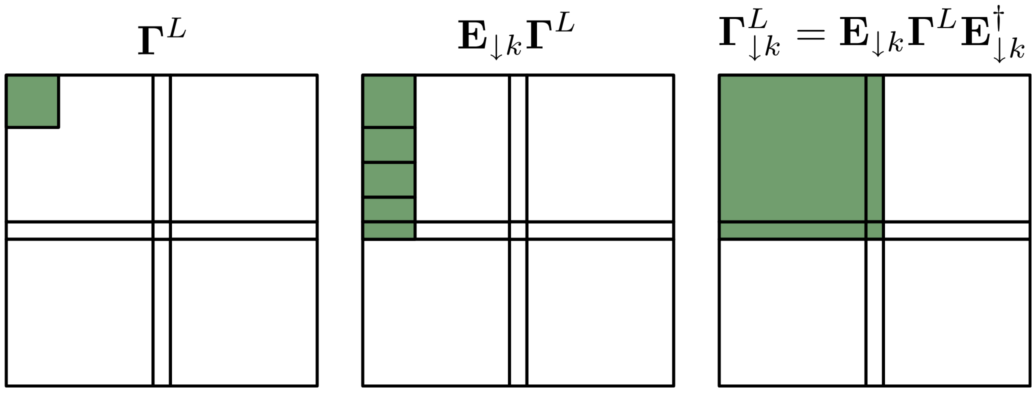

$\mathbf{\Gamma}^L$
$\mathbf{E}_{\downarrow k}\mathbf{\Gamma}^L$
$\mathbf{\Gamma}^L_{\downarrow k} = \mathbf{E}_{\downarrow k}\mathbf{\Gamma}^L\mathbf{E}^{\dagger}_{\downarrow k}$

Fig. 5. The zero/nonzero structure of $\mathbf{\Gamma}^{\mathrm{L}}$ and $\mathbf{\Gamma}^{\mathrm{L}}_{\downarrow k}$.

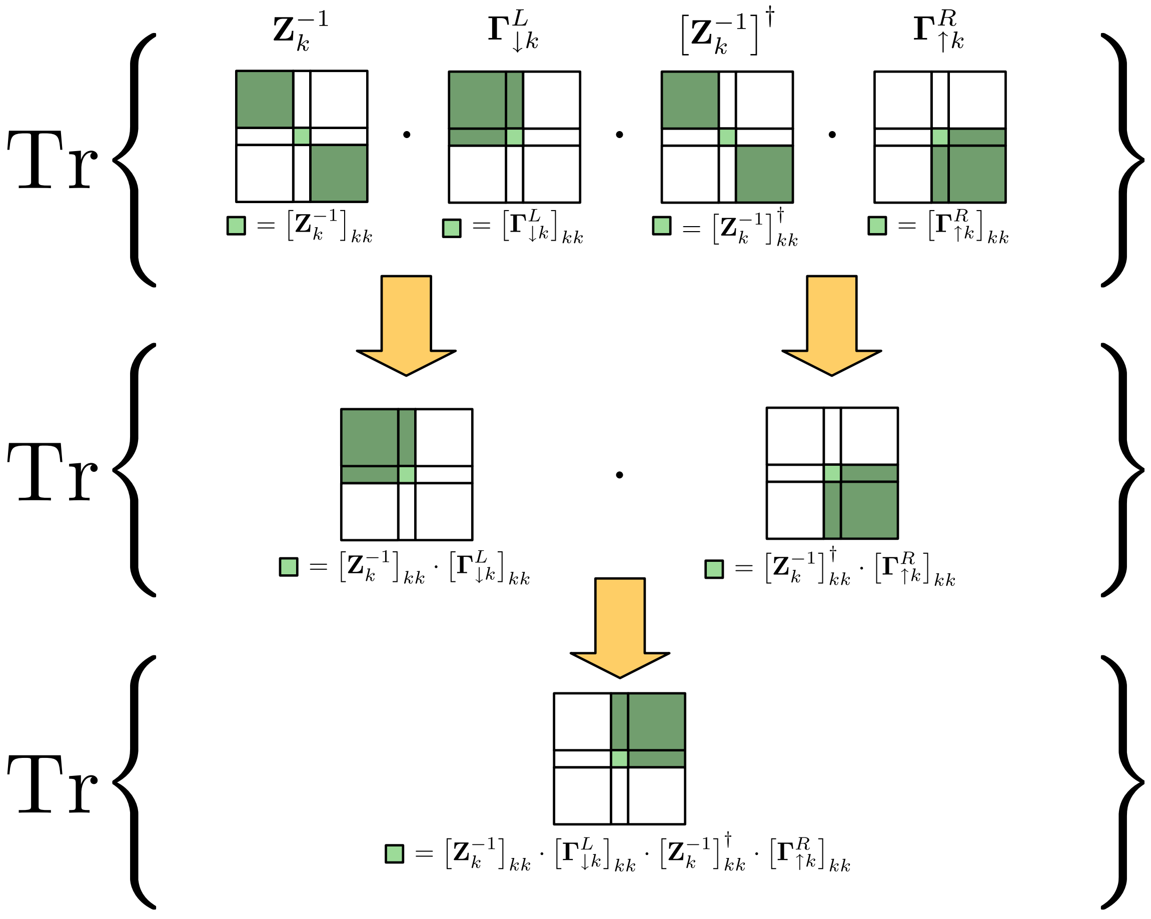

Tr
$\mathbf{Z}_k^{-1}$
$\mathbf{\Gamma}^L_{\downarrow k}$
$[\mathbf{Z}_k^{-1}]^{\dagger}$
$\mathbf{\Gamma}^R_{\uparrow k}$
🟩 $= [\mathbf{Z}_k^{-1}]_{kk}$
🟩 $= [\mathbf{\Gamma}^L_{\downarrow k}]_{kk}$
🟩 $= [\mathbf{Z}_k^{-1}]^{\dagger}_{kk}$
🟩 $= [\mathbf{\Gamma}^R_{\uparrow k}]_{kk}$
Tr
🟩 $= [\mathbf{Z}_k^{-1}]_{kk} \cdot [\mathbf{\Gamma}^L_{\downarrow k}]_{kk}$
🟩 $= [\mathbf{Z}_k^{-1}]^{\dagger}_{kk} \cdot [\mathbf{\Gamma}^R_{\uparrow k}]_{kk}$
Tr
🟩 $= [\mathbf{Z}_k^{-1}]_{kk} \cdot [\mathbf{\Gamma}^L_{\downarrow k}]_{kk} \cdot [\mathbf{Z}_k^{-1}]^{\dagger}_{kk} \cdot [\mathbf{\Gamma}^R_{\uparrow k}]_{kk}$

Fig. 6. Illustration of the derivation of Eq. (45) using the zero/nonzero structure of Figs. 4 and 5.

Hence, we require only the kth diagonal block of $\mathbf{Z}_k^{-1}$, and we note that this corresponds to the kth diagonal block of the Green's function matrix $\mathbf{G}$ via Eq. (17). Thus $[\mathbf{Z}_k^{-1}]_{kk} = \mathbf{g}_{kk}$, and $[\mathbf{Z}_k^{-1}]_{kk}^\dagger = \mathbf{g}_{kk}^\dagger$.

Next we consider $[\mathbf{\Gamma}_{\downarrow k}^{\mathrm{L}}]_{kk}$ and $[\mathbf{\Gamma}_{\uparrow k}^{\mathrm{R}}]_{kk}$. By means of Eq. (1) we can obtain the expression of a self-energy, e.g., $\mathbf{\Sigma}^{\mathrm{L}}$, and via Eq. (21) we now determine our desired matrix for the transmission calculation:

$$\begin{aligned}[\mathbf{\Gamma}_{\downarrow k}^{\mathrm{L}}]_{kk} &= [\mathbf{E}_{\downarrow k}\imath(\mathbf{\Sigma}^{\mathrm{L}} - (\mathbf{\Sigma}^{\mathrm{L}})^\dagger)\mathbf{E}_{\downarrow k}^\dagger]_{kk} = \imath[\mathbf{E}_{\downarrow k}((\varepsilon\mathbf{S} - \mathbf{H} - \mathbf{\Sigma}^{\mathrm{R}} - \mathbf{A}) - (\varepsilon\mathbf{S} - \mathbf{H} - \mathbf{\Sigma}^{\mathrm{R}} - \mathbf{A})^\dagger)\mathbf{E}_{\downarrow k}^\dagger]_{kk} \\ &= \imath[\mathbf{E}_{\downarrow k}(\mathbf{A}^\dagger - \mathbf{A} - (\mathbf{\Sigma}^{\mathrm{R}} - (\mathbf{\Sigma}^{\mathrm{R}})^\dagger))\mathbf{E}_{\downarrow k}^\dagger]_{kk} \\ &= \imath([\mathbf{E}_{\downarrow k}\mathbf{A}\mathbf{E}_{\downarrow k}^\dagger]_{kk}^\dagger - [\mathbf{E}_{\downarrow k}\mathbf{A}\mathbf{E}_{\downarrow k}^\dagger]_{kk}) - \imath[\mathbf{E}_{\downarrow k}(\mathbf{\Sigma}^{\mathrm{R}} - (\mathbf{\Sigma}^{\mathrm{R}})^\dagger)\mathbf{E}_{\downarrow k}^\dagger]_{kk}. \end{aligned} \tag{46}$$

Here we used that both $\mathbf{S}$ and $\mathbf{H}$ are Hermitian and therefore vanish in the expression. The first term involving $\mathbf{A}$ is simplified via the fact that the (k,k)-subblock of the block tridiagonal $\mathbf{E}_{\downarrow k}\mathbf{A}$ remains invariant under the column operations by $\mathbf{E}_{\downarrow k}^\dagger$, and thus $[\mathbf{E}_{\downarrow k}\mathbf{A}\mathbf{E}_{\downarrow k}^\dagger]_{kk} = \mathbf{d}_{kk}^{\mathrm{L}}$. The last term, involving self-energies, is simplified via Eq. (21). We get

$$[\mathbf{\Gamma}_{\downarrow k}^{\mathrm{L}}]_{kk} = \imath((\mathbf{d}_{kk}^{\mathrm{L}})^\dagger - \mathbf{d}_{kk}^{\mathrm{L}}) - [\mathbf{E}_{\downarrow k}\mathbf{\Gamma}^{\mathrm{R}}\mathbf{E}_{\downarrow k}^\dagger]_{kk}. \tag{47}$$

Since $\mathbf{E}_{\downarrow k}$ represents downwards elimination, the (k,k)-block in $\mathbf{E}_{\downarrow k}\mathbf{\Gamma}^{\mathrm{R}}\mathbf{E}_{\downarrow k}^\dagger$ is left unaltered, i.e., $[\mathbf{E}_{\downarrow k}\mathbf{\Gamma}^{\mathrm{R}}\mathbf{E}_{\downarrow k}^\dagger]_{kk} = [\mathbf{\Gamma}^{\mathrm{R}}]_{kk} = \gamma_{kk}^{\mathrm{R}}$. Hence:

$$[\mathbf{\Gamma}_{\downarrow k}^{\mathrm{L}}]_{kk} = \imath((\mathbf{d}_{kk}^{\mathrm{L}})^\dagger - \mathbf{d}_{kk}^{\mathrm{L}}) - \gamma_{kk}^{\mathrm{R}}. \tag{48}$$

Following a similar procedure, we obtain:

$$[\mathbf{\Gamma}_{\uparrow k}^{\mathrm{R}}]_{kk} = \imath((\mathbf{d}_{kk}^{\mathrm{R}})^\dagger - \mathbf{d}_{kk}^{\mathrm{R}}) - \gamma_{kk}^{\mathrm{L}}. \tag{49}$$

Thus we have all the terms necessary for the calculation of transmission via Eq. (43).

References

[1] P. Pernas, A. Martin-Rodero, F. Flores, Electrochemical–potential variations across a constriction, Phys. Rev. B 41 (1990) 8553–8556.

[2] W. Tian, S. Datta, Aharonov–Bohm-type effect in graphene tubules: a Landauer approach, Phys. Rev. B 49 (1994) 5097–5100.

[3] L. Chico, M. Sancho, M. Munoz, Carbon-nanotube-based quantum dot, Phys. Rev. Lett. 81 (1998) 1278–1281.

[4] A. de Parga, O.S. Hernan, R. Miranda, A.L. Yeyati, A. Martin-Rodero, F. Flores, Electron resonances in sharp tips and their role in tunneling spectroscopy, Phys. Rev. Lett. 80 (1998) 357–360.

[5] N.D. Lang, Resistance of atomic wires, Phys. Rev. B 52 (1995) 5335–5342.

[6] K. Hirose, M. Tsukada, First-principles calculation of the electronic structure for a bielectrode junction system under strong field and current, Phys. Rev. B 51 (1995) 5278–5290.

[7] M.B. Nardelli, Electronic transport in extended systems: application to carbon nanotubes, Phys. Rev. B 60 (1999) 7828–7833.

[8] M.B. Nardelli, J. Bernholc, Mechanical deformations coherent transport in carbon nanotubes, Phys. Rev. B 60 (1999) R16338–R16341.

[9] J.J. Palacios, A.J. Pérez-Jiménez, E. Louis, J.A. Vergés, Fullerene-based molecular nanobridges: a first-principles study, Phys. Rev. B 64 (2001) 115411.

[10] P.A. Derosa, J.M. Seminario, Electron transport through single molecules: scattering treatment using density functional and Green Function theories, J. Phys. Chem. B 105 (2001) 471–481.

[11] S.N. Yaliraki, A.E. Roitberg, C. Gonzalez, V. Mujica, M.A. Ratner, The injecting energy at molecule/metal interfaces: implications for conductance of molecular junctions from an ab initio molecular description, J. Chem. Phys. 111 (1999) 6997–7002.

[12] J. Taylor, H. Guo, J. Wang, Ab initio modeling of open systems: charge transfer, electron conduction, and molecular switching of a C_{60} device, Phys. Rev. B 63 (2001) 121104.

[13] M. Stilling, K. Stokbro, K. Flensberg, Electronic transport in crystalline magnetotunnel junctions: effects of structural disorder, J. Comput.-Aid. Mater. Des. 14 (2007) 141–149.

[14] M. Stilling, K. Stokbro, K. Flensberg, Crystalline magnetotunnel junctions: Fe–MgO–Fe, Fe–FeOMgO–Fe and Fe–AuMgOAu–Fe, Nanotech 3 (2006) 39–42.

[15] M. Brandbyge, J.L. Mozos, P. Ordejón, J. Taylor, K. Stokbro, Density-functional method for nonequilibrium electron transport, Phys. Rev. B 65 (2002) 165401.

[16] J. Taylor, M. Brandbyge, K. Stokbro, Theory of rectification in tour wires: the role of electrode coupling, Phys. Rev. Lett. 89 (2002) 138301.

[17] K. Stokbro, J. Taylor, M. Brandbyge, Do Aviram–Ratner diodes rectify? J. Amer. Chem. Soc. 125 (2003) 3674–3675.

[18] K. Stokbro, J.L. Mozos, P. Ordejón, M. Brandbyge, J. Taylor, Theoretical study of the nonlinear conductance of di-thiol benzene coupled to Au(1 1 1) surfaces via thiol and thiolate bonds, Comput. Mater. Sci. 27 (2003) 151–160.
[19] R. Hoffmann, An extended Hückel theory. I. Hydrocarbons, J. Chem. Phys. 39 (1963) 1397–1412.
[20] W. Kohn, L.J. Sham, Self-consistent equations including exchange and correlation effects, Phys. Rev. 140 (1965) A1133–A1138.
[21] S. Datta, Electronic Transport in Mesoscopic Systems, Cambridge Univ. Press, New York, 1996.
[22] E.M. Godfrin, A method to compute the inverse of an *n*-block tridiagonal quasi-Hermitian matrix, J. Phys.: Condens. Matter 3 (1991) 7843–7848.
[23] J.D. Gilbert, L. Gilbert, Linear Algebra and Matrix Theory, Academic Press Inc., 1995.
[24] D.S. Fisher, P.A. Lee, relation between conductivity and transmission matrix, Phys. Rev. B 23 (1981) 6851–6854.
[25] H. Haug, A.-P. Jauho, Quantum Kinetics in Transport and Optics of Semiconductors, Springer-Verlag, Berlin, Heidelberg, 1996.
[26] P.S. Drouvelis, P. Schmelcher, P. Bastian, Parallel implementation of the recursive Green's function method, J. Comp. Phys. 215 (2006) 741–756.
[27] S.V. Faleev, F. Léonard, D.A. Stewart, M. van Schilfgaarde, Ab initio tight-binding LMTO method for nonequilibrium electron transport in nanosystems, Phys. Rev. B 71 (2005) 195422.
[28] O. Hod, J.E. Peralta, G.E. Scuseria, First-principles electronic transport calculations in finite elongated systems: a divide and conquer approach, J. Chem. Phys. 125 (2006) 114704.
[29] M.P. López Sancho, J.M. López Sancho, J. Rubio, Highly convergent schemes for the calculation of bulk and surface Green functions, J. Phys. F: Met. Phys. 15 (1985) 851–858.
[30] H.H. Sørensen, D.E. Petersen, P.C. Hansen, S. Skelboe, K. Stokbro, Efficient wave function matching approach for quantum transport calculations, Phys. Rev. B, submitted for publication.
[31] P.A. Khomyakov, G. Brocks, V. Karpan, M. Zwierzycki, P.J. Kelly, Conductance calculations for quantum wires and interfaces: mode matching and Green's functions, Phys. Rev. B 72 (2005) 035450.
[32] T. Ando, Quantum point contacts in magnetic fields, Phys. Rev. B 44 (1991) 8017–8027.
[33] G.H. Golub, C.F. van Loan, Matrix Computations, Johns Hopkins University Press, London, 1996.
[34] S. Lang, Linear Algebra, Springer-Verlag, New York, 1987.

Appendix B

Article: Krylov subspace method for evaluating the self-energy matrices in electron transport calculations

Accepted at Physical Review B

Listed in references as [2].

Krylov subspace method for evaluating the self-energy matrices in electron transport calculations

Hans Henrik B. Sørensen* and Per Christian Hansen
Department of Informatics and Mathematical Modelling, Technical University of Denmark, Building 321, DK-2800 Lyngby, Denmark

Dan Erik Petersen and Stig Skelboe
Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark

Kurt Stokbro
Nano-Science Center, University of Copenhagen, Universitetsparken 5, Building D, DK-2100 Copenhagen, Denmark

(Received 29 August 2007; revised manuscript received 4 March 2008; published 1 April 2008; corrected 3 April 2008)

We present a Krylov subspace method for evaluating the self-energy matrices used in the Green's function formulation of electron transport in nanoscale devices. A procedure based on the Arnoldi method is employed to obtain solutions of the quadratic eigenvalue problem associated with the infinite layered systems of the electrodes. One complex and two real shift-and-invert transformations are adopted to select interior eigenpairs with complex eigenvalues on or in the vicinity of the unit circle that correspond to the propagating and evanescent modes of most influence in electron transport calculations. Numerical tests within a density functional theory framework are provided to validate the accuracy and robustness of the proposed method, which in most cases is an order of magnitude faster than conventional methods.

DOI: 10.1103/PhysRevB.77.155301 PACS number(s): 73.40.−c, 73.63.−b, 72.10.−d, 85.65.+h

I. INTRODUCTION

Quantum transport has been an important research subject for more than a decade due to the ever-growing interest in simulating and fabricating nanoscale electronic devices. In particular, the experimental and theoretical investigation of current-voltage (I-V) characteristics for molecules and atomic structures placed between conducting electrodes has attracted much effort.[1–11] Most theoretical approaches are based on the Landauer-Büttiker formulation of quantum transport,[12] where the electrical properties of a central interface are described by the transmission coefficients of a number of one-electron states propagating coherently through the system. The widely used Green's function method[13,14] and the wave function matching method[15–17] are two such techniques. To apply these in practice and determine the current through a device under finite bias, it is necessary to evaluate the bulk modes or, correspondingly, the self-energy matrices of each electrode for a considerable number of different energies (chemical potentials) and possibly k points.[18] In many cases, this represents the dominant part of the computational work associated with electron transport calculations, assuming that the Hamiltonian of the system has been provided.

In this paper we develop an efficient method for computing the self-energy matrices using an iterative Krylov subspace technique. The foundation of the method is the evaluation of the self-energy matrices for the semi-infinite electrodes from the solutions of the quadratic eigenvalue problem (QEP) that arises for infinite periodic systems. This approach has been suggested by Ando[19] and studied by several authors.[15,16,20–23] It has been shown[16,24] to be equivalent to well-established iterative and recursive schemes.[25,26] A disadvantage of the latter schemes from a computational point of view is the need to introduce a small imaginary part in the energy in order to ensure that the iterations converge to the correct retarded surface Green's function. This imaginary part forces complex arithmetic in the numerical algorithms used, which is not always the case in the eigenproblem approach.[15,19]

The key motivation for developing the proposed method is the physical observation that only the propagating and the slowly decaying evanescent modes in the bulk electrodes contribute to the transmission of electrons through a semiconductor device of some extension.[8] These modes correspond to the solutions of the QEP that have complex eigenvalues in the vicinity of the unit circle. As recently suggested by Khomyakov *et al.*,[15] this makes it plausible to generate reduced self-energy matrices on the basis of a few selected solutions of the QEP, which include all the electrode-device coupling information that is necessary to determine the correct transmission. To really exploit such an approach in practice, an algorithm to search for and compute *only* the desired quadratic eigenpairs is required.

We will here consider the Arnoldi method[27] combined with a shift-and-invert strategy in order to obtain the QEP solutions. These techniques have proven effective in obtaining selected interior eigenvalues of large-scale general complex eigenproblems.[28–30] In addition, the recent surge of papers studying the Arnoldi procedure applied specifically to polynomial matrix problems indicates that this is a successful technique to build the Krylov subspace for QEPs.[31–34] The algorithm we develop assumes real Hamiltonian matrices (generalization to the complex case is described in Appendix A 2), and targets the complex eigenvalues which are on or inside the unit circle by applying shift-and-invert spectral transformations to $\pm 1/\sqrt{2}$ and $\hat{\imath}/\sqrt{2}$, where $\hat{\imath}$ is the imaginary unit, and subsequently generating a Krylov subspace for each with the Arnoldi method. Ritz pairs obtained by projecting the QEP onto the three Krylov subspaces give good approximations to the eigenpairs with eigenvalues close to the corresponding shifts. We will show that this method of proceeding is both rigorous and efficient by applying it to various Hamiltonians obtained using density functional theory

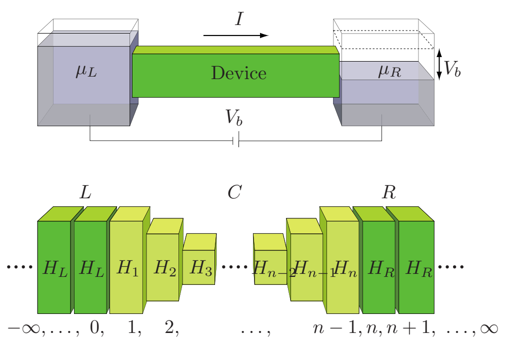

I
μ_L
Device
μ_R
V_b
V_b
L
C
R
H_L H_L H_1 H_2 H_3 H_{n-2} H_{n-1} H_n H_R H_R
$-\infty, \ldots,\ 0,\ \ 1,\ \ 2,\ \ \ldots,\ \ n-1, n, n+1,\ \ldots, \infty$

FIG. 1. (Color online) Schematic representation of a two-probe device with applied bias V_b. The top figure illustrates the Landauer-Büttiker picture of coherent scattering between electron reservoirs kept at chemical potentials μ_L and μ_R. The bottom figure shows the device part modeled by two semi-infinite electrodes (L and R) and a central region (C), each divided into principal layers that interact only with nearest-neighbor layers. The layers are described by square Hamiltonian matrices H_i of varying sizes and numbered $i=-\infty,\ldots,\infty$, as indicated.

(DFT) calculations with a localized basis of atomic orbitals.[35]

This paper is organized as follows. In Sec. II we give a brief exposition of our formalism for electron transport. The Krylov subspace method is introduced in Sec. III with details on its key parts: the Arnoldi method, the spectral transformations, and the convergence criterion. Typical convergence behavior is discussed in Sec. IV. The paper ends with numerical examples in Sec. V and a few concluding remarks.

II. ELECTRON TRANSMISSION AND SELF-ENERGY MATRICES

In this section we introduce our formalism, which combines the well-established Green's function method used for electron transport calculations[13,14,36] with the self-energy matrices obtained with the eigenvalue approach of Ando[19] as used in the wave function matching (WFM) method.[15–17] Our goal in combining the methods is to obtain, in the most efficient way, the spectrum of transmission coefficients $T(E)$ for two-probe systems (see top illustration in Fig. 1) in order to calculate the current $I=2e/h\int_{-\infty}^{\infty}T(E)[n_F(E-\mu_L)-n_F(E-\mu_R)]dE$ through the device, where E are the energies, n_F is the Fermi function, and μ_L and μ_R are the chemical potentials of the left (L) and right (R) electron reservoirs.[13,14]

A. Two-probe setup

Consider a two-probe system, as illustrated in the lower part of Fig. 1, where the device corresponds to the central region (C) and the reservoirs are two semi-infinite electrodes (L and R). The system has been divided into principal layers that interact only with nearest-neighbor layers and each layer is assumed to be described by appropriate Hamiltonian $\boldsymbol{H}_i$ and overlap $\boldsymbol{S}_i$ matrices, where i is the layer number, as represented, e.g., in a basis of localized nonorthogonal atomic orbitals. In this manner the Hamiltonian and overlap matrices are block-tridiagonal infinite matrices, where the off-diagonal blocks may be written $\boldsymbol{H}_{i,j}$ and $\boldsymbol{S}_{i,j}$. For the electrode Hamiltonian and overlap matrices we use subscripts L and R instead of numbers i,j. Notice also that the C region in this setup contains at least one layer of each electrode, which means that $\boldsymbol{H}_1=\boldsymbol{H}_L$ and $\boldsymbol{H}_n=\boldsymbol{H}_R$.

We refer the reader to Refs. 13, 14, and 36 for details on how to apply the Green's function method to the current setup. Here we limit ourselves to writing the primary results: First, the finite central region part of the infinite retarded Green's function matrix can be obtained as

$$\boldsymbol{G}_C^r=[(E+\hat{\imath}\eta)\boldsymbol{S}-\boldsymbol{H}_C-\boldsymbol{\Sigma}_L-\boldsymbol{\Sigma}_R]^{-1}, \tag{1}$$

where η is an infinitesimal quantity, $\boldsymbol{H}_C$ is the central region Hamiltonian, and the effect of the semi-infinite electrodes is accommodated through self-energy matrices $\boldsymbol{\Sigma}_L$ and $\boldsymbol{\Sigma}_R$. Second, the total transmission coefficient $T(E)$ is then given by

$$T(E)=\mathrm{Tr}\{\boldsymbol{\Gamma}_L\boldsymbol{G}_C^r\boldsymbol{\Gamma}_R\boldsymbol{G}_C^a\}, \tag{2}$$

where $\boldsymbol{\Gamma}_{L/R}=\hat{\imath}(\boldsymbol{\Sigma}_{L/R}-\boldsymbol{\Sigma}_{L/R}^{\dagger})$ are coupling matrices and $\boldsymbol{G}_C^a$ is the advanced central Green's function matrix, which is obtained from Eq. (1) by using $-\hat{\imath}\eta$ as the infinitesimal imaginary component in all terms (i.e., implicitly in $\boldsymbol{\Sigma}_L$ and $\boldsymbol{\Sigma}_R$).

We find that an efficient approach (see Appendix A 1) to applying Eqs. (1) and (2) is to compute only a single *diagonal* block of $\boldsymbol{G}_C^r$ in order to evaluate $T(E)$. The question remains how to calculate the required self-energy matrices $\boldsymbol{\Sigma}_{L/R}$ in the most efficient manner.

B. Electrode self-energy matrices from QEPs

It is known that the surface Green's function matrices for a semi-infinite ideal electrode can be evaluated by recursive techniques that take 2^n-1 electrode layers into account in n iterations.[25,26] This is a fast and widely used approach to obtain the self-energy matrices when employing the Green's function method.[1,37]

Another approach has been proposed by Ando,[19] where one constructs and solves an appropriate QEP (introducing notation $\overline{\boldsymbol{H}}\equiv E\boldsymbol{S}-\boldsymbol{H}$)

$$\overline{\boldsymbol{H}}_{L,L}^{\dagger}\phi_k+\lambda_k\overline{\boldsymbol{H}}_L\phi_k+\lambda_k^2\overline{\boldsymbol{H}}_{L,L}\phi_k=0, \tag{3}$$

for $k=1,\ldots,2M_L$, where M_L is the number of orbitals local to the unit cell of the left electrode and similarly for the right electrode with $L\rightarrow R$. The procedure to determine the nontrivial solutions (i.e., the Bloch factors λ_k and electrode modes ϕ_k) from Eq. (3), and subsequently characterize these as propagating or evanescent, right-going (+) or left-going (−), is well described in the literature (we refer the reader to details in Refs. 15 and 16).

Applying Ando's approach via the formalism of the WFM method yields expressions[16]

$$\boldsymbol{\Sigma}_0^L=-\overline{\boldsymbol{H}}_{L,L}^{\dagger}(\boldsymbol{B}_L^{-})^{-1}, \tag{4}$$

$$\boldsymbol{\Sigma}_{n+1}^R=-\overline{\boldsymbol{H}}_{R,R}\boldsymbol{B}_R^{+} \tag{5}$$

for the electrode self-energy matrices in the layers 0 and n

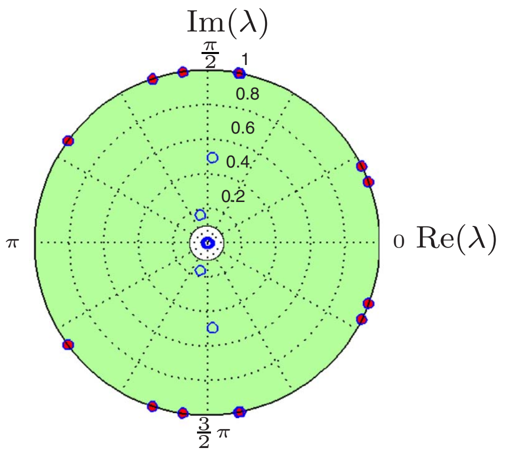


FIG. 2. (Color online) Positions of the 243 complex eigenvalues [blue (circles)] inside the unit disk for a Au(111) electrode with 27 atoms per unit cell at $E=-2$ eV. The 21 eigenvalues corresponding to propagating modes [red (filled) dots] are located on the unit circle. The modes of most significance in transmission calculations are located within the green (shaded) area given by $0.1 \leq |\lambda| \leq 1$.

$+1$ just *outside* the C region, where $\boldsymbol{B}_{L/R}^{\pm}$ are the Bloch matrices constructed from the solutions λ_k and ϕ_k [see the expressions in Ref. 16, in which the notation is $\mathbf{F}_{L/R}(\pm)$ for the Bloch matrices, and $\lambda_n(\pm)$ and $\boldsymbol{u}_n(\pm)$ for the solutions]. After evaluating these self-energy matrices we use them in the Green's function method described above (we set $\eta=0$ in this case, since the retarded Green's function is already uniquely defined by the self-energies[16,21]) and follow the steps outlined in Appendix A 1.

C. Reduced self-energy matrices

From a numerical perspective, it is convenient to keep only those eigenpairs from Eq. (3) that have eigenvalues λ_k within specific intervals[15]

$$\lambda_{\min} \leq |\lambda_k^{+}| \leq 1, \quad 1 \leq |\lambda_k^{-}| \leq \lambda_{\min}^{-1}, \tag{6}$$

for a reasonable choice of $\lambda_{\min}$. Evanescent modes with $|\lambda_k|$ outside these intervals are decaying or growing so fast that they have negligible influence in a two-probe setup like ours. The decisive factor in choosing $\lambda_{\min}$ is that the sets $\{\phi_k^{+}\}$ and $\{\phi_k^{-}\}$ of electrode modes included must be complete in the sense that they can fully represent the transmitted and reflected waves (cf. the WFM formalism).

In what follows, we exploit that a reasonable choice of $\lambda_{\min}$ for transmission calculations with our setup is often of the order 0.1.[38] For example, in the case of the polar plot in Fig. 2, where the Bloch factors with $|\lambda_k| \leq 1$ of a 27-atom Au(111) electrode unit cell are shown, the computationally significant modes can be identified as the eigenvalues inside the shaded area (i.e., by setting $\lambda_{\min}=0.1$). The numerical results given in Sec. V illustrate this observation quantitatively. A proper formal analysis is left for a future publication.[39]

III. KRYLOV SUBSPACE METHOD

In this section, we describe the Krylov subspace method for evaluating the electrode self-energy matrices $\boldsymbol{\Sigma}_0^L$ and $\boldsymbol{\Sigma}_{n+1}^R$. The crucial assumption in the approach is that we may strip the less important modes from the sets $\{\phi_k^{+}\}$ and $\{\phi_k^{-}\}$, and still obtain a good approximation to the self-energy matrix to be used in transmission calculations. For simplicity, we also assume that the electrode Hamiltonians are real, and give in Appendix A 2 a prescription to generalize to the complex case. Our current method, which targets the specific modes that are most important, can be characterized as a shift-and-invert Arnoldi method with adaptive subspace size. We will describe the key ingredients of the method: the Arnoldi procedure, the spectral transformations, and the convergence criterion. The goal is to present an alternative for obtaining the self-energy matrices, which is faster than existing techniques.

A. Arnoldi procedure

The Krylov subspace of dimension m generated by an $n \times n$ matrix $\boldsymbol{A}$ and an initial vector $\boldsymbol{v}_1$ is given by $\mathcal{K}_m(\boldsymbol{A},\boldsymbol{v}_1) \equiv \mathrm{span}\{\boldsymbol{v}_1, \boldsymbol{A}\boldsymbol{v}_1, \boldsymbol{A}^2\boldsymbol{v}_1, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{v}_1\}$.[40] In order to determine this space we apply the Arnoldi procedure[27] which generates an orthonormal basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ for $\mathcal{K}_m(\boldsymbol{A},\boldsymbol{v}_1)$. We use the numerically most stable scheme that employs the modified Gram-Schmidt orthogonalization to successively construct the orthonormal vectors $\boldsymbol{v}_i$. Algorithm I below lists the steps of a continuable version of the Arnoldi procedure which is initially called with a parameter $k=1$ and a random starting vector $\boldsymbol{v}_1$. After $m-1$ iterations the $n \times m$ matrix $\boldsymbol{V}_m = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ is available.

The projection of the matrix $\boldsymbol{A}$ onto $\mathcal{K}_m(\boldsymbol{A},\boldsymbol{v}_1)$ is then $\boldsymbol{H}_m = \boldsymbol{V}_m^{\dagger}\boldsymbol{A}\boldsymbol{V}_m$, where $\boldsymbol{H}_m$ is $m \times m$ and upper Hessenberg (i.e., it has zeros below its lower bidiagonal). The matrix $\boldsymbol{H}_m$ is also constructed by Algorithm I. Approximate solutions of the eigenproblem $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$ can subsequently be obtained as the so-called Ritz eigenpairs $(\gamma, \boldsymbol{V}_m\boldsymbol{y})$ of the projected eigenproblem $\boldsymbol{H}_m\boldsymbol{y} = \gamma\boldsymbol{y}$. As m increases the Ritz pairs become increasingly better approximations to certain eigenpairs of $\boldsymbol{A}$ (we point to Refs. 38 and 39 for details).

Algorithm I: Arnoldi procedure (continuable). Input: $k, m \in \mathbb{Z}$, $\boldsymbol{A} \in \mathbb{R}^{n,n}$, $\boldsymbol{V}_k \in \mathbb{R}^{n,k}$, $\boldsymbol{H}_k \in \mathbb{R}^{k,k}$. Output: $\boldsymbol{V}_{m+1} \in \mathbb{R}^{n,m+1}$, $\boldsymbol{H}_{m+1} \in \mathbb{R}^{m+1,m+1}$.

```
(1) If $k=1$, $\boldsymbol{v}_1 = \boldsymbol{v}_1/\|\boldsymbol{v}_1\|_2$
(2) for $j=k,k+1,\ldots,m$ do
(3) $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{v}_j$
(4) for $i=1,2,\ldots,j$ do
(5) $h_{ij} = \boldsymbol{v}_i^T\boldsymbol{v}$
(6) $\boldsymbol{v} = \boldsymbol{v} - h_{ij}\boldsymbol{v}_i$
(7) end
(8) $h_{j+1,j} = \|\boldsymbol{v}\|_2$
(9) if $h_{j+1,j}=0$, $m=j$, stop
(10) $\boldsymbol{v}_{j+1} = \boldsymbol{v}/h_{j+1,j}$
(11) end
```

One cannot know in advance how many steps will be needed before the eigenpairs of interest are well approximated by Ritz pairs. If many steps are necessary, then solving the projected eigenvalue problem becomes costly. Moreover, when applying our Krylov method to evaluate the self-energy matrices, we do not know the exact number of eigenpairs wanted and cannot estimate the required dimension of the Krylov subspace.

The first difficulty can be circumvented by restarting the Arnoldi method after a certain number of iterations using the obtained information to generate a better starting vector, or by deflating particular eigenvalues.[41] However, this will not improve on the second difficulty which requires an adaptive maximum dimension of the Krylov subspace. In addition, we observe in most of our applications that the gain from an efficient restart procedure (e.g., as devised by Morgan and Zeng[42]) does not outweigh the computational expense of the restarting overhead. The typical size of the self-energy matrices encountered is too small to make it beneficial to use such techniques, which have been developed for large-scale applications.

Therefore, we have chosen to employ a simple continuation scheme instead of restarting, where a check for convergence is performed after a given number of Arnoldi iterations, and if we are not satisfied, the procedure simply continues where it was left off. With the input parameter k, the listed Arnoldi algorithm is able to generate an initial Krylov subspace $\mathcal{K}_m$ of a given dimension m, but also to continue the process, augmenting the space with subsequent calls. This allows us to perform iterations as long as the approximations are unsatisfactory and/or there is doubt whether all wanted eigenpairs have been found.

An important special case to be considered when applying the Arnoldi procedure to solve an eigenvalue problem is that of algebraically multiple eigenvalues. A Krylov subspace method will, in theory, produce only one eigenvector corresponding to a multiple eigenvalue. So determination of multiplicity is quite difficult. Several approaches exist that deal with this problem, including deflation combined with effects of round-off error,[41] block Arnoldi procedures,[41] and so-called random restarts.[42,43] The present Krylov method does not incorporate any mechanisms to take algebraic multiplicity into account because such cases do not occur in practice for the applications of this work (eigenvalues will not be identical to machine precision in any of the numerical examples, but only to within ~10–11 digits; see Sec. IV).

B. Shift-and-invert transformations

Iterative methods based on Krylov subspaces produce Ritz values that converge fastest to the dominant part of the eigenvalue spectrum given by the extremal eigenvalues.[40] In the current application, it is the interior of the eigenvalue spectrum that is of interest, in particular the eigenvalues λ that satisfy $\lambda_{\min}\le|\lambda|\le\lambda_{\min}^{-1}$. To be able to find this part of the spectrum efficiently, we employ a shift-and-invert strategy which implies that the QEP in Eq. (3) is rewritten as

$$(\mu^2\boldsymbol{M}+\mu\boldsymbol{C}+\boldsymbol{K})\boldsymbol{c}_0=0, \tag{7}$$

where

$$\boldsymbol{M}=\overline{\boldsymbol{H}}_{L,L}^T+\sigma\overline{\boldsymbol{H}}_L+\sigma^2\overline{\boldsymbol{H}}_{L,L}, \tag{8}$$

$$\boldsymbol{C}=\overline{\boldsymbol{H}}_L+2\sigma\overline{\boldsymbol{H}}_{L,L}, \tag{9}$$

$$\boldsymbol{K}=\overline{\boldsymbol{H}}_{L,L}, \tag{10}$$

and

$$\mu=\frac{1}{\lambda-\sigma}. \tag{11}$$

With this approach, the eigenvalues λ of Eq. (3) have been shifted by σ and inverted while the eigenvectors $\boldsymbol{c}_0$ are unchanged. Thus the dominant part of the spectrum of Eq. (7) now corresponds to the eigenvalues of the original QEP closest to the shift σ.

The simplest and currently state-of-the-art technique for solving Eq. (7) is by linearizing it to a generalized eigenvalue problem of twice the size.[44] In our case $\boldsymbol{M}$ is nonsingular and has size M_L. Therefore, a linearization results in a standard eigenvalue problem of size $2M_L$:

$$\boldsymbol{A}\boldsymbol{x}=\mu\boldsymbol{x}, \tag{12}$$

where $\boldsymbol{A}$ is given by

$$\boldsymbol{A}=\begin{pmatrix}0 & \boldsymbol{I}\\ -\boldsymbol{M}^{-1}\boldsymbol{K} & -\boldsymbol{M}^{-1}\boldsymbol{C}\end{pmatrix}, \tag{13}$$

and the $2M_L$ eigenvalues μ are identical to the ones of Eq. (7). The eigenvectors of Eq. (12) are given by $\boldsymbol{x}^T=(\boldsymbol{c}_0^T,\mu\boldsymbol{c}_0^T)$, so that the original eigenvectors $\boldsymbol{c}_0$ can be selected as the first M_L elements of $\boldsymbol{x}$.

If we assume that the Hamiltonian and overlap matrices for the electrodes are real, then the spectrum of the QEP in Eq. (3) is symmetric with respect to the real axis of the complex plane, and the eigenvalues either are real or occur in complex conjugate pairs.[44] In addition, as seen by transposing Eq. (3), the eigenvalues in this case also come in pairs, λ and $1/\lambda$. We will use these properties to present a simplified method for the extraordinary case of real $\overline{\boldsymbol{H}}_L$ and $\overline{\boldsymbol{H}}_{L,L}$, and subsequently discuss the steps required for the general complex case in Appendix A 2.

The purpose of the current method is thus to determine the eigenpairs $(\lambda,\boldsymbol{c}_0)$ of Eq. (3) that satisfy $\lambda_{\min}\le|\lambda|\le 1$ for a given $\lambda_{\min}>0$ [the pairs that satisfy $1\le|\lambda|\le\lambda_{\min}^{-1}$ can subsequently be obtained as $(\lambda^{-1},\boldsymbol{c}_0)$]. As is apparent from the polar plot example in Fig. 2, the majority of the eigenvalues with $|\lambda|\le 1$ are located near the origin. Therefore, it is not efficient to apply the shift $\sigma=0$ in order to obtain the wanted eigenvalues, which lie in the outskirts of the unit disk. Instead we may apply four different shifts, given by $\sigma=\pm 1/\sqrt{2}$ and $\sigma=\pm\hat{\imath}/\sqrt{2}$, in four separate Arnoldi procedures. Each of these then covers a quarter slice of the unit disk and produces Ritz values that converge fast to eigenvalues close to the given shift. Simple sorting techniques can be employed in each Arnoldi procedure to take into account only the portion of the Ritz pairs that is covered by a given shift.

When applying the shift-and-invert strategy devised, two of the shifts have to be complex. In practice this means working in complex arithmetic or doubling the size of the problem.[45] However, in the case of real Hamiltonians it is advantageous to search for the complex eigenvalues in conjugate pairs and thereby eliminate one of the complex shifts. Moreover, this can be done almost entirely in real arithmetic as follows.

Notice that Eq. (12) was obtained by linearizing the shifted-and-inverted QEP written in Eq. (7). We may also reverse the order of the linearization and shift-and-invert operations. By performing, e.g., a first companion linearization of Eq. (3) that results in an eigenproblem $\hat{\boldsymbol{A}}\boldsymbol{x}=\lambda\boldsymbol{x}$ of double size, and subsequently a shift-and-invert transformation arriving at $(\hat{\boldsymbol{A}}-\sigma\boldsymbol{I})^{-1}\boldsymbol{x}=\mu\boldsymbol{x}$, we see that the matrix applied in the Arnoldi procedures can also be written[44]

$$(\hat{\boldsymbol{A}}-\sigma\boldsymbol{I})^{-1}=\begin{pmatrix} -\boldsymbol{M}^{-1}\hat{\boldsymbol{C}} & -\boldsymbol{M}^{-1}\boldsymbol{K} \\ \boldsymbol{I}-\sigma\boldsymbol{M}^{-1}\hat{\boldsymbol{C}} & -\sigma\boldsymbol{M}^{-1}\boldsymbol{K} \end{pmatrix}, \tag{14}$$

where

$$\hat{\boldsymbol{C}}=\overline{\boldsymbol{H}}_L+\sigma\overline{\boldsymbol{H}}_{L,L}. \tag{15}$$

The eigenpairs $(\mu,\boldsymbol{x})$ of $(\hat{\boldsymbol{A}}-\sigma\boldsymbol{I})^{-1}\boldsymbol{x}=\mu\boldsymbol{x}$ are exactly the same as those of Eq. (12). In addition, we may now consider the combined spectral transformation for two conjugate shifts σ and σ^*, given by

$$\boldsymbol{T}=(\hat{\boldsymbol{A}}-\sigma\boldsymbol{I})^{-1}(\hat{\boldsymbol{A}}-\sigma^*\boldsymbol{I})^{-1}=\frac{\mathrm{Im}\{(\hat{\boldsymbol{A}}-\sigma\boldsymbol{I})^{-1}\}}{\mathrm{Im}\{\sigma\}}, \tag{16}$$

which was originally proposed by Parlett and Saad.[45] Applying the matrix $\boldsymbol{T}$ in the Arnoldi procedure generates approximate solutions to $\boldsymbol{T}\boldsymbol{x}=\mu'\boldsymbol{x}$, where the eigenvalues are given by

$$\mu'=\frac{1}{(\lambda-\sigma)(\lambda-\sigma^*)}, \tag{17}$$

which becomes extreme for conjugate eigenvalues λ and λ^* of Eq. (3) that are close to σ and σ^*. In our case, the complex shifts are purely imaginary: $\sigma=\hat{i}\beta$, where β is real. Then we have $\mu'=(\lambda^2+\beta^2)^{-1}$ and, more importantly, the matrix $\boldsymbol{T}$ is simply given by β^{-1} times the imaginary part of Eq. (14), written as

$$\boldsymbol{T}=\begin{pmatrix} -\beta^{-1}\,\mathrm{Im}\{\boldsymbol{M}^{-1}\hat{\boldsymbol{C}}\} & -\beta^{-1}\mathrm{Im}\{\boldsymbol{M}^{-1}\boldsymbol{K}\} \\ \mathrm{Re}\{\boldsymbol{M}^{-1}\hat{\boldsymbol{C}}\} & \mathrm{Re}\{\boldsymbol{M}^{-1}\boldsymbol{K}\} \end{pmatrix}, \tag{18}$$

which is purely real. This makes it feasible to use real arithmetic in all parts of the algorithm except for the initial complex LU factorization of $\boldsymbol{M}$, which is required for the matrix multiplications by $\boldsymbol{M}^{-1}$.

C. Algorithm and convergence criterion

The algorithm for our Krylov method is composed of two main parts, an iterative part that determines the wanted Ritz pairs $(\lambda,\boldsymbol{c}_0)$ which approximate the eigenpairs of the QEP in Eq. (3), and a noniterative part that sets up the Bloch matrices and evaluates the self-energy matrix from these by direct methods. The iterative part is organized as three independent computations, one for each of the used shifts σ. It consists of the application of the Arnoldi procedure together with a check for convergence plus the initial work to construct the input matrices for Algorithm I. As described in the previous section, the actual calculations will depend on whether the shift is real or imaginary.

The key steps of the Krylov method for evaluating the self-energy matrix $\boldsymbol{\Sigma}^L$ of the left electrode are presented in Algorithm II below. It is important to stress that the details of each step are kept at a minimum to enhance the readability. Furthermore, for evaluating the self-energy matrix $\boldsymbol{\Sigma}^R$ of the right electrode, the steps are exactly the same, except for the substitution $L\rightarrow R$ of all super- and subscripts and the removal of line 1 [this line is only required for left electrodes in order to obtain $\boldsymbol{\Sigma}^L$ from solutions $(\lambda^{-1},\boldsymbol{c}_0)$, e.g., by transposing Eq. (3)]. In the rest of this section we will discuss the main aspects of the algorithm.

Algorithm II: Krylov method to evaluate $\boldsymbol{\Sigma}^L$. Input: $m\in\mathbb{Z}$, $\lambda_{\min}\in[0,1]$, $\overline{\boldsymbol{H}}_L,\overline{\boldsymbol{H}}_{L,L},\overline{\boldsymbol{H}}^T_{L,L}\in\mathbb{R}^{M_L,M_L}$. Output: $\boldsymbol{\Sigma}^L\in\mathbb{C}^{M_L,M_L}$.

```
(1) Exchange matrices $\overline{\boldsymbol{H}}_{L,L}$ and $\overline{\boldsymbol{H}}^T_{L,L}$
(2) for $\sigma=1/\sqrt{2},-1/\sqrt{2},\hat{i}/\sqrt{2}$ do
(3) if $\sigma$ is real, calculate $\boldsymbol{A}$ from Eq. (13)
    else calculate $\boldsymbol{T}$ from Eq. (18) and set $\boldsymbol{A}=\boldsymbol{T}$
(4) select random vector $\boldsymbol{v}_1$ of size $2M_L$
(5) apply Algorithm I to generate $\mathcal{K}_m(\boldsymbol{A},\boldsymbol{v}_1)$
(6) solve the projected eigenproblem $\boldsymbol{H}_m\boldsymbol{y}=\mu\boldsymbol{y}$
(7) if $\sigma$ is real, select all $(\mu,\boldsymbol{y})$ that satisfy $\lambda_{\min}\leq|\mu^{-1}+\sigma|\leq 1+\epsilon$, and store the Ritz pairs $(\lambda,\boldsymbol{c}_0)=(\mu^{-1}+\sigma,\boldsymbol{V}_m\boldsymbol{y})$ that have $\mathrm{Re}(\lambda)\mathrm{Re}(\sigma)\geq|\lambda|/2$
    else select all $(\mu,\boldsymbol{y})$ that satisfy $\lambda_{\min}\leq|\mu^{-1}+\sigma^2|^{1/2}\leq 1+\epsilon$, and evaluate the eigenvalues $\lambda$ with the MR-2 method of Ref. 44 and store the Ritz pairs $(\lambda,\boldsymbol{c}_0)=(\lambda,\boldsymbol{V}_m\boldsymbol{y})$ that have $|\mathrm{Im}(\lambda)\mathrm{Im}(\sigma)|>|\lambda|/2$.
(8) for all stored Ritz pairs $(\lambda,\boldsymbol{c}_0)$, find residual $\|(\overline{\boldsymbol{H}}^T_{L,L}+\lambda\overline{\boldsymbol{H}}_L+\lambda^2\overline{\boldsymbol{H}}_{L,L})\boldsymbol{c}_0\|_2$, and check for convergence. If not satisfied, increase $m$ appropriately and go to step 5
(9) end
(10) for all stored Ritz pairs $(\lambda,\boldsymbol{c}_0)$ having $(1+\epsilon)^{-1}\leq\lambda\leq 1+\epsilon$, calculate group velocity $v$ (see Ref. 15); discard the pairs with $v<0$ (i.e., the left-going modes)
(11) evaluate $\boldsymbol{B}^+_L$ and $\boldsymbol{\Sigma}^L=-\overline{\boldsymbol{H}}_{L,L}\boldsymbol{B}^+_L$ from the remaining pairs
```

First consider the steps 3–8 composing the body of the FOR loop, which are independently executed for the three given shifts σ. Each execution of these steps will determine Ritz pairs that are located in the corresponding quarter-slices of the unit disk. An illustration is shown in Fig. 3 for an Al(100) electrode, where the distinct slices are indicated by shaded areas and the current shifts by crosses. All wanted Ritz pairs found independently for the given shifts are assumed to be collected in a combined set when exiting the loop at step 9.

Initially, in step 3, the linearized and shifted-and-inverted matrix $\boldsymbol{A}$ to be applied in the Arnoldi procedure is determined from Eq. (13) if σ is real and from Eq. (18) if σ is complex. Then a starting vector $\boldsymbol{v}_1$ is selected randomly in step 4. A random starting vector is a reasonable choice in our case, where no prior information about the approximated eigenspace is available. In step 5 the Arnoldi procedure of Algorithm I is called to generate a Krylov subspace of size m, and in step 6, the corresponding eigenpairs $(\mu,\boldsymbol{y})$ of the

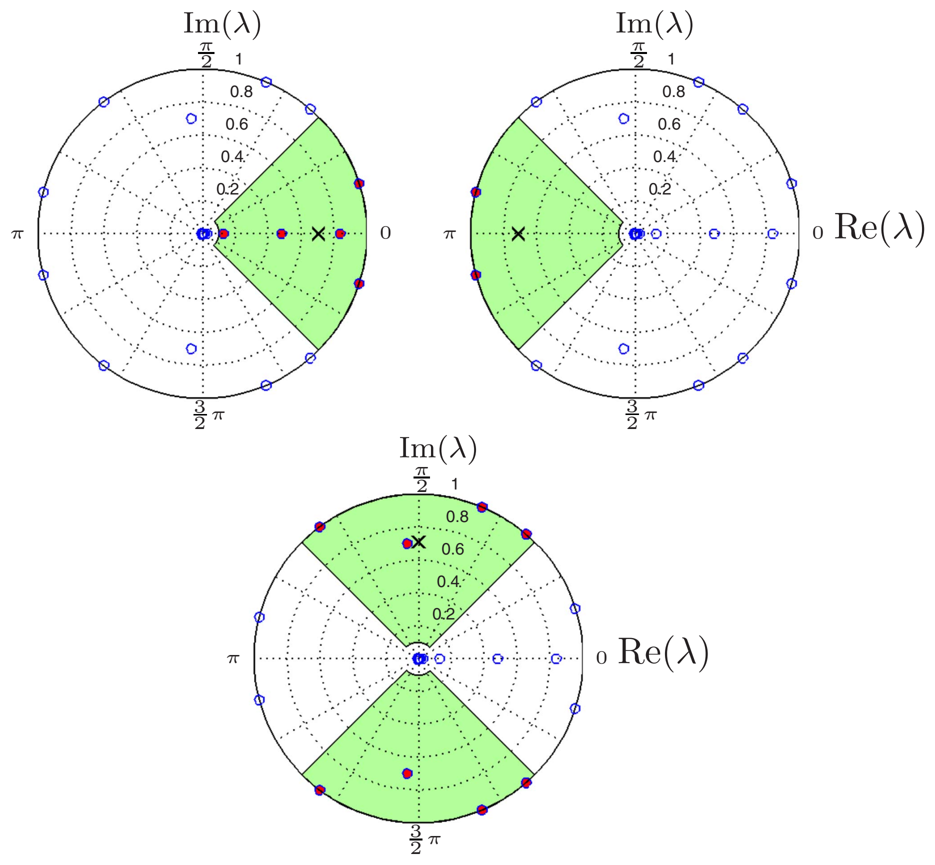


FIG. 3. (Color online) Illustration of the complex eigenvalues [blue (circles)] for the Al(100) electrode at $E=3$ eV. The eigenvalues corresponding to the wanted right-going modes [red (filled) dots] can be separated according to their location within three distinct green (shaded) areas of the unit disk and determined efficiently using shift-and-invert spectral transformations to $\pm 1/\sqrt{2}$ and $\hat{i}/\sqrt{2}$ (crosses).

shifted-and-inverted problem are found by solving the projected eigenproblem with a direct method. This is followed by an elaborate selection scheme to determine which of the available solutions $(\mu,\boldsymbol{y})$ correspond to wanted Ritz pairs $(\lambda,\boldsymbol{c}_0)$ that are located inside the valid quarter slice.

The selection scheme, as outlined in step 7, can be implemented as two separate processes. The first selection process is designed to identify those solutions $(\mu,\boldsymbol{y})$ that correspond to eigenpairs of the original QEP which satisfy $\lambda_{\min}\leq|\lambda|\leq 1$. It is important to realize, however, that, since all computations are done in finite-precision arithmetic, there is no guarantee that the propagating Bloch modes of the electrode will have magnitudes $|\lambda|$ exactly equal to 1. Even the left-going propagating modes that are targeted in our case can have $|\lambda|>1$. In practice, we therefore define the propagating modes to be those Ritz pairs $(\lambda,\boldsymbol{c}_0)$ that satisfy

$$(1+\epsilon)^{-1}\leq|\lambda|\leq 1+\epsilon \tag{19}$$

where ϵ is a small infinitesimal (set to 10^{-8} in our implementation). In order to make sure that all propagating modes are taken into consideration it is thus necessary to select all Ritz pairs that satisfy $\lambda_{\min}\leq|\lambda|\leq 1+\epsilon$.

To obtain the Ritz values λ used in the selection process, we have to transform the solutions $(\mu,\boldsymbol{y})$ of the projected eigenproblem to the corresponding Ritz pairs $(\lambda,\boldsymbol{c}_0)$ by reversing the shift-and-invert operation. The transformation again depends on whether the shift σ is real or imaginary. In the case of real σ, we have $\lambda=\mu^{-1}+\sigma$ from Eq. (11). For imaginary σ, Eq. (17) can be rearranged to $\lambda^2=\mu^{-1}+\sigma^2$, which has two solutions of equal magnitude. This is sufficient to allow selection on the basis of the magnitude $|\lambda|$; however, when it comes to obtaining the Ritz values λ themselves, it is necessary to use other means for imaginary σ, e.g., by forming the Rayleigh quotient.[40] In our case, and for QEPs in particular, it is possible and computationally advantageous to use alternatives to the Rayleigh quotient that work with vectors and matrices of size M_L instead of $2M_L$. Several such techniques that are both fast and accurate have recently been devised by Hochstenbach and van der Vorst.[46] We will adopt the MR-2 method of that paper, which yields $\lambda=\alpha/\beta$, for α and β defined as

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix}=-\tilde{\boldsymbol{Z}}\boldsymbol{H}_{L,L}^{T}\boldsymbol{c}_0, \tag{20}$$

where $\tilde{\boldsymbol{Z}}$ is the pseudoinverse of $\boldsymbol{Z}=(\overline{\boldsymbol{H}}_{L,L}\boldsymbol{c}_0,\overline{\boldsymbol{H}}_{L}\boldsymbol{c}_0)$. Since all eigenvectors are unchanged by the shift-and-invert operation, the $\boldsymbol{c}_0$ vectors applied here are the first M_L elements of the Ritz vectors $\boldsymbol{V}_m\boldsymbol{y}$.

The remaining selection process in step 7 should single out the Ritz pairs that are inside the valid slice of the unit disk. To this end, we can apply the inner product of $(\mathrm{Re}\{\lambda\},\mathrm{Im}\{\lambda\})$ and $(\mathrm{Re}\{\sigma\},\mathrm{Im}\{\sigma\})$, given by

$$\mathrm{Re}\{\lambda\}\mathrm{Re}\{\sigma\}+\mathrm{Im}\{\lambda\}\mathrm{Im}\{\sigma\}=|\lambda||\sigma|\cos\theta, \tag{21}$$

where θ is the angle between λ and σ in a polar representation of the complex plane. In order for λ to be inside the quarter slice that has σ on the bisector we must have $|\theta| \leq \pi/4$ or equivalently $\cos\theta \geq 1/\sqrt{2}$. For real shifts $\sigma = \pm 1/\sqrt{2}$, this observation yields the condition

$$\frac{\mathrm{Re}\{\lambda\}\mathrm{Re}\{\sigma\}}{|\lambda|} \geq \frac{1}{2}, \qquad (22)$$

and similarly for imaginary shift $\sigma = \hat{\imath}/\sqrt{2}$,

$$\frac{|\mathrm{Im}\{\lambda\}\mathrm{Im}\{\sigma\}|}{|\lambda|} > \frac{1}{2}, \qquad (23)$$

where the absolute value of the left-hand side is taken to allow λ to be in both the top and the bottom quarter slices. Notice that the equality is removed since the (very rare) event of λ lying exactly on the border of two slices is already taken into account in the condition for real σ.

In step 8 of Algorithm II the check for convergence is carried out. For each shift, the convergence condition is regarded as satisfied when all the Ritz pairs of interest that are also located inside the valid quarter slice are identified and accurate to a given tolerance. We estimate the accuracy of the obtained pairs $(\lambda, \boldsymbol{c}_0)$ by evaluating the corresponding relative residual norm, which yields the following convergence criterion:

$$\frac{\|(\overline{\boldsymbol{H}}_{L,L}^T + \lambda\overline{\boldsymbol{H}}_L + \lambda^2\overline{\boldsymbol{H}}_{L,L})\boldsymbol{c}_0\|_2}{\mathrm{norm}(\overline{\boldsymbol{H}}_L)} \leq \mathrm{tol} \qquad (24)$$

where tol is the convergence tolerance and $\mathrm{norm}(\overline{\boldsymbol{H}}_L)$ is an appropriate norm for matrix $\overline{\boldsymbol{H}}_L$. In our implementation we set $\mathrm{tol}=10^{-11}$ and apply the approximation $\mathrm{norm}(\overline{\boldsymbol{H}}_L) \approx \|\mathrm{diag}(\overline{\boldsymbol{H}}_L)\|_2$, that is, we include only the diagonal entries of the two-norm of $\overline{\boldsymbol{H}}_L$. These choices require very low computational effort and give the correct result for all numerical examples we have investigated.

In the event that the convergence check in step 8 of Algorithm II is not satisfied, we assume that the dimension m of the Krylov subspace $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{v}_1)$ generated in step 5, is insufficient. Therefore, we increase m by some fixed amount and go back to step 5 to continue the Arnoldi procedure where it was left off. In the current implementation, we chose to increase the size of the Krylov subspace by $\Delta m = m/2$, where m is the initial value of m given as input. Our experiments show that, for optimal efficiency with this Δm, it is favorable to have the initial m within the range 30–50 if the sizes of the input matrices are of order less than 1000. After convergence has been achieved, the final steps 10–11 of Algorithm II present the operations required to collect the Ritz pairs that have been determined and subsequently obtain the self-energy matrix.

IV. TYPICAL CONVERGENCE BEHAVIOR

In this section, we briefly exemplify the typical convergence behavior of Algorithm II by monitoring the relative residual norm of the wanted eigenpairs as a function of the number of iterations. An expression for this norm for a given eigenpair $(\lambda, \boldsymbol{c}_0)$ is available as the left-hand side of Eq. (24). We will consider the Al(100) electrode at $E=3$ eV and parameter $\lambda_{\min}=0.1$, which requires a total of 13 eigenpairs to be determined (eight propagating modes and five evanescent modes) from the three separate Arnoldi procedures. This example corresponds to the situation illustrated in Fig. 3 and represents a typical calculation for an Al(100) electrode with 18 atoms per unit cell (the size of the self-energy matrix is 72).

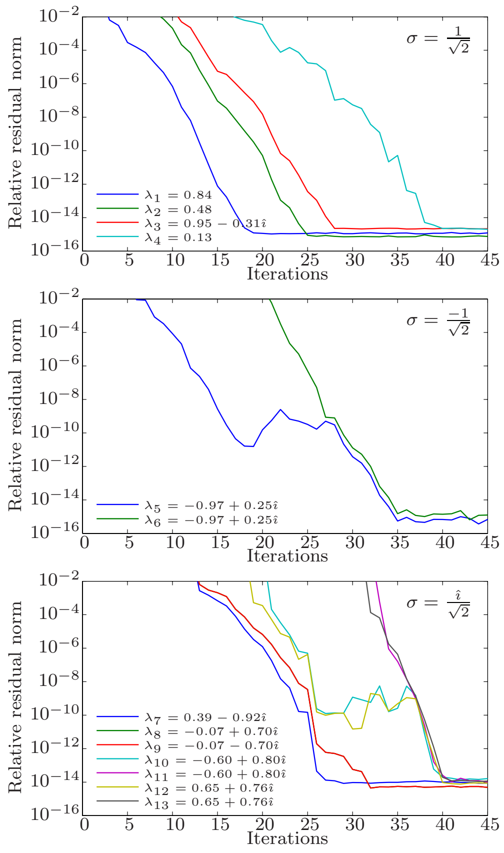


FIG. 4. (Color online) Convergence behavior of the Krylov algorithm for the Al(100) electrode at $E=3$ eV. The figures show the residual norm as a function of iterations for Ritz pairs that satisfy $0.1 \leq |\lambda| \leq 1+\epsilon$, in the case of shift-and-invert transformations to $\pm 1/\sqrt{2}$ and $\hat{\imath}/\sqrt{2}$, respectively.

In Fig. 4 we present curves showing the history of the residual norms for the wanted eigenpairs in each of the separate shift-and-invert Arnoldi procedures. We show only the 45 first iterations since this number is enough for convergence in all cases. Also, only residuals for eigenpairs corresponding to right-going modes are displayed.

The top figure of Fig. 4 illustrates the results from applying the shift $\sigma = 1/\sqrt{2}$ and shows that the Arnoldi procedure determines four different Ritz pairs with individual convergence curves. Comparing with the corresponding polar plot in Fig. 3 (top left), we observe a fifth eigenvalue $(\lambda = 0.95 + 0.31\hat{\imath})$ located inside the valid quarter slice. This fifth eigenvalue represents a left-going mode and is thus discarded in step 10 of Algorithm II. We also see by comparison with Fig. 3 that the eigenpair with eigenvalues furthest from the current shift (the cross) in the complex plane, in this case λ_4, is the slowest to converge.

The middle figure of Fig. 4 shows the convergence of the two Ritz pairs that are covered by the Arnoldi procedure with $\sigma=-1/\sqrt{2}$ and correspond to right-going modes in the present example. We note that λ_5 and λ_6 are nearly multiple eigenvalues, and that the behavior of the residual norms, where one eigenpair is available many iterations before its counterpart, is typical in such a case. Here, in particular, we see that eigenvalue λ_5 is determined to an accuracy of $\sim 10^{-11}$ after 18 iterations before λ_6 even shows up as a Ritz value of the projected eigenproblem. This indicates that λ_5 and λ_6 must be identical to around ten significant digits, and that they cannot be distinguished in our Arnoldi procedure before this accuracy is achieved. Without additional mechanisms to deal with multiple eigenvalues this then implies an upper bound condition on the value of the tol parameter.

The bottom figure of Fig. 4 shows the residual norm history of the remaining seven Ritz pairs required in the current example. These are determined by the Arnoldi procedure with imaginary shift $\sigma=\hat{i}/\sqrt{2}$ and correspond to filled dots in the bottom polar plot of Fig. 3 which represent right-going modes. We observe that the eigenvalue closest to σ, here denoted by λ_8, constitutes a complex conjugate pair together with λ_9, and that these have exactly the same residual norm curve (indistinguishable in the figure), although they are obtained separately as individual Ritz pairs in the algorithm.

In all residual norm figures, we see the trend that the eigenvalues located far from the position of the shift are slow to converge. This suggests that eigenvalues located in the vicinity of the intersections between the unit circle and the dividing lines of the four quarter slices will be the most difficult to determine since they are furthest from the corresponding shifts. The maximum distance from such an eigenvalue to σ is $1/\sqrt{2}$, which is the same as from σ to the origin. This raises concern whether the many unwanted eigenvalues close to the origin can become dominant compared to the wanted border eigenvalues. Fortunately, this is not the case because the unwanted eigenvalues close to the origin are clustered and therefore easy to represent in the Krylov subspace with only a few iterations.[40] We observe this in practice, e.g., from the bottom figure of Fig. 4, where the Ritz pair corresponding to λ_{12}, which lies close to the worst-case position on the unit circle, initially converges only slightly slower than the Ritz pair for λ_8 positioned right next to the shift.

V. NUMERICAL EXAMPLES

To illustrate the accuracy and practical aspects of the proposed Krylov subspace method we present transmission calculations for a metal-device-metal system that has been widely studied in the literature. In addition, we compute the current through this system at 1 and 2 V biases, and use the parameter $\lambda_{\min}$ to investigate the significance of the evanescent modes in obtaining the correct currents. Last, we apply the method to evaluate the self-energy matrices of a variety of electrodes (different types and sizes) and compare the actual measured CPU times[47] with those required by conventional methods.

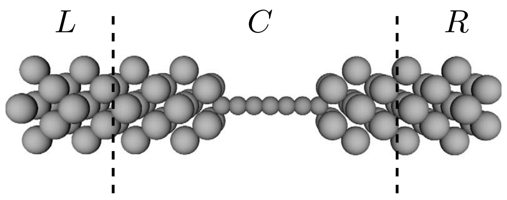


FIG. 5. Schematic illustration of the Al(100)-C7-Al(100) two-probe system.

A. Carbon wire between aluminum electrodes

To demonstrate the applicability of the proposed Krylov subspace method, we first consider carbon chains coupled to metallic electrodes, which have been investigated in detail recently.[1,5,6] Carbon atomic wires are interesting conductors since the equilibrium conductance of short monatomic chains varies with their length in an oscillatory fashion. We will examine the two-probe system shown in Fig. 5 corresponding to a straight wire of seven carbon atoms attached to Al(100) electrodes (lattice constant 4.05 Å). This structure exhibits a local maximum in the oscillatory conductance since it represents an odd-numbered C chain.[5] In our configuration, we fix the C-C distance to 2.5 bohrs and the distance between the ends of the carbon chain and the first plane of Al atoms at 1.0 Å. We use single-ζ basis sets for both types of atoms. The considered Al(100) electrode unit cell consists of 18 atoms in four layers with identical unit cells for the left and right electrodes. Notice that we do not use any symmetry properties of the metallic electrode to reduce the lateral size of the cells (as done, e.g., in Ref. 17) but rather use the full size matrices in Algorithm II. The same system has been studied by Brandbyge *et al.*[1]

We apply the proposed Krylov subspace method to calculate the self-energy matrices $\mathbf{\Sigma}_L$ and $\mathbf{\Sigma}_R$ of the left and right electrodes for a range of energies $E\in[-4\ \text{eV}, 4\ \text{eV}]$ and for different choices of the parameter $\lambda_{\min}$. The self-energy matrices are then used in the evaluation of the corresponding transmission coefficients $T(E)$.

Figure 6 presents the results for bias voltages $V_b=0$, 1, and 2 V in three cases of $\lambda_{\min}$. These significant bias settings are chosen for benchmarking and comparison reasons. The (black) full curves corresponding to $\lambda_{\min}=0.1$ reproduce the transmission spectra obtained in Ref. 1 (for 0 and 1 V) exactly except for the peak at $E=3.63$ eV (for 0 V), which is probably due to finer sampling in our work. In addition, we have calculated the similar curve with the full sets of electrode modes and the results are indistinguishable from those with the setting $\lambda_{\min}=0.1$ (and therefore not displayed in the figure). We note this as quantitative verification that the exclusion of the rapidly decaying evanescent modes is plausible in our setup.

We also see in Fig. 6 that the curves for the parameter $\lambda_{\min}$ set to 0.1 [black (full)] and 0.5 [red (dashed)] are almost identical, which indicates that the vast majority of the evanescent modes (those satisfying $|\lambda|<0.5$) have very little influence on $T(E)$ in the energy regime considered. However, when $\lambda_{\min}$ is set to 0.99 [blue (dotted curves)], in which case only propagating modes and very close to propagating modes are included in the evaluation of self-energy matrices,

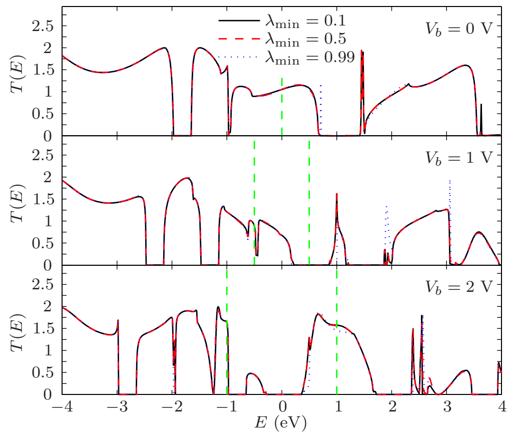


FIG. 6. (Color online) Transmission spectrum of the Al(100)-C7-Al(100) system for different bias voltages V_b. The self-energy matrices used in the $T(E)$ calculations have been obtained at the Γ point by the proposed Krylov subspace method with parameter $\lambda_{\min}$ at several settings: 0.1 [black (full) curve], 0.5 [red (dashed) curve], and 0.99 [blue (dotted) curve]. The bias windows are indicated by the vertical dashed lines.

there are several noticeable deviations from the other curves. Also inside the bias windows and especially for V_b=2 V, the disregard of the evanescent modes produces errors in the obtained transmission coefficients $T(E)$.

The deviations become even more evident in Fig. 7, where the current is displayed as a function of the parameter $\lambda_{\min}$ for nonzero bias voltages. As the value of $\lambda_{\min}$ is increased from around 0.5 to 1, the computed current I starts to depart significantly from the correct value. Therefore, we anticipate that at least some slowly decaying evanescent modes must be taken into account in order to describe the transmission properties of the Al(100)-C7-Al(100) system. Moreover, we see that this can be achieved in a rigorous and systematic fashion by selecting $\lambda_{\min}$ appropriately when using the proposed Krylov subspace method to calculate the self-energy matrices.

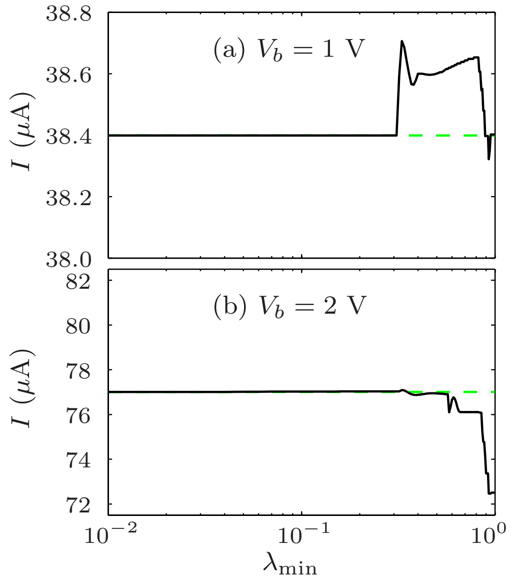


FIG. 7. (Color online) Current as a function of the parameter $\lambda_{\min}$ used by the Krylov subspace method for the Al(100)-C7-Al(100) system with applied bias voltages V_b= (a) 1 and (b) 2 V. The correct currents obtained by conventional methods are $I \approx 38.4$ and ≈ 77.0 μA, respectively, indicated here by the green (dashed) lines.

TABLE I. CPU times in seconds for computing the left self-energy matrix $\mathbf{\Sigma}_L$ at 20 different energies E between −2 and 2 eV for selected electrode types and matrix sizes N. The parameter $\lambda_{\min}$ was set to 0.1.

Electrode type	Size	2^n iterative	DGEEV	Krylov
Li[a]	16	0.1	0.0	0.0
Fe[b]	54	4.2	2.3	0.6
Al(100)[c]	72	4.9	3.3	0.8
Al(100)[c]	128	27.9	17.5	3.6
Au(111)[d]	243	167.2	73.7	11.5
(2,2) CNT[e]	64	3.6	2.4	0.7
(4,4) CNT[e]	128	26.0	14.4	2.9
(8,8) CNT[e]	256	208.8	118.8	17.0
(12,12) CNT[e]	384	608.4	373.6	45.6
(16,16) CNT[e]	512	1230.0	1403.9	121.5
(20,20) CNT[e]	640	1542.3	1125.7	148.0

[a]Measurements from transmission calculations for ideal Li system.
[b]Measurements from transmission calculations for Fe-MgO-Fe; see geometry description in Ref. 10.
[c]Measurements from transmission calculations for Al(100)-C7-Al(100) described in this work (see also Ref. 1).
[d]Measurements from transmission calculations for Au(111)-BDT-Au(111); see, e.g., description in Ref. 11.
[e]Measurements from transmission calculations for ideal armchair (n,n) carbon nanotubes; see, e.g., description in Ref. 4.

B. CPU run times

In this section we focus on the typical savings in the computational time that can be achieved when computing the self-energy matrices $\mathbf{\Sigma}_L$ and $\mathbf{\Sigma}_R$ with the proposed Krylov subspace method. We will compare run times directly with some conventional schemes usually applied in electron transport calculations. Our aim is to illustrate a significant speedup in calculating the self-energy matrices. This is of interest in future efforts to model much larger systems, and, in particular, for electrode unit cells that do not have any lateral symmetry properties.

Table I presents the profiling results when applying three different methods to calculate the same left self-energy matrix $\mathbf{\Sigma}_L$ for common types of electrodes and various matrix sizes N. In every case we consider only the Γ point and use single-ζ basis sets, except for Au(111) where a double-ζ-polarized set is used. Since the computational cost can vary significantly with E, the seconds listed represent the accumulated time of 20 independent calculations at equidistant energies in the interval $E \in [-2 \text{ eV}, 2 \text{ eV}]$. We focus on the

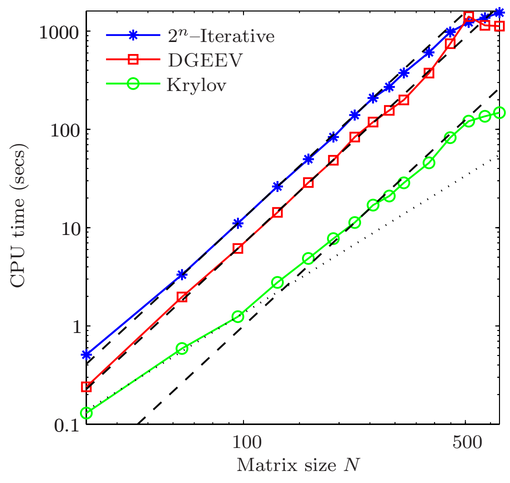


FIG. 8. (Color online) CPU times for computing the left self-energy matrix $\mathbf{\Sigma}_L$ plotted as a function of the size N of $\mathbf{\Sigma}_L$ for a range of armchair (n,n) CNT electrodes, where $n=1,\ldots,20$. The dotted and dashed lines indicate $O(N^2)$ and $O(N^3)$ computational complexity, respectively.

profiling for general electrode configurations and do not use lattice symmetries to reduce the order of the unit cells to elementary size even when this is possible.[17]

In the third column of Table I the run times to compute the correct self-energy matrices with the widely used iterative scheme of López Sancho *et al.*[26] are displayed. As the error in $\mathbf{\Sigma}_L$ obtained by this technique is reduced by $1/2^n$ after n iterations (we denote this method as 2^n iterative), it generally converges in $n \sim 22$ steps. In addition, run times for the conventional eigenvalue approach to evaluating the self-energy matrices, in which a standard eigensolver is used to determine the full set of modes, are presented in the fourth column. For this version, we simply substituted part of our Krylov subspace algorithm (steps 1–9 of Algorithm II) with the state-of-the-art LAPACK routine DGEEV.[47] In the last column the time required by the proposed Krylov subspace method is shown. In all cases of the latter the parameter $\lambda_{\min}$ was set to 0.1.

From the profiling results in Table I we see that the computational time of the Krylov subspace method is significantly reduced compared with the presently widely used 2^n-iterative technique. Also the conventional eigensolver scheme using DGEEV is typically faster than the 2^n-iterative algorithm [the exception for the (16,16) carbon nanotube (CNT) is related to cache usage[48]]. A comparison of the timings in the last two columns verifies that the cost to evaluate the self-energy matrices from only the few most important modes of the electrodes, as in our Krylov subspace method, is in general much lower than required by a direct eigensolver to determine all possible modes.

In order to illustrate the computational complexity of the methods we show the CNT run times as a function of the matrix size N in a logarithmic plot in Fig. 8. Clearly, all methods have $O(N^3)$ complexity; however, the Krylov subspace method initially follows the typical $O(N^2)$ complexity of the Arnoldi procedure[49] until the cost of the shift-and-invert operations becomes dominant. For $N>500$ we observe effects due to more and sometimes less favorable cache usage. Overall, we see that the Krylov subspace method is fastest by an order of magnitude for all but the smallest cases.

It is important to point out that the obtained self-energy matrices $\mathbf{\Sigma}_L$ are in all cases applied in a subsequent transmission calculation of $T(E)$ for the two-probe systems indicated in Table I, and the results then checked against those of the conventional methods [the resulting transmissions $T(E)$ are identical for the three methods in all cases of E to at least three decimals]. Furthermore, the setting of the parameter $\lambda_{\min}$ to 0.1 yields self-energy matrices evaluated from all the modes that have phases λ satisfying $0.1<|\lambda|<1+\epsilon$. This is more than adequate for obtaining correct results to an accuracy of three decimals for all the systems considered in this section. In practice, the parameter $\lambda_{\min}$ can often be selected >0.1 if lower accuracy in the $T(E)$ calculation is satisfactory, and this would show off the approach as even faster.

VI. CONCLUSIONS

In conclusion, we have developed an efficient and robust Krylov subspace method for evaluating the self-energy matrices that are required in electron transport calculations of nanoscale devices. The method exploits the observation that only the propagating and slowly decaying evanescent modes in the electrodes are computationally significant for determining the transmission coefficients when the system is appropriately set up.

The proposed method is based on the Arnoldi procedure and applies carefully chosen shift-and-invert spectral transformations to enhance the convergence toward the wanted interior eigenpairs that correspond to significant modes. We have investigated the convergence properties and shown that the accuracy and efficiency are mainly controlled by two parameters: the tolerance tol to be satisfied by of the relative residuals of the obtained Ritz values and the parameter $\lambda_{\min}$ that implicitly sets the number of modes taken into account.

In Sec. V we tested the Krylov subspace method on a metal-device-metal system and compared it to conventional methods. The applications show that the proposed method can be applied to calculate the transmission characteristics in a rigorous and systematic fashion and that the basic assumption of only including selective solutions in the electrode self-energy matrix is valid for many two-probe systems. The overall saving in computational time achieved by the Krylov subspace method is significant and in most cases more than an order of magnitude in comparison with conventional methods.

ACKNOWLEDGMENTS

The authors would like to thank J. Taylor and the people at Atomistix for helpful discussions. This work was supported by the Danish Council for Strategic Research (NABIIT) under Grant No. 2106-04-0017, "Parallel Algorithms for Computational Nano-Science."

APPENDIX A: COMPUTATIONAL DETAILS

1. Fast transmission calculation

We give the numerical steps to efficiently evaluate $T(E)$ via Eqs. (1) and (2). From the outset, the computational costs are reduced by taking into account that the self-energy matrices are nonzero only in the corner blocks, that is,

$$G_C = \begin{pmatrix} \overline{H}_1 - \Sigma_1^L & \overline{H}_{1,2} & & & \\ \overline{H}_{1,2}^\dagger & \overline{H}_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \overline{H}_{n-1} & \overline{H}_{n-1,n} \\ & & & \overline{H}_{n-1,n}^\dagger & \overline{H}_n - \Sigma_n^R \end{pmatrix}^{-1}, \tag{A1}$$

where the self-energy blocks are numbered similarly to the Hamiltonian blocks. We then select a given diagonal block k and define self-energy matrices for every layer of the system, as[50–52]

$$\Sigma_i^L = \overline{H}_{i-1,i}^\dagger (\overline{H}_{i-1} - \Sigma_{i-1}^L)^{-1} \overline{H}_{i-1,i}, \quad -\infty < i \leq k, \tag{A2}$$

$$\Sigma_i^R = \overline{H}_{i,i+1} (\overline{H}_{i+1} - \Sigma_{i+1}^R)^{-1} \overline{H}_{i,i+1}^\dagger, \quad k \leq i < \infty, \tag{A3}$$

which can be used to recursively evaluate the self-energy matrices Σ_k^L and Σ_k^R when the matrices Σ_1^L and Σ_n^R (or Σ_0^L and Σ_{n+1}^R of the semi-infinite electrodes) are available. The kth block of the Green's function matrix is now given by

$$G_{k,k} = (\overline{H}_k - \Sigma_k^L - \Sigma_k^R)^{-1}, \tag{A4}$$

which corresponds to inverting the block of smallest size in the system, if k is chosen accordingly. Finally Eq. (2) is applied in a simplified version

$$T(E) = \mathrm{Tr}\{\Gamma_k^L G_{k,k} \Gamma_k^R G_{k,k}^\dagger\}, \tag{A5}$$

where the relation $G_{k,k}^a = (G_{k,k}^r)^\dagger$ between the advanced (a) and retarded (r) Green's functions is used [$G^a = (G^r)^\dagger$ is valid when E is real, since H is Hermitian and $\Sigma^a = (\Sigma^r)^\dagger$; see Ref. 13].

2. Generalization to complex Hamiltonian matrices and k-point sampling

In the Krylov subspace method presented in this paper we have assumed that the electrode Hamiltonian matrices are real in order to simplify the computational procedures. We now discuss the steps required to handle the case of complex H_L and $\overline{H}_{L,L}$, which is the case, e.g., when applying k-point sampling (Algorithm II works only for the Γ point).

As noted in Sec. III B, the assumption of real $\overline{H}_L$ and $\overline{H}_{L,L}$ leads to simplifications with the shift-and-invert operations:

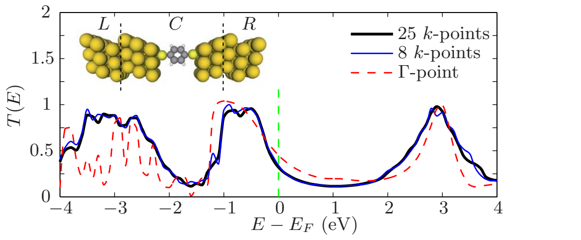


FIG. 9. (Color online) Transmission spectrum of the Au(111)-BDT-Au(111) system for different k-point samplings and V_b=0. The self-energy matrices used in the $T(E)$ calculations have been obtained by the generalized Krylov subspace method with parameter $\delta_{\min}$=0.1.

First, we may consider only right-going modes (λ, c_0) with $|\lambda| \leq 1$ since the left-going modes are uniquely related as (λ^{-1}, c_0), and, second, we can use the spectral transformation T in Eq. (18) to determine the wanted eigenpairs for the two imaginary shifts $\sigma = \pm \hat{i}/\sqrt{2}$ simultaneously and in real arithmetic.

In order to generalize the Krylov subspace method to complex Hamiltonian matrices, it is thus necessary to determine the left-going modes satisfying $1 \leq |\lambda| \leq \lambda_{\min}^{-1}$ (i.e, located outside the unit circle) directly, since there is no general relation to the right-going modes (we note that it is advantageous to change the shift positions to be outside the unit circle, although this is not necessary for good convergence). Furthermore, we must abandon the T matrix and perform two independent shift-and-invert operations for $\sigma = \pm \hat{i}/\sqrt{2}$. It is clear that all this is now done in complex arithmetic and that the extra shift required will make the general algorithm a little more expensive (as shown in Sec. V B, the LU factorization required for each shift-and-invert operation is the dominant cost of our approach).

We have implemented the generalization and can illustrate its applicability by converging the transmission spectrum of the benzene di-thiol (BDT) molecule coupled to gold (111) surfaces in Fig. 9 by 3×3 and 7×7 k-point sampling of the Monkhorst type.[53] The calculation setup used is exactly the same as in Ref. 11 and the results can be confirmed.[3,11] Also, we have computed $T(E)$ for each E and k with self-energy matrices of both the 2^n-iterative method and the Krylov subspace method and checked that the results are identical to within three decimals. The CPU times required for, e.g., the 3×3 curve (eight k points) were 167 and 32 min for the two methods, respectively, while the Γ-point curve takes 2.7 min with Algorithm II. We conclude that the generalized Krylov subspace algorithm is, in this case, 1.5 times slower (per k point) than the real matrix version presented in Sec. III but still more than five times faster than the commonly used 2^n-iterative approach.

*hhs@imm.dtu.dk

[1] M. Brandbyge, J.-L. Mozos, P. Ordejón, J. Taylor, and K. Stokbro, Phys. Rev. B **65**, 165401 (2002).

[2] M. Di Ventra, S. T. Pantelides, and N. D. Lang, Phys. Rev. Lett. **84**, 979 (2000).

[3] S. V. Faleev, F. Léonard, D. A. Stewart, and M. van Schilfgaarde, Phys. Rev. B **71**, 195422 (2005).

[4] H. S. Gokturk, in *Proceedings of the Fifth IEEE Conference on Nanotechnology*, 2005, Vol. 2, pp. 677–680.

[5] N. D. Lang and P. Avouris, Phys. Rev. Lett. **84**, 358 (2000).

[6] B. Larade, J. Taylor, H. Mehrez, and H. Guo, Phys. Rev. B **64**, 075420 (2001).

[7] A. Nitzan and M. A. Ratner, Science **300**, 1384 (2003).

[8] P. Pomorski, C. Roland, and H. Guo, Phys. Rev. B **70**, 115408 (2004).

[9] M. A. Reed, C. Zhou, C. J. Muller, T. P. Burgin, and J. M. Tour, Science **278**, 252 (1997).

[10] M. Stilling, K. Stokbro, and K. Flensberg, Mol. Simul. **33**, 557 (2007).

[11] K. Stokbro, J.-L. Mozos, P. Ordejon, M. Brandbyge, and J. Taylor, Comput. Mater. Sci. **27**, 151 (2003).

[12] M. Büttiker, Y. Imry, R. Landauer, and S. Pinhas, Phys. Rev. B **31**, 6207 (1985).

[13] S. Datta, *Electronic Transport in Mesoscopic Systems* (Cambridge University Press, Cambridge U.K., 1995).

[14] Y. Meir and N. S. Wingreen, Phys. Rev. Lett. **68**, 2512 (1992).

[15] P. A. Khomyakov and G. Brocks, Phys. Rev. B **70**, 195402 (2004).

[16] P. A. Khomyakov, G. Brocks, V. Karpan, M. Zwierzycki, and P. J. Kelly, Phys. Rev. B **72**, 035450 (2005).

[17] K. Xia, M. Zwierzycki, M. Talanana, P. J. Kelly, and G. E. W. Bauer, Phys. Rev. B **73**, 064420 (2006).

[18] K. S. Thygesen and K. W. Jacobsen, Phys. Rev. B **72**, 033401 (2005).

[19] T. Ando, Phys. Rev. B **44**, 8017 (1991).

[20] P. S. Krstić, X.-G. Zhang, and W. H. Butler, Phys. Rev. B **66**, 205319 (2002).

[21] D. H. Lee and J. D. Joannopoulos, Phys. Rev. B **23**, 4997 (1981).

[22] S. Sanvito, C. J. Lambert, J. H. Jefferson, and A. M. Bratkovsky, Phys. Rev. B **59**, 11936 (1999).

[23] T. Shimazaki, H. Maruyama, Y. Asai, and K. Yamashita, J. Chem. Phys. **123**, 164111 (2005).

[24] J. Velev and W. Butler, J. Phys.: Condens. Matter **16**, R637 (2004).

[25] F. Guinea, C. Tejedor, F. Flores, and E. Louis, Phys. Rev. B **28**, 4397 (1983).

[26] M. P. Lopez Sancho, J. M. Lopez Sancho, J. M. L. Sancho, and J. Rubio, J. Phys. F: Met. Phys. **15**, 851 (1985).

[27] W. E. Arnoldi, Q. Appl. Math. **9**, 17 (1951).

[28] M. N. Kooper, H. A. van der Vorst, S. Poedts, and J. P. Goedbloed, J. Comput. Phys. **118**, 320 (1995).

[29] K. Meerbergen and D. Roose, IMA J. Numer. Anal. **16**, 297 (1996).

[30] N. Nayar and J. M. Ortega, J. Comput. Phys. **108**, 8 (1993).

[31] Z. Bai and Y. Su, SIAM J. Matrix Anal. Appl. **26**, 640 (2005).

[32] L. Hoffnung, R.-C. Li, and Q. Ye, Linear Algebr. Appl. **415**, 52 (2006).

[33] U. B. Holz, G. H. Golub, and K. H. Law, SIAM J. Matrix Anal. Appl. **26**, 498 (2004).

[34] Q. Ye, Appl. Math. Comput. **172**, 818 (2006).

[35] First-principles DFT calculations are done with the commercial software package ATOMISTIX TOOLKIT 2.0. We use norm-conserved pseudopotentials for the core electrons and the local density approximation for the exchange-correlation potential (Ref. 1). More details about the software can be found on the company website (www.atomistix.com).

[36] P. N. C. Caroli, R. Combescot, and D. Saint-James, J. Phys. C **4**, 916 (1971).

[37] M. B. Nardelli, Phys. Rev. B **60**, 7828 (1999).

[38] A brief explanation for this is that, since the boundary layers of the C region in our setup are given by principal electrode layers, the evanescent modes that decay very fast do not “survive” the propagation through these layers and therefore do not give any components outside the sets $\{\phi_k^+\}$ and $\{\phi_k^-\}$ at the boundaries of C.

[39] H. H. B. Sørensen, D. E. Petersen, S. Skelboe, P. C. Hansen, and K. Stokbro (unpublished).

[40] L. N. Trefethen and D. Bau, *Numerical Linear Algebra* (SIAM, Philadelphia, 1997).

[41] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide* (SIAM, Philadelphia, 2000).

[42] Ronald B. Morgan and M. Zeng, Linear Algebr. Appl. **415**, 96 (2006).

[43] Z. Jia, J. Comput. Math. **17**, 257 (1999).

[44] F. Tisseur and K. Meerbergen, SIAM Rev. **43**, 235 (2001).

[45] B. N. Parlett and Y. Saad, Linear Algebr. Appl. **88–89**, 575 (1987).

[46] M. E. Hochstenbach and H. A. van der Vorst, SIAM J. Sci. Comput. **25**, 591 (2003).

[47] All computations in this work were done on a Sun ULTRASPARC IV dual-core CPUs (1350 MHz/8 MB L2-cache). We use the vendor-supplied Sun Performance Library that includes platform-optimized versions of LAPACK routines.

[48] For the armchair (16,16) CNT electrode ($N=512$) the call to DGEEV produces an extremely high number of L2 cache misses, many more than for the larger (18,18) CNT electrode ($N=576$). This causes the very poor run times of the DGEEVmethod for this particular electrode.

[49] G. W. Stewart, *Matrix Algorithms* (SIAM, Philadelphia, 2001).

[50] E. M. Godfrin, J. Phys.: Condens. Matter **3**, 7843 (1991).

[51] D. E. Petersen, H. H. B. Sørensen, S. Skelboe, P. C. Hansen, and K. Stokbro, J. Comput. Phys. **227**, 3174 (2008).

[52] S. Y. Wu, J. Cocks, and C. S. Jayanthi, Phys. Rev. B **49**, 7957 (1994).

[53] H. J. Monkhorst and J. D. Pack, Phys. Rev. B **13**, 5188 (1976).

Appendix C

Article: Efficient Wave Function Matching Approach for Quantum Transport Calculations

Article submitted to Physical Review B (preprint on arXiv)

Listed in references as [3].

Efficient wave function matching approach for quantum transport calculations

Hans Henrik B. Sørensen[*] and Per Christian Hansen

Informatics and Mathematical Modelling,

Technical University of Denmark, Bldg. 321, DK-2800 Lyngby, Denmark

Dan Erik Petersen, Stig Skelboe, and Kurt Stokbro

Department of Computer Science, University of Copenhagen,

Universitetsparken 1, DK-2100 Copenhagen, Denmark

(Dated: April 28, 2008)

Abstract

The Wave Function Matching (WFM) technique has recently been developed for the calculation of electronic transport in quantum two-probe systems. In terms of efficiency it is comparable with the widely used Green's function approach. The WFM formalism presented so far requires the evaluation of all the propagating and evanescent bulk modes of the left and right electrodes in order to obtain the correct coupling between device and electrode regions. In this paper we will describe a modified WFM approach that allows for the exclusion of the vast majority of the evanescent modes in all parts of the calculation. This approach makes it feasible to apply iterative techniques to efficiently determine the few required bulk modes, which allows for a significant reduction of the computational expense of the WFM method. We illustrate the efficiency of the method on a carbon nanotube field-effect-transistor (FET) device displaying band-to-band tunneling and modeled within the semi-empirical Extended Hückel theory (EHT) framework.

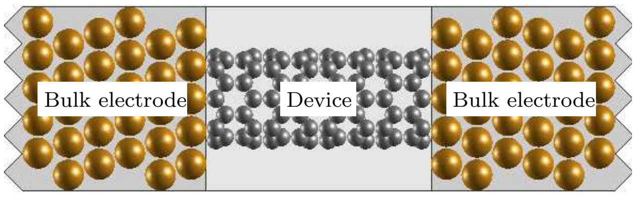


FIG. 1: (Color online) Schematic illustration of a nano-scale two-probe system in which a device is sandwiched between two semi-infinite bulk electrodes.

I. INTRODUCTION

Quantum transport simulations have become an important theoretical tool for investigating the electrical properties of nano-scale systems.[1–5] The basis for the approach is the Landauer-Büttiker picture of coherent transport, where the electrical properties of a nano-scale constriction is described by the transmission coefficients of a number of one-electron modes propagating coherently through the constriction. The approach has been used successfully to describe the electrical properties of a wide range of nano-scale systems, including atomic wires, molecules and interfaces.[6–15] In order to apply the method to semiconductor device simulation, it is necessary to handle systems comprising many thousand atoms, and this will require new efficient algorithms for calculating the transmission coefficient.

Our main purpose in this paper is to give details of a method we have developed, based on the WFM technique,[16–18] which is suitable for studying electronic transport in large-scale atomic two-probe systems, such as large carbon nanotubes or nano-wire configurations.

We adopt the many-channel formulation of Landauer and Büttiker to describe electron transport in nano-scale two-probe systems composed of a left and a right electrode attached to a central device, see Fig. 1. In this formulation, the conduction $\mathcal{G}$ of incident electrons through the device is intuitively given in terms of transmission and reflection matrices, $\mathbf{t}$ and $\boldsymbol{r}$, that satisfy the unitarity condition $\mathbf{t}^\dagger\mathbf{t} + \boldsymbol{r}^\dagger\boldsymbol{r} = \mathbf{1}$ in the case of elastic scattering. The matrix element t_{ij} is the probability amplitude of an incident electron in a mode i in the left electrode being scattered into a mode j in the right electrode, and correspondingly r_{ik} is the probability of it being reflected back into mode k in the left electrode. This simple interpretation yields the Landauer-Büttiker formula[3]

$$\mathcal{G} = \frac{2e^2}{h}\mathrm{Tr}[\mathbf{t}^\dagger\mathbf{t}], \tag{1}$$

which holds in the limit of infinitesimal voltage bias and zero temperature.

To our knowledge, the WFM schemes presented so far in the literature requires the evaluation of all the Bloch and evanescent bulk modes of the left and right electrodes in order to obtain the correct coupling between device and electrode regions. The reason for this is that it requires the complete set of bulk modes to be able to represent the proper reflected and transmitted wave functions. In this paper we will describe a modified WFM approach that allows for the exclusion of the vast majority of the evanescent modes in all parts of the calculation. The primary modification can be pictured as a simple extension of the central region with a few principal electrode layers. In this manner, it becomes advantageous to apply iterative techniques for obtaining the relatively few Bloch modes and slowly decaying evanescent modes that are required. We have developed such an iterative method in Ref. 19, which allows for an order of magnitude reduction of the computational expense of the WFM method in practice.

In this work, the proper analysis of the modified WFM approach is presented. The accuracy of the method is investigated and appropriate error estimates are developed. As an illustration of the applicability of our WFM scheme we consider a 1440 atom CNTFET device of 14 nm in length. We calculate the zero-bias transmission curves of the device under various gate voltages and reproduce previously established characteristics of band-to-band tunneling.[20] We compare directly the results of the modified WFM method to those of the standard WFM method for quantitative verification of the calculations.

The rest of the paper is organized as follows. The WFM formalism used to obtain $\mathbf{t}$ and $\boldsymbol{r}$ is introduced in Sect. II. In Sect. III we present our method to effectively exclude the rapidly decaying evanescent modes from the two-probe transport calculations. Numerical results are presented in Sect. IV. and the paper ends with a short summary and outlook.

II. FORMALISM

In this section we give a minimal review of the formalism and notation that is used in the current work in order to determine the transmission and reflection matrices $\mathbf{t}$ and $\boldsymbol{r}$. This WFM technique has several attractive features compared to the widely used and mathematically equivalent Green's function approach.[1,2] Most importantly, the transparent Landauer picture of electrons scattering via the central region between Bloch modes of the electrodes is retained throughout the calculation. Moreover, WFM allows one to consider the significance of each available mode individually in order to achieve more efficient numerical procedures to obtain $\mathbf{t}$ and $\boldsymbol{r}$.

A. Wave function matching

The WFM method is based upon direct matching of the bulk modes in the left and right electrode to the scattering wave function of the central region. For the most part this involves two major tasks; obtaining the bulk electrode modes and solving a system of linear equations. The available modes in the left and right electrodes are the solutions from the corresponding ideal electrodes. These solutions can be characterized as either propagating or evanescent (exponentially decaying) modes but only the propagating modes contribute to $\mathcal{G}$ in Eq. (1). We may write $\mathcal{G} = (2e^2/h)T$, where

$$T = \sum_{kk'} |t_{kk'}|^2 \tag{2}$$

is the total transmission and the sum is limited to propagating modes k and k' in the left and right electrode, respectively. Notice, however, that the evanescent modes are still needed in order to obtain the correct matrix elements $t_{kk'}$. We will discuss this matter in Sect. III C.

We assume a tight-binding setup for the two-probe systems in which the infinite structure is divided into principal layers numbered $i = -\infty, \ldots, \infty$ and composed of a finite central (C) region containing the device and two semi-infinite left (L) and right (R) electrode regions, see Fig. 2. The wave function is $\boldsymbol{\psi}_i(\boldsymbol{x}) = \sum_j^{m_i} c_{i,j}\chi_{i,j}(\boldsymbol{x} - \boldsymbol{X}_{i,j})$ in layer i, where $\chi_{i,j}$ denotes localized non-orthogonal atomic orbitals and $\boldsymbol{X}_{i,j}$ are the positions of the m_i orbitals in layer i. We represent $\boldsymbol{\psi}_i(\boldsymbol{x})$ by a column vector of the expansion coefficients, given by $\boldsymbol{\psi}_i = [c_{i,1}, \ldots, c_{i,m_i}]^T$, and write the wave function $\boldsymbol{\psi}$ extending over the entire system as $\boldsymbol{\psi} = [\boldsymbol{\psi}_{-\infty}^{\mathrm{T}}, \ldots, \boldsymbol{\psi}_{\infty}^{\mathrm{T}}]^{\mathrm{T}}$. We also assume that the border layers 1 and n of the central region are always identical to a layer of the connecting electrodes.

We refer the reader to Refs. 16–18 for details on how to employ WFM to our setup. Here and in the rest of this paper, we will use the following notation for the key elements: The matrices $\boldsymbol{\Phi}_L^{\pm} = [\boldsymbol{\phi}_{L,1}^{\pm}, \ldots, \boldsymbol{\phi}_{L,m_L}^{\pm}]$ contain in their columns the full set of m_L left-going ($-$) and m_L right-going ($+$) bulk modes $\boldsymbol{\phi}_{L,k}^{\pm}$ of the left electrode, and the diagonal matrices $\boldsymbol{\Lambda}_L^{\pm} = \mathrm{diag}[\lambda_{L,1}^{\pm}, \lambda_{L,2}^{\pm}, \ldots, \lambda_{L,m_L}^{\pm}]$ hold the corresponding Bloch factors.[29] If trivial modes with

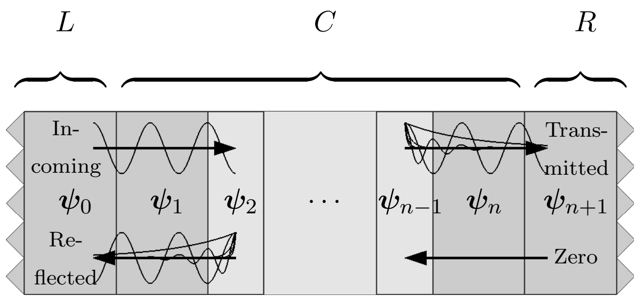


FIG. 2: (Color online) Schematic representation of WFM applied to layered two-probe systems, where the central device region, consisting of layers $i = 1, \ldots, n$, is attached to left and right semi-infinite electrodes. The incoming propagating mode from the left electrode is scattered in the central region and ends up as reflected and transmitted superpositions of propagating and evanescent modes.

$|\boldsymbol{\phi}^+_{L,k}| = \mathbf{0}$ or $|\boldsymbol{\phi}^-_{L,k}| = \infty$ occur they are simply rejected. We assume that all the evanescent bulk modes are (state-)normalized $\boldsymbol{\phi}^{\pm\dagger}_{L,k}\boldsymbol{\phi}^{\pm}_{L,k} = 1$, while all the Bloch bulk modes are flux-normalized[30] $\boldsymbol{\phi}^{\pm\dagger}_{L,k}\boldsymbol{\phi}^{\pm}_{L,k} = d_L/v^{\pm}_{L,k}$, where $v^{\pm}_{L,k}$ are the group velocities[15,21] and d_L is the layer thickness. Similarly for the right electrode the matrices $\boldsymbol{\Phi}^{\pm}_R$ and $\boldsymbol{\Lambda}^{\pm}_R$ are formed. We can then define the Bloch matrices[17] as $\boldsymbol{B}^{\pm}_L = \boldsymbol{\Phi}^{\pm}_L\boldsymbol{\Lambda}^{\pm}_L(\boldsymbol{\Phi}^{\pm}_L)^{-1}$ and $\boldsymbol{B}^{\pm}_R = \boldsymbol{\Phi}^{\pm}_R\boldsymbol{\Lambda}^{\pm}_R(\boldsymbol{\Phi}^{\pm}_R)^{-1}$. The system of linear equations for $\boldsymbol{\psi}_C$ is subsequently written as

$$(E\boldsymbol{S}_C - \boldsymbol{H}_C)\boldsymbol{\psi}_C = \boldsymbol{b}, \tag{3}$$

where E is the energy, $\boldsymbol{\psi}_C = [\boldsymbol{\psi}_1^{\mathrm{T}}, \ldots, \boldsymbol{\psi}_n^{\mathrm{T}}]^{\mathrm{T}}$ is the central region wave function, and $\boldsymbol{S}_C$ and

$$\boldsymbol{H}_C = \begin{pmatrix} \boldsymbol{H}_1 + \boldsymbol{\Sigma}_L & \boldsymbol{H}_{1,2} & & & \\ \boldsymbol{H}^{\dagger}_{1,2} & \boldsymbol{H}_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \boldsymbol{H}_{n-1} & \boldsymbol{H}_{n-1,n} \\ & & & \boldsymbol{H}^{\dagger}_{n-1,n} & \boldsymbol{H}_n + \boldsymbol{\Sigma}_R \end{pmatrix} \tag{4}$$

are the tight-binding overlap and Hamiltonian matrices of the central region. The right-hand side source term $\boldsymbol{b} = [\boldsymbol{b}_1^{\mathrm{T}}, \mathbf{0}^{\mathrm{T}}, \ldots, \mathbf{0}^{\mathrm{T}}]^{\mathrm{T}}$ is specified by the expression

$$\boldsymbol{b}_1 = -(\bar{\boldsymbol{H}}^{\dagger}_{0,1} + \boldsymbol{\Sigma}_L\boldsymbol{B}^{+}_L)\boldsymbol{\psi}_0, \tag{5}$$

where $\boldsymbol{\psi}_0$ is the incoming wave function (notice that this source term is defined for layer 1 and not layer 0, as is the case in Refs. 18 and 17). In Eq. (5) we have introduced the overline notation $\bar{\boldsymbol{H}}_i \equiv E\boldsymbol{S}_i - \boldsymbol{H}_i$ and $\bar{\boldsymbol{H}}_{i,j} \equiv E\boldsymbol{S}_{i,j} - \boldsymbol{H}_{i,j}$ (also used below) to enhance the readability.

The matrices $\boldsymbol{\Sigma}_L$ and $\boldsymbol{\Sigma}_R$ are the left and right self-energy matrices. We stress that for the current setup, these matrices are *identical* to the self-energy matrices introduced in the Green's function formalism[1] (to within an infinitesimal imaginary shift of E), and may be evaluated by well-known recursive techniques[22,23] or, more conveniently for WFM, in terms of the Bloch matrices:[16,17]

$$\boldsymbol{\Sigma}_L = \bar{\boldsymbol{H}}_{0,1}^{\dagger}(\bar{\boldsymbol{H}}_1 + \bar{\boldsymbol{H}}_{0,1}^{\dagger}(\boldsymbol{B}_L^{-})^{-1})^{-1}\bar{\boldsymbol{H}}_{0,1}, \tag{6}$$

and

$$\boldsymbol{\Sigma}_R = \bar{\boldsymbol{H}}_{n,n+1}(\bar{\boldsymbol{H}}_n + \bar{\boldsymbol{H}}_{n,n+1}\boldsymbol{B}_R^{+})^{-1}\bar{\boldsymbol{H}}_{n,n+1}^{\dagger}, \tag{7}$$

For notational simplicity in the following sections, we leave out the implied subscripts L or R, indicating the left or right electrode, whenever the formalism is the same for both (e.g, for symbols $m, \lambda_k, \boldsymbol{\phi}_k, \boldsymbol{\Phi}^{\pm}, \boldsymbol{\Lambda}^{\pm}, \boldsymbol{B}^{\pm}, \boldsymbol{\Sigma}$, etc.).

B. Transmission and reflection coefficients

As a final step we want to determine the $\mathbf{t}$ and $\boldsymbol{r}$ matrices from the boundary wave functions $\boldsymbol{\psi}_1$ and $\boldsymbol{\psi}_n$ that have been obtained by solving Eq. (3).

When the incoming wave $\boldsymbol{\psi}_0$ is specified to be the kth right-going mode $\boldsymbol{\phi}_{L,k}^{+}$ of the left electrode, we can evaluate the kth column of the transmission matrix $\mathbf{t}_k$ by solving

$$\boldsymbol{\Phi}_R^{+}\mathbf{t}_k = \boldsymbol{\psi}_n, \tag{8}$$

where $\boldsymbol{\Phi}_R^{+}$ is the $m_R \times m_R$ column matrix holding the right-going bulk modes of the right electrode (and here assumed to be non-singular). Similarly the kth column of the reflection matrix $\boldsymbol{r}_k$ is given by

$$\boldsymbol{\Phi}_L^{-}\boldsymbol{r}_k = \boldsymbol{\psi}_1 - \lambda_{L,k}^{+}\boldsymbol{\phi}_{L,k}^{+}, \tag{9}$$

where $\boldsymbol{\Phi}_L^{-}$ holds the left-going bulk modes of the left electrode. The flux normalization ensures that $\mathbf{t}^{\dagger}\mathbf{t} + \boldsymbol{r}^{\dagger}\boldsymbol{r} = \mathbf{1}$.[1]

TABLE I: CPU times in seconds when using WFM for calculating $\mathbf{t}$ and $\boldsymbol{r}$ at 20 different energies inside $E \in [-2\text{ eV}; 2\text{ eV}]$ for various two-probe systems. The numbers of atoms in the central region (electrode unit cell) are indicated. The four right-most columns show the CPU times spent for computing the electrode bulk modes with DGEEV and in this work vs. solving the central region linear systems in Eq. (3) and the system with two extra principal layers on each side.

System	Atoms	Eq. (3)	Eq. (3)($l = 2$)	DGEEV	This work
Fe–MgO–Fe	27(6)	0.8	0.9	1.3	1.1
Al–C×7–Al	74(18)	0.4	0.6	3.6	1.6
Au–DTB–Au	102(27)	8.1	13.5	91.0	28.2
Au–CNT(8,0)×1–Au	140(27)	11.4	16.6	77.6	17.1
Au–CNT(8,0)×5–Au	268(27)	45.3	50.3	83.6	17.8
CNT(8,0)–CNT(8,0)	192(64)	7.0	11.9	129.0	19.4
CNT(4,4)–CNT(8,0)	256(64|64)	7.2	12.4	121.5	21.0
CNT(5,0)–CNT(10,0)	300(40|80)	24.7	31.5	113.3	22.6
CNT(18,0)–CNT(18,0)	576(144)	172.2	225.5	1362.2	253.3

III. EXCLUDING EVANESCENT MODES

The most time consuming task of the WFM method is to determine the electrode modes, which requires solving a quadratic eigenvalue problem.[16] As examples, see the profiling results listed in Table I, where we have used the method to compute $\mathbf{t}$ and $\boldsymbol{r}$ for a selection of two-probe systems.[31] The CPU timings show that to determine the electrode modes by employing the state-of-the-art LAPACK eigensolver DGEEV is, in general, much more expensive than to solve the system of linear equations in Eq. (3). We expect this trend to hold for larger systems as well. Therefore, in the attempt to model significantly larger devices (thousands of atoms), it is of essential interest to reduce the numerical cost of the electrode modes calculation. We argue that a computationally reasonable approach is to limit the number of electrode modes taken into account, e.g., by excluding the least important evanescent modes. In this section, a proper technique to do this in a rigorous and systematic fashion is presented.

A. Decay of evanescent modes

The procedure to determine the Bloch factors λ_k and non-trivial modes $\boldsymbol{\phi}_k$ of an ideal electrode and subsequently characterize these as right-going (+) or left-going (−) is well described in the literature.[16–18,24] We note that only the obtained propagating modes with $|\lambda_k| = 1$ are able to carry charge deeply into the electrodes and thus enter the Landauer expression in Eq. (2). The evanescent modes with $|\lambda_k| \neq 1$, on the other hand, decay exponentially but can still contribute to the current in a two-probe system, as the "tails" may reach across the central region boundaries.

Consider a typical example of an electrode modes evaluation: We look at a gold electrode with 27 atoms in the unit cell represented by 9 (sp^3d^5) orbitals for each Au-atom. Such a system results in 243 right-going and 243 left-going modes. Fig. 3a shows the positions in the complex plane of the Bloch factors corresponding to the right-going modes (i.e., $|\lambda_k| \leq 1$) for energy $E = -1.5$ eV. We see that there are exactly three propagating modes, which have Bloch factors located on the unit circle. The remaining modes are evanescent, of which many have Bloch factors with small magnitude very close to the origin.

Fig. 3b illustrates how the 243 left-going modes would propagate through 10 successive gold electrode unit cells. The figure shows that the amplitudes of the three propagating modes are unchanged, while the evanescent modes are decaying exponentially. In particular, we note that the evanescent modes with Bloch factors of small magnitude are very rapidly decaying and vanishes in comparison to the propagating modes after only a few layers. In the following, we will exploit this observation and attempt to exclude such evanescent modes from the WFM calculation altogether. Formally this can be accomplished if only the electrode modes $\boldsymbol{\phi}_k$ with Bloch factors λ_k satisfying

$$\lambda_{\min} \leq |\lambda_k| \leq \lambda_{\min}^{-1}, \tag{10}$$

are computed and subsequently taken into account, for a reasonable choice of $0 < \lambda_{\min} < 1$. We adopt Eq. (10) as the key relation to select a particular subset of the available electrode modes (as recently suggested in Ref. 17).

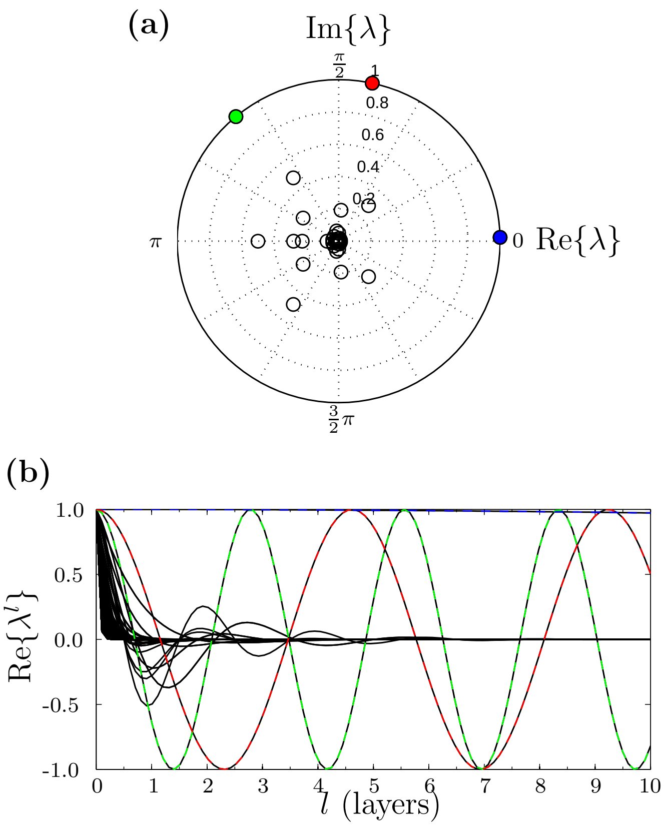


FIG. 3: (Color online) (a) Positions of the Bloch factors λ_k ($|\lambda_k| \leq 1$) obtained for a bulk Au(111) electrode with 27 atoms per unit cell at $E = -1.5$ eV. (b) Amplitudes of the corresponding normalized electrode modes $\boldsymbol{\phi}_k$ moving through 10 layers of the ideal bulk electrode. A total of 243 modes are shown of which 3 are propagating (colored/dashed) and the rest are evanescent (circles/black).

B. Extra electrode layers

We will denote the mode matrices from which the rapidly decaying evanescent modes are excluded via Eq. (10), and also the Bloch matrices and self-energy matrices obtained from these, with a tilde, i.e., as $\tilde{\boldsymbol{\Phi}}^{\pm}$, $\tilde{\boldsymbol{B}}^{\pm}$ and $\tilde{\boldsymbol{\Sigma}}$. The mode matrices holding the excluded modes are denoted by a math-ring accent $\mathring{\boldsymbol{\Phi}}^{\pm}$, so that

$$\boldsymbol{\Phi}^{\pm} = [\tilde{\boldsymbol{\Phi}}^{\pm}, \mathring{\boldsymbol{\Phi}}^{\pm}], \tag{11}$$

is the assumed splitting of the full set. All expressions to evaluate the Bloch and self-energy matrices are unchanged as given in Sect. II (now $(\tilde{\boldsymbol{\Phi}}^{\pm})^{-1}$ merely represents the *pseudo-inverses* of $\tilde{\boldsymbol{\Phi}}^{\pm}$). However, since the column spaces of $\tilde{\boldsymbol{\Phi}}^{\pm}$ are not complete, there is no

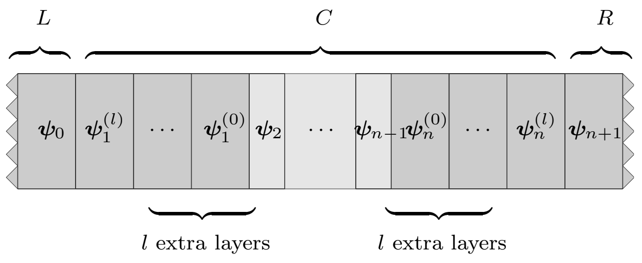


FIG. 4: (Color online) Two-probe system in which the C region boundaries are expanded by l extra electrode layers.

longer any guaranty that WFM can be performed so that the resulting self-energy matrices and, in turn, the solution $\boldsymbol{\psi}_C = [\boldsymbol{\psi}_1^{\mathrm{T}}, \ldots, \boldsymbol{\psi}_n^{\mathrm{T}}]^{\mathrm{T}}$ of the linear system in Eq. (3), are correct. In addition, it is clear that errors can occur in the calculation of $\mathbf{t}$ and $\boldsymbol{r}$ from Eqs. (8) and (9) because the boundary wave functions $\boldsymbol{\psi}_1$ and $\boldsymbol{\psi}_n$ might not be fully represented in the reduced sets $\tilde{\boldsymbol{\Phi}}_R^+$ and $\tilde{\boldsymbol{\Phi}}_L^-$.

As explicitly shown in Refs. 16–18, the key to deriving Eq. (3) is twofold: fixing the layer wave functions coming into the C region (e.g., in our case $\boldsymbol{\psi}_1^+ = \lambda_{L,k}^+ \boldsymbol{\phi}_{L,k}^+$ and $\boldsymbol{\psi}_n^- = \mathbf{0}$) and matching the layer wave functions across the C region boundaries (we remind the reader that our setup has one more electrode layer on both sides of C compared to the setup of Refs. 18 and 17).

The matching is accomplished by using the $\boldsymbol{B}^\pm$ matrices, which by construction propagate the layer wave functions in the bulk electrode,[16–18] i.e.,

$$\boldsymbol{\psi}_j^\pm = (\boldsymbol{B}^\pm)^{j-i} \boldsymbol{\psi}_i^\pm, \tag{12}$$

where subscript L is implied for the left electrode ($i, j \leq 1$), and R for the right electrode ($i, j \geq n$). Notice that the Bloch matrices are always square and also invertible since any trivial electrode are rejected from the outset in the current formalism. When the reduced Bloch matrices $\tilde{\boldsymbol{B}}^\pm$ are used instead of $\boldsymbol{B}^\pm$, however, the possible components of the wave functions outside the column spaces of $\tilde{\boldsymbol{\Phi}}^\pm$ are not properly matched, and the boundary conditions are not necessarily satisfiable.

In order to diminish the errors introduced by excluding evanescent modes, we propose to insert additional electrode layers in the central region, see Fig. 4. As illustrated in the previous section, this would quickly reduce the imprint of the rapidly decaying evanescent modes in the boundary layer wave functions $\tilde{\boldsymbol{\psi}}_1$ and $\tilde{\boldsymbol{\psi}}_n$, which means that the critical components outside the column spaces $\tilde{\mathbf{\Phi}}^{\pm}$ becomes negligible at an exponential rate in terms of the number of additional layers. We emphasize that the inserted layers may be "fictitious" in the sense that they can be accommodated by simple block-Gaussian-eliminations prior to the solving of Eq. (3) for the original system.

The above modements are confirmed by the following analysis. In the particular case, where l extra electrode layers are inserted and the border layers of the C region are identical to the connecting electrode layers, we can write the boundary matching equations as[16–18]

$$\boldsymbol{\psi}_0 = (\tilde{\boldsymbol{B}}_L^+)^{-1}\boldsymbol{\psi}_1^{(l)+} + (\tilde{\boldsymbol{B}}_L^-)^{-1}\boldsymbol{\psi}_1^{(l)-} \tag{13}$$

for the left boundary and

$$\boldsymbol{\psi}_{n+1} = \tilde{\boldsymbol{B}}_R^+\boldsymbol{\psi}_n^{(l)+} + \tilde{\boldsymbol{B}}_R^-\boldsymbol{\psi}_n^{(l)-} \tag{14}$$

for the right boundary, where $\boldsymbol{\psi}_1^{(l)+} = \lambda_{L,k}^+\boldsymbol{\phi}_{L,k}^+$ and $\boldsymbol{\psi}_n^{(l)-} = \mathbf{0}$ are fixed as boundary conditions. We point out, that the l extra layers are bulk layers extending from each electrode and therefore connected via the relation in Eq. (12) for L and R, respectively. Moreover, since the electrode wave functions can always be expanded in the corresponding complete set of bulk modes, we may write

$$\boldsymbol{\psi}_i^{\pm} = \mathbf{\Phi}^{\pm}\boldsymbol{a}_i^{\pm} = [\tilde{\mathbf{\Phi}}^{\pm}, \mathring{\mathbf{\Phi}}^{\pm}]\begin{bmatrix} \tilde{\boldsymbol{a}}_i^{\pm} \\ \mathring{\boldsymbol{a}}_i^{\pm} \end{bmatrix}, \tag{15}$$

where $\boldsymbol{a}_i^{\pm} = [\tilde{\boldsymbol{a}}_i^{\pm T}, \mathring{\boldsymbol{a}}_i^{\pm T}]^T$ are vectors that contain the expansion coefficients and subscript L is implied for the left electrode ($i \leq 1$), and R for the right electrode ($i \geq n$). Thus we may consider the (unfixed) boundary wave functions entering Eqs. (13) and (14), by explicitly writing

$$\boldsymbol{\psi}_1^{(l)-} = (\boldsymbol{B}_L^-)^{-l}\boldsymbol{\psi}_1^- = [\tilde{\mathbf{\Phi}}_L^-, \mathring{\mathbf{\Phi}}_L^-]\begin{bmatrix} (\tilde{\mathbf{\Lambda}}_L^-)^{-l}\tilde{\boldsymbol{a}}_1^- \\ (\mathring{\mathbf{\Lambda}}_L^-)^{-l}\mathring{\boldsymbol{a}}_1^- \end{bmatrix}, \tag{16}$$

and

$$\boldsymbol{\psi}_n^{(l)+} = (\boldsymbol{B}_R^{\pm})^l\boldsymbol{\psi}_n^+ = [\tilde{\mathbf{\Phi}}_R^+, \mathring{\mathbf{\Phi}}_R^+]\begin{bmatrix} (\tilde{\mathbf{\Lambda}}_R^+)^l\tilde{\boldsymbol{a}}_n^+ \\ (\mathring{\mathbf{\Lambda}}_R^+)^l\mathring{\boldsymbol{a}}_n^+ \end{bmatrix}, \tag{17}$$

using the definition $\boldsymbol{B}^{\pm} = \mathbf{\Phi}^{\pm}\mathbf{\Lambda}^{\pm}(\mathbf{\Phi}^{\pm})^{-1}$. This shows that the critical components outside the column spaces of $\tilde{\mathbf{\Phi}}_L^{\pm}$ and $\tilde{\mathbf{\Phi}}_R^{\pm}$ are given by coefficients $(\mathring{\mathbf{\Lambda}}_L^-)^{-l}\mathring{\boldsymbol{a}}_1^-$ and $(\mathring{\mathbf{\Lambda}}_R^+)^l\mathring{\boldsymbol{a}}_n^+$, respectively, and assuming we exclude most rapidly decaying of the evanescent modes according to Eq. (10), that is, $|\lambda_k| > \lambda_{\min}^{-1}$ for the diagonal elements of $\mathring{\mathbf{\Lambda}}_L^-$ and $|\lambda_k| < \lambda_{\min}$ for the diagonal elements of $\mathring{\mathbf{\Lambda}}_R^+$, where $\lambda_{\min}$ is less than 1, these coefficients always decrease as a function of l.

We conclude that WFM with the reduced Bloch matrices $\tilde{\boldsymbol{B}}^{\pm}$ approaches the exact case with $\boldsymbol{B}^{\pm}$ if additional electrode layers are inserted as suggested, and therefore, that the solution $\tilde{\boldsymbol{\psi}}_C$ obtained from Eq. (3) when only a reduced set of bulk modes are used, approaches the correct solution $\boldsymbol{\psi}_C$ accordingly.

C. Accuracy

As pointed out above, the exclusion of some of the evanescent modes from the mode matrices $\Phi^{\pm}$ may introduce errors because the column spaces in $\tilde{\Phi}^{\pm}$ are incomplete. However, it is not obvious to which extend this influences the accuracy of the transmission and reflection calculations from the scattering states solutions $\boldsymbol{\psi}_1^{(l)-}$ and $\boldsymbol{\psi}_n^{(l)+}$. It is therefore important to be able to estimate and monitor the accuracy of the results obtained. We now discuss how this can be done in a systematic fashion in terms of the parameter $\lambda_{\min}$ and the number l of extra electrode layers.

Consider first the accuracy of the transmission matrix $\mathbf{t}$ in the case of the extended two-probe system in Fig. 4. Initially, for a specific incoming mode k, we would like to compare the correct result obtained with the complete set of modes (cf. Eq. (8)),

$$\mathbf{t}_k = \begin{bmatrix} \tilde{\mathrm{t}}_k \\ \mathring{\mathrm{t}}_k \end{bmatrix} = [\tilde{\Phi}_R^+, \mathring{\Phi}_R^+]^{-1} \boldsymbol{\psi}_n^{(l)+}, \tag{18}$$

to the result obtained with the reduced mode matrix (denoted by a prime),

$$\mathbf{t}'_k = \begin{bmatrix} \tilde{\mathrm{t}}'_k \\ \mathring{\mathbf{0}}' \end{bmatrix} = [\tilde{\Phi}_R^+, \mathring{\mathbf{0}}]^{-1} \boldsymbol{\psi}_n^{(l)+}, \tag{19}$$

where $\mathring{\mathbf{0}}'$ and $\mathring{\mathbf{0}}$ represents the zero vector and zero matrix of size $\mathring{m}_R$ and $m_R \times \mathring{m}_R$, respectively.

Notice that the important coefficients in $\mathbf{t}_k$ and $\mathbf{t}'_k$ for transmission calculations are the ones representing the Bloch modes which enters the Landauer-Büttiker formula in Eq. (2). Since these are never excluded they will always be located within the first $\tilde{m}_R$ elements, i.e., in $\tilde{\mathrm{t}}_k$ and $\tilde{\mathrm{t}}'_k$. It then suffices to compare these parts of the transmission matrix which we can do as follows.

From the properties of the pseudo inverse we are able to write the relation

$$(\tilde{\Phi}_R^+)^{-1}[\tilde{\Phi}_R^+, \mathring{\Phi}_R^+] = [\tilde{\mathbf{I}}, (\tilde{\Phi}_R^+)^{-1}\mathring{\Phi}_R^+], \tag{20}$$

where $\tilde{\mathbf{I}}$ is the identity matrix of order equal to the number of included modes $\tilde{m}_R$. Using the expression in Eq. (17) it then follows that

$$\tilde{\mathrm{t}}_k = (\tilde{\mathbf{\Lambda}}_R^+)^l \tilde{\mathbf{a}}_n^+, \tag{21}$$

and

$$\tilde{\mathrm{t}}'_k = \tilde{\mathrm{t}}_k + (\tilde{\Phi}_R^+)^{-1}\mathring{\Phi}_R^+(\mathring{\mathbf{\Lambda}}_R^+)^l \mathring{\mathbf{a}}_n^+, \tag{22}$$

where the $\tilde{\mathrm{t}}'_k$ expression clearly corresponds to the correct coefficients $\tilde{\mathrm{t}}_k$ plus an error term.

We have already established in the previous section that the $(\mathring{\mathbf{\Lambda}}_R^+)^l \mathring{\mathbf{a}}_n^+$ factor in the error term will decrease as a function of l. To ascertain that the total error term also decreases, we look at the 2-norm of $(\tilde{\Phi}_R^+)^{-1}\mathring{\Phi}_R^+$, which satisfies

$$||(\tilde{\Phi}_R^+)^{-1}\mathring{\Phi}_R^+||_2 \leq \mathring{m}_R^{\frac{1}{2}} ||(\tilde{\Phi}_R^+)^{-1}||_2, \tag{23}$$

since $||\mathring{\Phi}_R^+||_2 \leq \mathring{m}_R^{\frac{1}{2}}$ when all evanescent modes are assumed to be normalized. The norm $||(\tilde{\Phi}_R^+)^{-1}||_2$ can be readily evaluated and depends on the set of modes included via the parameter $\lambda_{\min}$ but not on l. We then have that $(\tilde{\Phi}_R^+)^{-1}\mathring{\Phi}_R^+$ is independent of l, and consequently, that the error term in Eq. (22) must decrease as a function of l.

Writing Eq. (22) as $\tilde{\mathbf{t}}'_k = \tilde{\mathbf{t}}_k + \tilde{\boldsymbol{\epsilon}}_k$, where $\tilde{\boldsymbol{\epsilon}}_k$ holds the errors on the coefficients of the kth column, we further obtain that the corresponding total transmission T' obtained from Eq. (2) can be expressed as

$$T' = T + \sum_{kk'} (\tilde{\mathrm{t}}^*_{kk'}\tilde{\epsilon}_{kk'} + \tilde{\epsilon}^*_{kk'}\tilde{\mathrm{t}}_{kk'} + |\tilde{\epsilon}_{kk'}|^2) \tag{24}$$

where T is the exact result and the summation is over the Bloch modes k and k' in the left and right electrode, respectively.

In the attempt to estimate the order of the error term in Eq. (24) we may (as a worst case approximation) take all diagonal elements of $\mathring{\mathbf{\Lambda}}_R^+$ to be equal to the maximum range $\lambda_{\min}$ of Eq. (10), which makes all elements $\tilde{\epsilon}_{kk'}$ proportional to $\lambda_{\min}^l$. Thus we arrive at the simple relation

$$|T' - T| \sim \lambda_{\min}^l + \mathcal{O}\big((\lambda_{\min}^l)^2\big), \tag{25}$$

which shows that the error decreases exponentially in terms of the number of extra layers l.

In practice, Eq. (25) can be adopted as a reasonable order of magnitude estimate of the accuracy of T'. Alternatively, we are able to directly monitor the error arising on the boundary conditions, e.g., in terms of the coefficient vectors $\tilde{\boldsymbol{b}}_{L,k} \equiv (\tilde{\boldsymbol{\Phi}}_R^+)^{-1}(\boldsymbol{\psi}_1^{(l)+} - \lambda_{L,k}^+ \boldsymbol{\phi}_{L,k}^+)$ and $\tilde{\boldsymbol{b}}_{R,k} \equiv (\tilde{\boldsymbol{\Phi}}_R^-)^{-1}\boldsymbol{\psi}_n^{(l)-}$, where $\boldsymbol{\psi}_1^{(l)+}$ and $\boldsymbol{\psi}_n^{(l)-}$ are given by solving Eq. (3). It is clear that $|\tilde{\boldsymbol{b}}_{L,k}| = 0$ and $|\tilde{\boldsymbol{b}}_{R,k}| = 0$ in the case where the boundary conditions are exactly satisfied. Taking into account the similarity in the expressions for $\tilde{\mathbf{t}}'_k$ and $\tilde{\boldsymbol{b}}_{R,k}$ and assuming a similar order of errors in $\boldsymbol{\psi}_n^{(l)+}$ and $\boldsymbol{\psi}_n^{(l)-}$, we would also expect the same order of magnitude of $|\tilde{\boldsymbol{\epsilon}}_k|$ and $|\tilde{\boldsymbol{b}}_{R,k}|$. This suggests another order of magnitude accuracy estimate from Eq. (24), which is straight forward to monitor using the results available with the reduced set of bulk modes:

$$|T' - T| \leq \sum_k (2|\tilde{\mathbf{t}}_k||\tilde{\boldsymbol{\epsilon}}_k| + |\tilde{\boldsymbol{\epsilon}}_k|^2) \sim \sum_k (2|\tilde{\mathbf{t}}_k||\tilde{\boldsymbol{b}}_{R,k}| + |\tilde{\boldsymbol{b}}_{R,k}|^2), \tag{26}$$

where all the vector norms (e.g., $|\tilde{\mathbf{t}}_k|^2 = \sum_{k'} |\tilde{\mathrm{t}}_{kk'}|^2$) are assumed to be taken over the elements corresponding to propagating bulk modes k' only.

Finally, we note without explicit derivation, that similar arguments for the reflection matrix with columns $\tilde{\boldsymbol{r}}'_k = (\tilde{\boldsymbol{\Phi}}_L^-)^{-1}(\boldsymbol{\psi}_1^{(l)-} - \lambda_{L,k}^+ \boldsymbol{\phi}_{L,k}^+)$ and the total reflection coefficient R', as presented above for $\tilde{\mathbf{t}}'_k$ and T', results in the same accuracy expressions for $|R' - R|$ as for $|T' - T|$ in Eqs. (25) and (26), if we substitute $\tilde{\mathbf{t}}_k \to \tilde{\boldsymbol{r}}_k$ and $\tilde{\boldsymbol{b}}_{R,k} \to \tilde{\boldsymbol{b}}_{L,k}$.

D. Example

To end this section, we exemplify the previous discussion quantitatively by looking at the Au(111) electrode described earlier, and assuming a 128 atom (4 unit cells) device of zigzag-(8,0) carbon nano tube (CNT) sandwiched between the gold electrodes, see the configuration in Fig. 1. For energy $E = -1.5$ eV, we have calculated the deviation between the total transmission obtained when all bulk modes are taken into account (T) and when some evanescent modes are excluded (T') as specified with different settings of $\lambda_{\min}$. Deviations are also determined for the corresponding total reflection coefficients (R and R'). Fig. 5 shows the results as a function of l, together with the estimate $\lambda_{\min}^l$ of Eq. (25) and the estimate of Eq. (26) both for the transmission and reflection coefficients, where the higher order terms have been neglected,

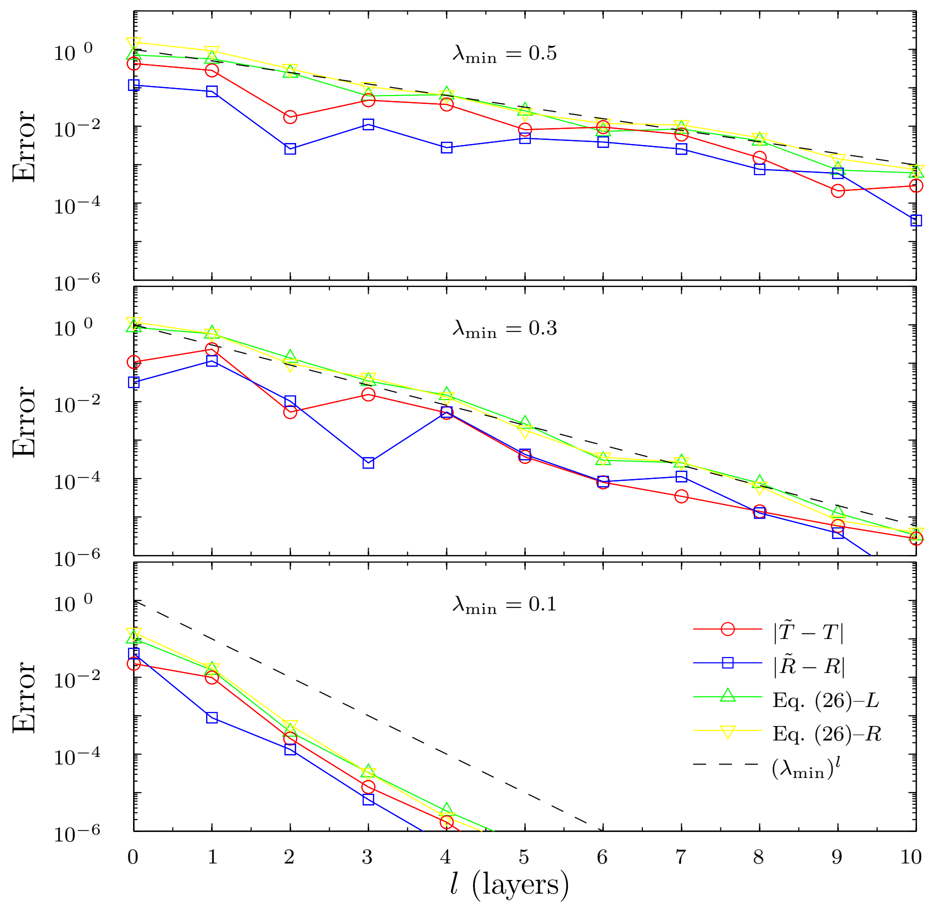


FIG. 5: (Color online) Error (absolute) in the calculated total transmission (solid red lines) and reflection (solid blue lines) coefficients T' and R' as a function of l. The panels show the cases of $\lambda_{\min}$ set to 0.5, 0.3 and 0.1, which corresponds to 3, 14 and 31 Au bulk modes (out of 243, see Fig. 3) taken into account, respectively. The dashed line indicate the theoretical accuracy estimate $\lambda_{\min}^l$. The yellow and green lines show error estimates obtained from Eq. (26).

We observe that the absolute error in the obtained transmission coefficients (red curves) and reflection coefficients (blue curves) are generally decreasing as a function of l, following the same convergence rate as $\lambda_{\min}^l$ (dashed line). Looking closer at results for neighbor l values, we see that the errors initially exhibit wave-like oscillations. This is directly related to the wave form of the evanescent modes that have been excluded (see the propagation of the slowest decaying black curves in Fig. 3(b)), since the representation of these modes in the reduced spaces $\tilde{\mathbf{\Phi}}^{\pm}$ (i.e., the expansion coefficients in $\tilde{\boldsymbol{\epsilon}}_k$) may shift when l is increased. In other words, although the norm of the errors $|\tilde{\boldsymbol{\epsilon}}_k|$ are decreasing as a function of l, the

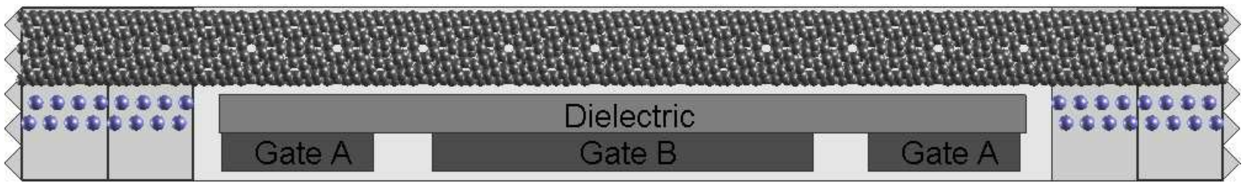


FIG. 6: (Color online) Schematic illustration of a carbon nanotube (8,4) band-to-band tunneling device. The carbon nanotube is positioned on Li surfaces next to an arrangement of three gates.

specific error $\tilde{\epsilon}_{kk'}$ on a given (large) coefficient of $\tilde{t}'_{kk'}$ or $\tilde{r}'_{kk'}$ may increase, which means that the overall error term in Eq. (24) can go up. Fortunately, however, this is only a local phenomenon with the global trend being rapidly decreasing errors.

Consider also the quality of the simple accuracy estimate of $\lambda^l_{\min}$ and the estimates expressed by Eq. (26) for the transmission coefficients (green curves) and reflection coefficients (yellow curves), respectively. For relatively large $\lambda_{\min}$ all estimates are very good. However, for smaller values of $\lambda_{\min}$, only the latter two retain a high quality while the $\lambda^l_{\min}$ estimate tends to be overly pessimistic. It is important to remember, that these estimates are by no means strict conditions but they appear to give very reasonable estimates of the accuracy.

We note in passing, that the results in the top panel of Fig. 5 corresponds to using *only* the propagating Bloch modes in the transmission calculation. Still we are able to compute T and R to an absolute accuracy of three digits by inserting 2×5 extra electrode layers in the two-probe system. This is quite remarkable and shows promise for large-scale systems, e.g., with nano-wire electrodes, for which the total number of evanescent modes available becomes exceedingly great.

IV. APPLICATION

In this section we will apply the developed method to a nano-device consisting of a CNT stretched between to two metal electrodes and controlled by three gates. The setup is inspired by Appenzeller *et al.*[20], and we expect this particular arrangement to be able to display so-called band-to-band (BTB) tunneling, where one observes gate induced tunneling from the valence band into the conduction band of a semi-conducting CNT and vice versa.

We show the configuration of the two-probe system in Fig. 6. The device configuration contains 10 principal layers of a CNT(8,4), having 112 atoms in each. The diameter of the tube and the principal layer length are 8.3 Å and 11.3 Å, respectively. The electrodes consist of CNT(8,4) resting on a thin surfaces of Li, where the lattice constant of the Li layers is stretched to fit the layer thickness of the CNT. The central region of the two-probe system comprises a total of 1440 atoms. An arrangement of rectangular gates are positioned below the carbon nanotube as indicated on the figure. In the plane of the illustration (length $\times$ height) the dimensions are as follows: Dielectric 108 Å $\times$ 5 Å; Gate-A 108 Å $\times$ 5 Å; Gate-B 20 Å $\times$ 5 Å. We set $\epsilon = 4$ for the dielectric constant of the dielectric in order to simulate SiO_2 or Al_2O_3 oxides. All the regions are centered with respect to the electrodes so that the complete setup has mirror symmetry in the length direction. In the direction perpendicular to the illustration the configuration is assumed repeated every 19.5 Å as a super-cell.

We have obtained the density matrix of the BTB device by combining the NEGF formalism with a semi-empirical Extended Hückel model (EHT) using the parameterization of Hoffmann[25]. From the density matrix we calculate Mulliken populations on each atom, and represent the total density of the system as a superposition of Gaussian distributions on each atom properly weighted by the Mulliken population. The width of the Gaussian is chosen to be consistent with CNDO parameters[26]. The electrostatic interaction between the charge distribution and the dielectrics and gates are then calculated and a Hartree like term is included in the Hamiltonian and the set of equations are solved selfconsistently. The resulting selfconsistent EHT model is closely related to the work of Ref.[26], and a detailed description of the model will be presented elsewhere[27].

In order to adjust the charge transfer between the CNT and the Li electrodes we add the term $\delta\epsilon$S to the Li parameters. With an appropriate adjusted value of $\delta\epsilon$ the carbon nanotube becomes n-type doped. We adjust the value such that the average charge transfer from Li to the nanotube at selfconsistency is 0.002 e per carbon atom in the electrode. The Fermi energy is located at -4.29 eV, which is 0.07 eV below the conduction band of the CNT(8,4).

In the following we fix $V_{\text{Gate-A}} = -2.0$ eV and vary the Gate-B potentials in the range $[-2\ \text{eV}, 4\ \text{eV}]$. Note that we report the gate potentials as an external potential on the electrons, and to translate the values into a gate potential of unit Volts the values must be divided with $-e$.

In the left part of Fig. 7 we present the total selfconsistent potential induced by the three gates on the carbon atoms in the CNT over the full extension of the device. The electrostatic potential is shown with two similar curves displaced relative to each other with the energy of the valence band and conduction band edge, respectively. In this way the curves not only represent the electrostatic potential of the device, but also the position of the valence and conduction band edges.

Along with this, in the right part of Fig. 7, we show the corresponding transmission spectrum $T(E)$, for four gate potentials $V_{\text{Gate−B}} = -2.0$ eV, 1.0 eV, 2.0 eV, and 4.0 eV.

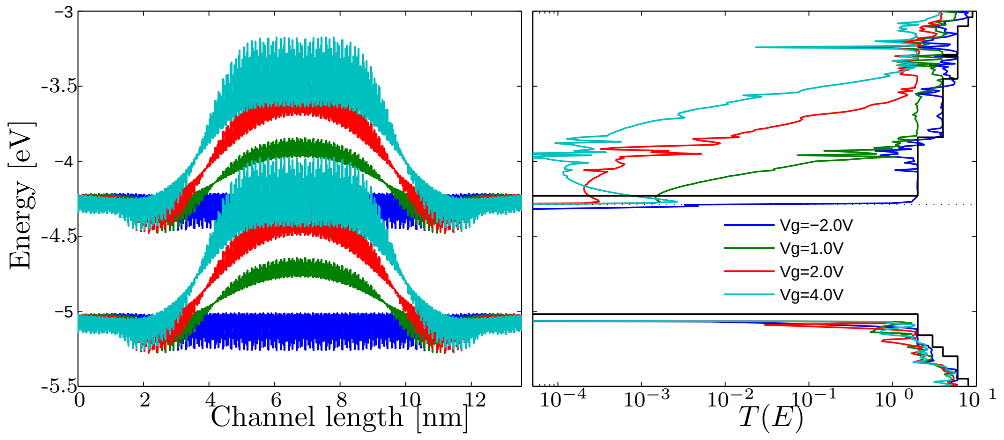


FIG. 7: (Color online) Left panel: Representation of the electrostatic induced shift of the valence and conduction band edges along the length of the device. Right panel: The transmission spectrum for gate potentials $V_{\text{Gate−B}} = -2.0$ eV, 1.0 eV, 2.0 eV and 4.0 eV. The dotted line shows the position of the Fermi level, and the solid line shows the transmission coefficient for an ideal (8,4) CNT.

From Fig. 7 we see how the bands are shifted upwards by an increasing amount as the Gate-B potential is turned up. To begin with, e.g., for $V_{\text{Gate−B}} = 1$ eV, this results in lower conduction since the conduction band bends away from the Fermi level and the Fermi energy electrons need to tunnel through the central region. When the gate voltage is at $V_{\text{Gate−B}} = 2$ eV the valence band almost reaches the conduction band in which case BTB tunneling becomes possible. By increasing the gate voltage further more bands become available for BTB tunneling and the effect is visible as a steady increase in the calculated transmission $T(E)$ just above the Fermi level.

The results for the Fermi level transmission, $T(E_F)$, corresponding to the T = 0 K unit conduction G_0, are displayed with the black curve in Fig. 8. It shows an initial conductance for $V_{\text{Gate−B}} = -2.0$ V of the order of one, a subsequent drop by four orders of magnitude

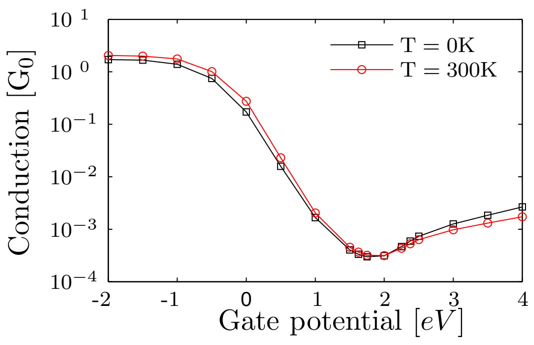


FIG. 8: (Color online) Conduction in units of the conductance quantum G_0 as a function of the Gate-A potential. In the calculations we use a dielectric constant of 4, $V_{\mathrm{Gate-A}} = -2.0$ eV, and vary $V_{\mathrm{Gate-B}}$ from -2.0 eV to 4.0 eV as indicated.

around $V_{\mathrm{Gate-B}} = 2.0$ V, and a final increase of one order of magnitude towards $V_{\mathrm{Gate-B}} =$ 4.0 V. We also display the results for the room temperature T = 300 K conductance(red curve), which can be obtained from

$$\mathrm{G} = \int \mathrm{d}E\ T(E) \frac{e^{(E-E_F)/\mathrm{k}_B\mathrm{T}}}{(1+e^{(E-E_F)/\mathrm{k}_B\mathrm{T}})^2}. \tag{27}$$

The two conduction curves are similar, showing that the device is operating in the tunneling regime rather than the thermal emission regime.

We next briefly comment on the comparison of the simulation with the experiment of Appenzeller *et al.*[20]. In both cases the conduction curves have two branches, which we denote Field Emission (FE) and Band to Band Tunneling (BTB). Initially, the conduction decreases with applied gate potential due to the formation of a barrier in the central region, this is the FE regime. For larger biases the conduction increases again due to BTB tunneling, this is the BTB regime. The experimental device display thermal emission conduction and shows a corresponding subthreshold slope, S, of $k_B T ln(10)/e \approx 60$ mV/dec in the FE regime. The theoretical device, on the other hand, display tunneling conduction and has $S \approx 500$ mV/dec in the FE regime. In the BTB regime, the theoretical device has $S \approx 2000$ mV/dec, while the experimental device show $S \approx 40$ mV/dec.

The very different behaviour is due to the short channel length of the theoretical device. The central barrier has a length of ≈ 5 nm, and at this length the electron can still tunnel through the barrier. We see that the short channel length not only affects the subthreshold slope of the FE regime, but also strongly influence the BTB regime. Work are in progress

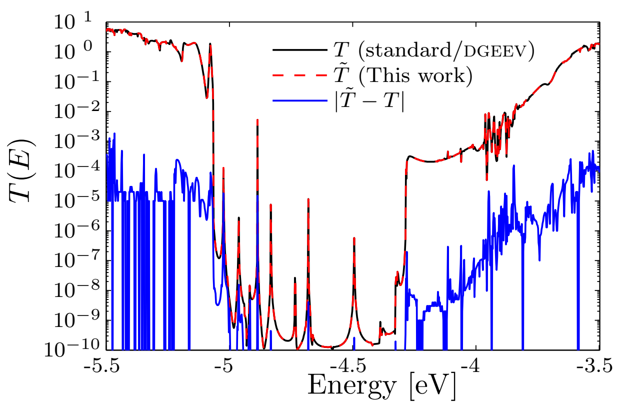


FIG. 9: (Color online) Error (absolute) in the calculated total transmission coefficients $|\tilde{T} - T|$ as a function of energy E for the $V_{\text{Gate-B}} = 2$ V case.

for a parallel implementation of the methodology, which will make it feasible to simulate larger systems, and thereby investigate the transition from the tunnelling to the thermal emission regime.

All the above results have been calculated with the modified WFM method using parameters $\lambda_{\min} = 0.1$ and $l = 1$. Thus, the results presents a non trivial application of the new method. To verify the transmission results in Fig. 9 we present a comparison with the standard WFM method in Fig. 7. The figure shows that the transmissions curves are identical to about three significant digits. The CPU time required for calculating a complete transmission spectrum for Fig. 8 is ($\sim$ 3 hours), while the corresponding calculation presented in Fig. 9 with the standard WFM method took ($\sim$ 35 hours). Thus, the overall time saving achieved with the new method was therefore more than an order of magnitude. The results in Table I indicates that similar timesavings can be expected for other systems with non trivial electrodes.

V. SUMMARY

We have developed an efficient approach for calculating quantum transport in nano-scale systems based on the WFM scheme originally proposed by Ando in reference [16]. In the standard implementation of the WFM method for two-probe systems, all bulk modes of the electrodes are required in order to represent the transmitted and reflected waves in a complete basis. By extending the central region of two-probe system with extra electrode principal layers, we are able to exclude the vast majority of the evanescent bulk modes from the calculation altogether. Our final algorithm is therefore highly efficient, and most importantly, errors and accuracy can be closely monitored.

We have applied the developed WFM algorithm to a CNFET in order to study the mechanisms of band-to-band tunneling. The setup was inspired by reference [20], and the calculation display features also observed in the experiment, however, due to the short channel length the theoretical device operates in the tunneling regime, while the experimental device operates in the thermal emission regime.

By measuring the CPU-times for calculating transmission spectra of the CNFET two-probe system and comparing to cost of the standard WFM method we have observed a speed-up by more than a factor of 10. We see similar speedup for other non-trivial systems. We therefore believe that this is an ideal method to be used with ab initio transport schemes for large-scale simulations.

Acknowledgments

This work was supported by the Danish Council for Strategic Research (NABIIT) under grant number 2106–04–0017, "Parallel Algorithms for Computational Nano–Science".

---

* Electronic address: `hhs@imm.dtu.dk`

[1] S. Datta, *Quantum Transport: Atom to Transistor* (Cambridge University Press, Cambridge, UK, 2005).

[2] M. Brandbyge, J.-L. Mozos, P. Ordejón, J. Taylor, and K. Stokbro, Phys. Rev. B **65**, 165401 (2002).

[3] M. Büttiker, Y. Imry, R. Landauer, and S. Pinhas, Phys. Rev. B **31**, 6207 (1985).

[4] Y. Meir and N. S. Wingreen, Phys. Rev. Lett. **68**, 2512 (1992).

[5] M. A. Reed, C. Zhou, C. J. Muller, T. P. Burgin, and J. M. Tour, Science **278**, 252 (1997).

[6] S. V. Faleev, F. Léonard, D. A. Stewart, and M. van Schilfgaarde, Phys. Rev. B **71**, 195422 (2005).

[7] P. Pomorski, C. Roland, and H. Guo, Phys. Rev. B **70**, 115408 (2004).

[8] H. S. Gokturk, in *Nanotechnology, 2005. 5th IEEE Conference on* (2005), vol. 2, pp. 677–680.

[9] M. Stilling, K. Stokbro, and K. Flensberg, in *NSTI Nanotech 2006 Technical Proceedings* (2006), vol. 3, p. 39.

[10] A. Nitzan and M. A. Ratner, Science **300**, 1384 (2003).

[11] M. Di Ventra, S. T. Pantelides, and N. D. Lang, Phys. Rev. Lett. **84**, 979 (2000).

[12] K. Stokbro, J.-L. Mozos, P. Ordejon, M. Brandbyge, and J. Taylor, Comp. Mat. Sci. **27**, 151 (2003).

[13] N. D. Lang and P. Avouris, Phys. Rev. Lett. **84**, 358 (2000).

[14] B. Larade, J. Taylor, H. Mehrez, and H. Guo, Phys. Rev. B **64**, 075420 (2001).

[15] P. A. Khomyakov and G. Brocks, Phys. Rev. B **70**, 195402 (2004).

[16] T. Ando, Phys. Rev. B **44**, 8017 (1991).

[17] P. A. Khomyakov, G. Brocks, V. Karpan, M. Zwierzycki, and P. J. Kelly, Phys. Rev. B **72**, 035450 (pages 13) (2005).

[18] G. Brocks, V. M. Karpan, P. J. Kelly, P. A. Khomyakov, I. Marushchenko, A. Starikov, M. Talanana, I. Turek, K. Xia, P. X. Xu, et al., Ψ_k-Newsletter **80**, 144 (2007), URL `http://www.psi-k.org/newsletters/News_80/newsletter_80.pdf`.

[19] H. H. B. S. rensen, P. C. Hansen, D. E. Petersen, S. Skelboe, and K. Stokbro, Physical Review B (Condensed Matter and Materials Physics) **77**, 155301 (pages 12) (2008), URL `http://link.aps.org/abstract/PRB/v77/e155301`.

[20] J. Appenzeller, Y.-M. Lin, J. Knoch, and P. Avouris, Phys. Rev. Lett. **93**, 196805 (2004).

[21] N. W. Ashcroft and D. N. Mermin, *Solid State Physics* (Brooks Cole, 1976).

[22] F. Guinea, C. Tejedor, F. Flores, and E. Louis, Phys. Rev. B **28**, 4397 (1983).

[23] M. P. Lopez Sancho, J. M. Lopez Sancho, J. M. L. Sancho, and J. Rubio, J. Phys. F. **15**, 851 (1985).

[24] P. S. Krstić, X.-G. Zhang, and W. H. Butler, Phys. Rev. B **66**, 205319 (2002).

[25] R. Hoffmann, The Journal of Chemical Physics **39**, 1397 (1963), URL `http://link.aip.org/link/?JCP/39/1397/1`.

[26] F. Zahid, M. Paulsson, E. Polizzi, A. W. Ghosh, L. Siddiqui, and S. Datta, J. of Chem. Phys. **123**, 064707 (2005).

[27] K. Stokbro and *et al.*, unpublished.

[28] D. S. Fisher and P. A. Lee, Phys. Rev. B **23**, 6851 (1981).

[29] Bloch's theorem[21] $\boldsymbol{\psi}_i = \lambda_k \boldsymbol{\psi}_{i-1}$ for the ideal electrodes defines the phase factors $\lambda_k \equiv e^{\imath q_k d}$, where q_k is the complex wave number and d is the layer thickness, which are referred to as Bloch factors throughout this paper.

[30] When using the Landauer formula in Eq. (1) it is assumed that the electrode Bloch modes carry unit current in the conduction direction. This can be conveniently accommodated by flux-normalizing the Bloch modes, i.e., $\boldsymbol{\phi}^{\pm}_{L,k} \rightarrow (d_L/v^{\pm}_{L,k})^{\frac{1}{2}} \boldsymbol{\phi}^{\pm}_{L,k}$, in the case of the left electrode.[28]

[31] We should point out that the metallic electrodes in the two-probe systems considered in Table I can be fully described by much smaller unit cells than indicated (often only a few atoms are needed) and therefore the time spend on computing the bulk modes can be vastly reduced in these specific cases. For a general method, however, which supports CNTs, nano wires, etc. as electrodes, the timings are appropriate for showing the overall trend in the computational costs.

References

[1] Dan Erik Petersen, Hans Henrik B. Sørensen, Per Christian Hansen, Stig Skelboe, and Kurt Stokbro. Block tridiagonal matrix inversion and fast transmission calculations. *Journal of Computational Physics*, 2007. 2, 3, 34, 58, 133, 134, 138, 146, 162, 163, 165

[2] Hans Henrik B. Sørensen, Per Christian Hansen, Dan Erik Petersen, Stig Skelboe, and Kurt Stokbro. Krylov subspace method for evaluating the self-energy matrices in electron transport calculations. *Physical Review B*, 77:155301, 2008. 2, 29, 183

[3] Hans Henrik B. Sørensen, Dan Erik Petersen, Per Christian Hansen, Stig Skelboe, and Kurt Stokbro. Efficient wave function matching approach for quantum transport calculations. *arXiv*, (0804.4306v1), 2008. 2, 197

[4] Dan Erik Petersen, Song Li, Eric Darve, Per Christian Hansen, Stig Skelboe, and Kurt Stokbro. A hybrid method for the parallel calculation of the green's function matrix. *in preparation*, 2008. 2, 34

[5] P.J. Knowles and N.C. Handy. A new determinant-based full configuration interaction method. *Chemical Physics Letters*, 111:315–321, 1984. 11

[6] Gian Luigi Bendazzoli and Stefano Evangelisti. A vector and parallel full configuration interaction algorithm. *Journal of Chemical Physics*, 98(4):3141–3150, 1993. 11

[7] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Physical Review*, 136:864–871, 1964. 12, 13, 16

[8] N. David Mermin. Thermal properties of the inhomogeneous electron gas. *Physical Review*, 137(5A):1441–1443, 1965. 13

[9] M. Levy. Universal variational functionals of electron densities, first-order density matrices, and natural spin-orbitals and solution of the n-representability problem. *Proceedings of the National Academy of Sciences of the United States of America*, 76(12):6062–6065, 1979. 13, 16

[10] M. Levy. Electron densities in search of hamiltonians. *Physical Review A*, 26(3):1200–1208, 1982. 13, 16

[11] A. Shimony and H. Feshbach, editors. *Physics as Natural Philosophy*. MIT Press, Cambridge, 1982. 13, 16

[12] E. Lieb. Density functionals for coulomb systems. *International Journal of Quantum Chemistry*, 24:243–277, 1983. 13, 16

[13] R.M. Dreizler and J. da Providencia, editors. *Density Functional Methods in Physics*. Plenum, New York, 1985. 13, 16

[14] R.O. Jones and O. Gunnarsson. The density functional formalism, its applications and prospects. *Reviews of Modern Physics*, 61(3):689–746, 1989. 13, 15, 17

[15] W. Kohn and L.J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140(4A):1133–1138, 1965. 15, 17

[16] W. Kohn. Density functional and density matrix method scaling linearly with the number of atoms. *Physical Review Letters*, 76(17):3168–3171, 1996. 16, 21

[17] E. Prodan and W. Kohn. Nearsightedness of electronic matter. *Proceedings of the National Academy of Sciences of the United States of America*, 102(33):11635–11638, 2005. 16, 21

[18] A.D. Becke. A new mixing of hartree-fock and local density-functional theories. *Journal of Chemical Physics*, 98:1372–1377, 1993. 17

[19] Stefan Kurth, John P. Perdew, and Peter Blaha. Molecular and solid-state tests of density functional approximations: Lsd, ggas, and meta-ggas. *International Journal of Quantum Chemistry*, 75:889–909, 1999. 17

[20] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6:15–50, 1996. 23

[21] Kurt Stokbro, J. Taylor, M. Brandbyge, and H. Guo. Ab-initio non-equilibrium green's function formalism for calculating electron transport in molecular devices. *Lect. Notes Phys.: Intro. Mol. Electronics*, 2005. 23, 29

[22] Supriyo Datta. *Quantum transport: atom to transistor*. Cambridge University Press, 2005. 24, 32

[23] R. Landauer. Spatial variation of currents and fields due to localized scatterers in metallic conduction. *IBM Journal of Research and Development*, 1(3):223, 1957. 25

[24] M. Büttiker, Y. Imry, R. Landauer, and S. Pinhas. Generalized many-channel conductance formula with application to small rings. *Physical Review B*, 31(10):6207–6215, 1985. 25

[25] M.P. López-Sancho, J.M. López-Sancho, J.M.L. Sancho, and J. Rubio. Highly convergent schemes for the calculation of bulk and surface green functions. *Journal of Physics F: Metal Physics*, 15:851–858, 1985. 29

[26] M. Brandbyge, J.L. Mozos, P. Ordejón, J. Taylor, and Kurt Stokbro. Density-functional method for nonequilibrium electron transport. *Physical Review B*, 65:165401, 2002. 31

[27] Kurt Stokbro, J. Taylor, M. Brandbyge, J.L. Mozos, and P. Ordejón. Theoretical study of the nonlinear conductance of di-thiol benzene coupled to au(111) surfaces via thiol and thiolate bonds. *Computational Materials Science*, 27:151–160, 2003. 33

[28] Kurt Stokbro, J. Taylor, and M. Brandbyge. Do aviram-ratner diodes rectify? *Journal of the American Chemical Society*, 125:3674–3675, 2003. 33

[29] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, second edition edition, 1989. 67, 136

[30] John von Neumann. First draft of a report on the edvac. Technical report, Moore School of Electrical Engineering, University of Pennsylvania, 1945. 69

[31] Barry Wilkinson and Michael Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, 1999. 78

[32] Walter Gander and Gene Golub. Cyclic reduction — history and applications. In *Scientific Computing: Proceedings of the Workshop 10-12 March, 1997*, pages 73–85, 1997. 83, 85, 97, 109, 135

[33] Don Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis*, 13(4):484–496, 1976. 85, 109

[34] O. Buneman. A compact non-iterative poisson solver. Technical report, Stanford Univ. Institute for Plasma Research, 1969. 109

[35] B.L. Buzbee, G.H. Golub, and C.W. Nielson. On direct methods for solving poisson's equations. *SIAM Journal on Numerical Analysis*, 7(627-656), 1970. 109

[36] J. Lambiotte and R. Voigt. The solution of tridiagonal linear systems on the cdc star100 computer. *ACM Transactions on Mathematical Software*, 1:308–329, 1975. 109

[37] H.S. Stone. Parallel tridiagonal equation solvers. *ACM Transactions on Mathematical Software*, 1:289–307, 1975. 109

[38] Paul N. Swarztrauber. A parallel algorithm for solving general tridiagonal equations. *Mathematics of Computation*, 33(145):185–199, 1979. 109

[39] Plamen Yalamov and Velisar Pavlov. Stability of the block cyclic reduction. *Linear Algebra and its Applications*, 249:341–358, 1996. 109

[40] J. Appenzeller, Y.-M. Lin, J. Knoch, and Ph. Avouris. Band-to-band tunneling in carbon nanotube field-effect transistors. *Physical Review Letters*, 93(19):196805, 2004. 161

[41] P. Arbenz and M Hegland. The stable parallel solution of general banded linear systems. Technical Report 252, Computer Science Department at ETH Zürich, 1996.

[42] R.W. Hockney. A fast direct solution of poisson's equation using fourier analysis. *Journal of the Association for Computing Machinery*, 8:95–113, 1965.

[43] H.S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the Association for Computing Machinery*, 20:27–38, 1973.

[44] Paul N. Swarztrauber. A direct method for the discrete solution of separable elliptic equations. *SIAM Journal on Numerical Analysis*, 11:1136–1150, 1974.

[45] Elena M. Godfrin. A method to compute the inverse of an n-block tridiagonal quasi-hermitian matrix. *Journal of Physics: Condensed Matter*, 3:7843–7848, 1991.

[46] Hartmut Haug and Antti-Pekka Jauho. *Quantum Kinetics in Transport and Optics of Semiconductors*. Springer, 2008.

[47] D.S. Fisher and P.A. Lee. Relation between conductivity and transmission matrix. *Physical Review B*, 23(12):6851–6854, 1981.

[48] Iain S. Duff and Henk A. van der Vorst. Developments and trends in the paralle solution of linear systems. *Parallel Computing*, 25:1931–1970, 1999.

[49] R.G. Parr and W. Yang. *Density-Functional Theory of Atoms and Molecules*. Oxford University Press, 1989.

[50] Attila Szabo and Neil S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Dover Publications, 1996.

[51] Richard M. Martin. *Electronic structure: basic theory and practical methods*. Cambridge University Press, 2004.