

Datalogi 2B

Bachelor project

*Hardware implementation af parallel løsning af
lineære ligningssystemer*

af Dennis Haney og Martin Leopold

Datalogisk Institut, Københavns Universitet
Forår 2001

Indholdsfortegnelse

1 Forord	1
2 Ligningssystemer	3
2.1 Kredsløbsanalyse	3
2.2 Specielle karakteristika	4
3 Løsning af ligningssystemer	7
3.1 Numeriske metoder	7
3.2 Direkte løsningsmetoder	7
3.3 LU faktorisering	8
3.3.1 Algoritme uden pivotering	10
3.3.2 Blok LU faktorisering	10
3.4 Substitution	12
3.5 Problemer	13
3.5.1 Pivotering og repræsentation	13
4 Arkitektur	15
4.1 Lager	16
4.2 Uniprocessor	17
4.2.1 Én modificeret RISC-kerne	18
4.2.2 VLIW-kerne	19
4.2.3 En vektor processor	19
4.3 Multiprocessor	20
4.3.1 En multiprocessormaskine med fælles lager	21
4.3.2 COTS maskiner	21
4.4 Massiv parallelisering	22
4.4.1 Flere RISC-kerner på én chip	22
4.4.2 Et systolisk array	23
5 Opbygning af systolisk array	27
5.1 Generelle problemer	27
5.2 Lagertilgang	28
5.2.1 Buffer	30

INDHOLDSFORTEGNELSE

5.3	Størrelse	32
5.3.1	Systolisk celle	32
5.3.2	Samlet transistorforbrug	33
6	Ydeevne analyse	35
6.1	Små tætbefolkede ligningssystemer	36
6.2	BBD ligningssystemer	37
6.2.1	Løsning med et systolisk array	38
6.3	Blok tridiagonale ligningssystemer	39
6.3.1	Tilpasning til det systoliske array	41
6.3.2	Parallelisering	41
6.3.3	Udvidelse af systolisk array	43
7	Konklusion	45
7.1	Yderligere arbejde	45
	Appendix	47
A	Litteraturliste	47
B	Synopsis	49
(C)	Output fra AMG (vedlagt)	52
(D)	Ekstrapoleret data(vedlagt)	53

Kapitel 1

Forord

Denne rapport er udarbejdet af Dennis Albert Vous Haney og Martin Leopold i perioden 9/2-2001 til 17/4-2001 som bachelorprojekt i datalogi ved Københavns Universitet. Firmaet Silicide A/S har stillet lokaler og computere til rådighed samt sørget for bespisning.

Projektet omhandler en hardware implementation af en ligningsløser til et lineært ligningssystem. Det vil igennem rapporten blive belyst, hvordan et ligningssystem genereres ud fra et elektronisk kredsløb, hvordan et generelt lineært ligningssystem matematisk løses og hvordan forskellige hardware arkitekturer kan benyttes til løsningen. Dette vil resultere i et udkast til implementation af en speciel arkitektur. Kombineret med specielle matematiske egenskaber af ligningssystemerne vises det, at et lineært ligningssystem ideelt kan løses $O(\log_2(N))$ tid.

Der vil igennem rapporten være en datalogisk tilgangsvinkel til emnet og derfor forventes det, at læseren er bekendt med viden svarende til en datalogisk førstedel samt en rimelig matematisk modenhed.

Under arbejdet med projektet er idéerne fra synopsis (vedlagt som bilag) blevet videreudviklet. Der bliver ikke udelukkende behandlet ligningssystemer fra kredsløbsanalyse, der er heller ikke implementeret en prototype og idéen med at benytte programmet ESACAP til at levere testdata er derfor droppet.

Kapitel 2

Ligningssystemer

I dette kapitel introduceres kort kredsløbsanalyse for senere at bruge de ligningssystemer, som opstår i denne forbindelse. Emnet kredsløbsanalyse er langt større, end det her er antydnet, og der foregår betydelig forskning inden for repræsentation af kredsløb, kredsløbsmodeller, ligningsopstilling og simulation. Mange af de problemstillinger, der findes inden for dette område, har ikke direkte relevans for denne rapport og dækkes derfor ikke. Introduktionen er inspireret af [RD93, kap. 27].

2.1 Kredsløbsanalyse

Essensen i kredsløbsanalyse ligger i modelleringen af kredsløbet. For en given elektrisk komponent kan opstilles en såkaldt overførelsesfunktion, som på passende vis modellerer komponentens opførsel. Der kan vælges forskellige niveauer af detaljering. På laveste niveau simuleres hver enkelt transistor og modstand. På højeste niveau ses på funktionalitet af store blokke, ofte beskrevet i programmeringssproglignende sprog (på DIKU benyttes SIMSYS). For store kredsløb vil det være beregningsmæssigt uoverkommeligt at simulere hver enkelt transistor i hele kredsløbet, derfor kan vælges at blande forskellige niveauer af detaljering således, at simuleringen koncentrerer sig om særlige dele af et kredsløb. Denne rapport befinder sig på nederste niveau ofte omtalt som “transistor level” af indlysende årsager. Egenskaber, som ofte ønskes simuleret på dette niveau, er spændings- og strømkurver som funktion af tid. Disse mål skal afsløre forskellige karakteristika om kredsløbet, det kunne være stigtider eller timing.

Ved hjælp af Kirchoffs strømlov samt Ohms lov kan nu opstilles et sæt knudeligninger for kredsløbet. Metoden til opstilling af knudeligninger er ikke fast, men kan tilpasses de givne omstændigheder. Én metode er den modificerede knude tilgang (Modified nodal approach). Denne metode vælges ofte, da den

håndterer både strøm- og spændingsstyrede komponenter (se [RD93] for detaljeret gennemgang).

Grundlæggende er der to karakteristiske overførselsfunktioner: Lineære (modstande og spændingskilder) og ulineære (dioder, kondensatorer, transistorer og evt. selvinduktanser). For de lineære elementer gælder, at det kan lade sig gøre at opstille et lineært ligningssystem, som modellerer kredsløbet. Tilsvarende kan opstilles et ulineært ligningssystem for kredsløb med ulineære komponenter. Det er ikke muligt at løse ulineære ligningssystemer direkte, derfor skitseres kort en numeriske metode. Der henvises til [WJM88, kap.4] for en mere fyldestgørende beskrivelse. For et givet ulineært system, det kunne være et system fremkommet fra en kredsløbsbeskrivelse, gælder:

$$g(V) = I \quad (2.1)$$

hvor g er en overførselsfunktion, V er spænding og I er strøm. Ligningen kan også skrives som

$$\bar{g}(V) = g(V) - I = 0 : \mathcal{R}^n \rightarrow \mathcal{R}^n \quad (2.2)$$

Altså transformeres problemet til at finde nulpunkter i $\bar{g}(V)$. En metode, som kan benyttes, er Newton-Raphson iteration udvidet til funktioner af flere variable. Essensen i Newton-Raphson er en Taylorudvikling af $\bar{g}(V)$. Den afledte af $\bar{g}(V) : \mathcal{R}^n \rightarrow \mathcal{R}^n$ er kendt som den totalafledte eller jakobimatricen. Som antydnet er Newton-Raphson en iterativ metode, som løser problemet $f(x) = 0$. Metoden kan ud fra et nuværende bud x_c finde et nyt bud x_+ :

$$x_+ = x_c - \left[\frac{\partial f}{\partial x} \right]_{x=x_c}^{-1} f(x_c) \quad (2.3)$$

hvilket kan omskrives til

$$\left[\frac{\partial f}{\partial x} \right]_{x=x_c} (x_c - x_+) = f(x_c) \quad (2.4)$$

som tydeligt er på formen $Ax = b$. Altså kan et ulineært ligningssystem løses som en række lineære ligningssystemer.

2.2 Specielle karakteristika

De ligningssystemer, som fremkommer fra analysen af lineære og ulineære kredsløb, udviser karakteristika, som kan udnyttes under løsningen af ligningssystemerne. Det er klart hvordan disse karakteristika fremkommer i ligningssystemer fra lineære kredsløb, men de samme karakteristika vil også optræde i Newton-Raphson iterationens lineære ligningssystemer. Det ligger uden for denne opgave at beskrive, hvordan disse karakteristika overføres fra det ulineære ligningssystem til de lineære ligningssystemer.

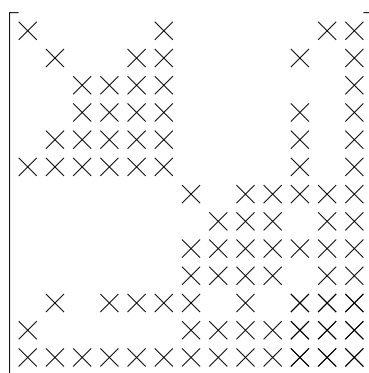
Ligningssystemerne vil altid være strukturelt symmetriske. Modeller som opstiller ligningssystemer ud fra knuder i kredsløbet, som den modificerede knudetilgang fra sidste kapitel, tilføjer informationer om knuderne i par symmetrisk om diagonalen. Det vil sige, at en forbindelse altid har en tilsvarende symmetrisk på den anden side af diagonalen.

Ligningssystemerne er kvadratiske. Idet, der for hver knude i kredsløbet, altid tilføjes et eller flere par indgange i matricen på hver side af diagonalen, vil sidelængderne vokse ens og proportionalt med antallet af knuder.

Ligningssystemerne er tyndtbefolkede. Hver knude har typisk kun forbindelse til få andre knuder, således at der kun vil være få indgange for hver knude i matricerne.

Ligningssystemet har en rand. I et kredsløb vil der ofte være knuder med forbindelser til mange andre knuder - power er en knude, som vil være forbundet til næsten alle andre knuder. Sådanne knuder vil afstedkomme en rand omkring ligningssystemet.

Ligningssystemerne er hierakisk opbygget i blokke. En anden egenskab, som springer i øjnene, er formen af ligningssystemet. Typisk består et stort elektrisk kredsløb, som en chip, af en række store kredsløbsblokke, der internt er tæt forbundet, men som kun er forbundet med hinanden med få forbindelser. Som et eksempel på dette ses kredsløbskortet i en almindelig PC; her vil der være en række chips forbundet med printbaner. Inde i en chip er der et stort antal forbindelser sammenlignet med benforbindelserne til omverdenen. Organiseres ligningssystemerne så naboknuder (i eksemplet: Nabo-chips) ligger nær hinanden i matricerne, opnås et ligningssystem med en meget karakteristisk struktur (se figur 2.1). Denne karakteristiske struktur er hierakisk således, at de enkelte blokke er opbygget på samme måde. Fortsættes der med PC eksemplet vil hver chip være opbygget af en række blokke med forbindelser - det kunne være en ALU. En ALU er igen bygget op af en række forbundne blokke. På denne måde gentager strukturen sig i kredsløbet og dermed også i matricen. En matrix som har denne struktur kaldes ofte for "BBD - Block Bordered Diagonal".



Figur 2.1 – Et BBD ligningssystem kunne være et ligningssystem fremkommet ved kredsløbsanalyse.

Kapitel 3

Løsning af ligningssystemer

Dette kapitel ser overordnet på løsningen af lineære ligningssystemer. Indledningsvis diskuteres anvendeligheden numeriske metoder kort, ligeledes præsenteres Gauss-elimination kort og resten af kapitlet beskriver den ofte anvendte variant LU faktorisering.

Som notation vil summationer med negativt index (f.eks. \sum_1^0) forløbe 0 gange.

3.1 Numeriske metoder

Til løsning af ligningssystemer virker det besnærende at benytte sig af iterative numeriske metoder, der paralleliserer godt, og som ofte er hurtigere end direkte metoder.

For at sikre konvergens indfører de klassiske numeriske metoder uheldigvis begrænsninger på de ligningssystemer, som kan løses. For eksempel kræver Gauss-Siedel metoden at ligningssystemet er diagonaldominant [DW96, side 230]. For generelle ligningssystemer kan det ikke garanteres, at disse specielle egenskaber er opfyldt, således at en af de klassiske numerisk metoder ville kunne anvendes.

Der findes dog specielle numeriske metoder, der dækker specielle områder. [TI97] har en tekst om en løsningsmetode til iterativ løsning af partielle differentialligninger med finite element metoden.

3.2 Direkte løsningsmetoder

Den klassiske algoritme inden for de direkte ligningsløser er Gauss elimination. Gauss elimination beskrives her ikke i detaljer, men en glimrende gennemgang kan findes i [RM93].

Gauss elimination løser ligningssystemer på formen:

$$A\bar{x} = \bar{b} \quad (3.1)$$

hvor A er en kendt matrix, der indeholder koefficienterne til ligningssystemet, \bar{b} er en kendt vektor og \bar{x} er vektoren af ubekendte. Ligningssystemet løses i to trin: Først transformeres 3.1 ved hjælp af rækkeoperationer til et ækvivalent ligningssystem, hvor der gælder at A er transformeret til et øvre triangulært system og \bar{b} er blevet ændret passende. Altså er A transformeret til en matrix \hat{A} , som kun indeholder 0-elementer under diagonalen og \bar{b} til en ny \hat{b} . Dernæst udføres baglæns substitution, hvorved de variable x_i i vektoren \bar{x} løses én ad gangen:

$$\bar{x} = \hat{A}^{-1}\hat{b} \quad (3.2)$$

Der er udviklet et utal af variationer af Gauss eliminationen specialiseret til hver sit formål, hvoraf der kun ses på en enkelt - LU faktorisering. LU betegner en opdeling af A til to nye matricer L og U :

$$A = LU \quad (3.3)$$

L er en matrix, hvor alle indgange over diagonalen består af 1 og tilsvarende har U kun 0 i alle indgange under diagonalen. \bar{x} findes ved at løse det ækvivalente system:

$$LU\bar{x} = \bar{b} \quad (3.4)$$

Algoritmen for løsning af lineære ligningssystemer på formen $A\bar{x} = \bar{b}$ bliver [WJM88, kap. 2.1] altså:

- Faktoriser A til L og U
- Løs $L\bar{z} = \bar{b}$ for \bar{z} (også kendt som fremad substitution)
- Løs $U\bar{x} = \bar{z}$ for \bar{x} (også kendt som baglæns substitution)

Den store fordel ved LU faktorisering fremfor Gauss elimination ligger i, at løse flere ligningssystemer med samme koefficientmatrix (A) og forskellige højresider (\bar{b}). Det kan vises, at antallet af tidkritiske operationer er ens for Gauss elimination og LU faktorisering [WJM88, p.22]. Dermed bliver LU faktorisering altid en ækvivalent eller hurtigere metode end Gauss elimination.

3.3 LU faktorisering

I dette afsnit udledes faktoriseringen af A til L og U . Udledningen er hovedsagligt inspireret af [DW96, kap.4.2]. Der startes med en $n \times n$ matrix A , idet systemet består af n ligninger med n ubekendte. LU faktorisering vil også være

mulig for systemer, som ikke er $n \times n$, men der henvises til anden litteratur for yderligere detaljer. Med matrix A ledes efter

$$L = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \text{ og } U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix} \quad (3.5)$$

som opfylder ligning 3.3. Det kan vises, at ligning 3.5 ikke definerer diagonalelementerne i L og U entydigt. Derfor viser det sig nødvendigt at fastlægge diagonalelementerne i enten L eller U . De skal dog fastlægges til en ikke-0 værdi. Hvis elementerne l_{ii} fastlægges, er det muligt at beregne u_{ii} eller tilsvarende lade u_{ii} bestemme l_{ii} . Et indlysende valg kunne være at lade enten l_{ii} eller u_{ii} være 1, da der, som det vil fremgå senere, spares en division. Hvis $l_{kk} = 1$ kaldes denne faktorisering for Doolittle's faktorisering og tilsvarende er $u_{kk} = 1$ en Crout's faktorisering.

Der ses nu på matrixproduktet af L og U og egenskaberne af de to matricer udnyttes. Det antages, at diagonalelementerne i A er forskellig fra 0, idet det kan vises, at denne antagelse er tilstrækkelig til at sikre, at der findes en LU faktorisering af A [DW96, p.166].

$$a_{ij} = \sum_{s=1}^n l_{is}u_{sj} = \sum_{s=1}^{\min(i,j)} l_{is}u_{sj} \quad (3.6)$$

idet $l_{is} = 0$ for $s > i$ og $u_{is} = 0$ for $s < i$.

For hvert a_{ij} fastlægges altså den j 'te søjle i U og den i 'te række i L . I et givet trin a_{ij} vil alle rækker $1, 2, \dots, i-1$ og alle søjler $1, 2, \dots, j-1$ være bestemt i et foregående trin. Diagonalelementerne i A udtrykkes ud fra matrixproduktet LU ($i = j = k$):

$$a_{kk} = \sum_{s=1}^{\min(k,k)} l_{ks}u_{sk} = l_{kk}u_{kk} + \sum_{s=1}^{k-1} l_{ks}u_{sk} \quad (3.7)$$

Hvis l_{kk} er specificeret kan ligning 3.7 bruges til at beregne u_{kk} , idet a_{kk} kendt og summeringen er kendt (idet den kun indeholder rækker/søjler fra forrige trin). Det samme gør sig gældende for u_{kk} specificeret.

I det k 'te trin er nu u_{kk} og l_{kk} kendt. Derfor kan ligning 3.6 nu bruges til at finde de øvrige elementer; for den k 'te række og den k 'te søjle gælder da:

$$a_{kj} = l_{kk}u_{kj} + \sum_{s=1}^{k-1} l_{ks}u_{sj} \quad (k+1 \leq j \leq n) \quad (3.8)$$

$$a_{ik} = l_{ik}u_{kk} + \sum_{s=1}^{k-1} l_{is}u_{sk} \quad (k+1 \leq i \leq n) \quad (3.9)$$

Hvis $l_{kk} \neq 0$ kan 3.8 bruges til at finde elementerne u_{kj} . Hvis $u_{kk} \neq 0$ kan 3.9 bruges til at finde elementerne l_{ik} .

3.3.1 Algoritme uden pivoting

I dette afsnit udledes to algoritmer til LU faktoriseringen uden brug af pivoting. Pivoting diskuteres i afsnit 3.5.1.

l_{kk} sættes til 1 i det følgende. Det udnyttes, at beregningen af diagonalelementet u_{kk} er identisk med beregningen for de øvrige u_{ij} elementer. Det bemærkes, at der i ligning 3.9 skal gælde $u_{kk} \neq 0$. Den tidligere beskrevne antagelse $a_{ii} \neq 0$ sikrer at dette altid gælder. Ud fra ligning 3.8 og 3.9 findes da de øvrige elementer i L og U, hvilket fører til algoritme 3.1.

I stedet for at summere i hvert i 'te og j 'te trin i algoritme 3.1 viser det sig at en akkumulering af summationerne kan gemmes i A under beregningen (samme idé ses i [WJM88, 2.4]). Denne algoritme vil senere vise sig nyttig. Varianten ses i algoritme 3.2.

LU FAKTORISERING UDEN PIVOTERING	(3.1)
(1) input $n, [a_{ij}]$	
(2) for $k = 1$ to n do	
(3) $l_{kk} = 1$	
(4) for $j = k$ to n do	
(5) $u_{kj} = a_{kj} - \sum_{s=1}^{k-1} l_{ks}u_{sj}$	
(6) end do	
(7) for $i = k+1$ to n do	
(8) $l_{ik} = \frac{a_{ik}}{u_{kk}} - \sum_{s=1}^{k-1} \frac{l_{is}u_{sk}}{u_{kk}}$	
(9) end do	
(10) end do	
(11) output $[l_{ij}], [u_{ij}]$	

3.3.2 Blok LU faktorisering

I stedet for at regne på en søjle/række ad gangen viser det sig nyttigt at LU faktorisere i blokke.

Studerer algoritme 3.2 er det klart, at LU faktoriseringen forløber i 3 trin (se også figur 3.1):

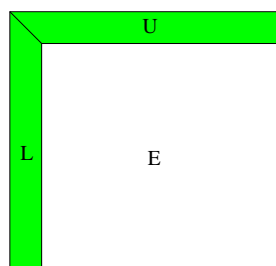
1. Opdater en søjle i L
2. Opdater en række i U
3. Opdater E

Denne idé kan udvides til at LU faktorisere A i blokke af et antal elementer. Skrives definitionen af LU produktet med A opdelt i blokke, er det nemt at se

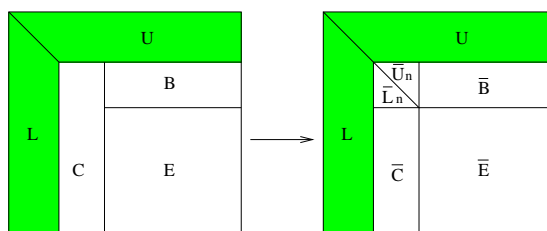
LU FAKTORISERING UDEN PIVOTERING MED AKKUMULERET SUM

(3.2)

```
(1) input n, [aij]  
(2) for k = 1 to n do  
(3)     lkk = 1  
(4)     for i = k+1 to n do  
(5)         lik =  $\frac{a_{ik}}{a_{kk}}$   
(6)     end do  
(7)     for j = k to n do  
(8)         ukj = akj  
(9)     end do  
(10)    for i = k+1 to n do  
(11)        for j = k+1 to n do  
(12)            aij = aij - likukj  
(13)        end do  
(14)    end do  
(15) output [lij],[uij]
```



Figur 3.1 – Algoritme 3.2 skematisk opdatering.



Figur 3.2 – Blok LU faktorisering.

hvordan blok LU faktoriseringen forløber:

$$\begin{aligned}
 A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} &= \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} * \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \\
 &= \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}
 \end{aligned} \tag{3.10}$$

hvor L_{11} og U_{11} findes ved at LU faktorisere A_{11} og de øvrige løses som:

$$\begin{aligned}
 A_{21} = L_{21}U_{11} &\Rightarrow L_{21} = A_{21}U_{11}^{-1} \\
 A_{12} = L_{11}U_{12} &\Rightarrow U_{12} = L_{11}^{-1}A_{12} \\
 A_{22} = L_{21}U_{12} + L_{22}U_{22} &\Rightarrow E = L_{22}U_{22} = A_{22} - L_{21}U_{12}
 \end{aligned}$$

Vælges blokke som på figur 3.2 forløber blok LU faktoriseringen som:

1. Faktoriser C til \bar{L} , \bar{U} , \bar{C}
2. Beregn \bar{B} ved at løse $\bar{L}\bar{B} = B$
3. Opdater $\bar{E} = E - \bar{C}\bar{B}$
4. Faktoriser E, enten rekursivt med samme algoritme eller direkte.

3.4 Substitution

Efter LU faktoriseringen løses de to ligningssystemer $L\bar{z} = \bar{b}$ og $U\bar{x} = \bar{z}$ som:

$$\bar{z} = L^{-1}\bar{b} \tag{3.11}$$

$$\bar{x} = U^{-1}\bar{z} \tag{3.12}$$

Idet både L og U er triangulære, kan metoden kendt som substitutionsmetoden benyttes til at løse 3.11 og 3.12. Substitutionsmetoden udnytter, at L og U er triangulære. Ved at se på strukturen af et triangulært system, er det klart, at ligningen i den ende, som er "spids", kan løses trivielt. Med denne løsning kan den næste ligning med 2 variable løses osv. Altså kan U løses ligning for ligning fra neden. Tilsvarende kan L løses ligning for ligning ovenfra.

Lad L være en $n \times n$ matrix og nedre triangulær. Da løses 3.11 ved fremad substitution som:

$$z_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} z_j \right) / l_{ii} \quad i = 1, \dots, n \quad (3.13)$$

da $l_{ii} = 1$ gælder

$$z_i = b_i - \sum_{j=1}^{i-1} l_{ij} z_j \quad i = 1, \dots, n \quad (3.14)$$

Lad U være en $n \times n$ matrix og øvre triangulær. Ligning 3.12 løses ved baglæns substitution:

$$x_i = \left(z_i - \sum_{j=i+1}^n u_{ij} x_j \right) / u_{ii} \quad i = n, \dots, 1 \quad (3.15)$$

3.5 Problemer

LU algoritmen som præsenteret i sidste afsnit virker med sin enkelthed utrolig tiltrækkende. Desværre er der nogle problemer, når den skal implementeres på en computer:

- Hvis $u_{ii} = 0$ for et givet i bryder processen sammen.
- Da computeren kun har en endelig præcision i sin talrepræsentation, kan det risikeres at præcisionen ikke er så god som antallet af bits (numerisk fejl). Altså at en række af løsningens cifre ligger uden for usikkerheden og kan derfor ikke stoles på.
- Den endelige præcision i computeren kan også afstedkomme fænomenet "numerisk ustabilitet". I værste fald kan fænomenet resultere i, at den beregnede løsning ikke er korrekt. I en Newton-Raphson iteration kan det medføre divergens af iterationsprocessen.

Matricerne fra kapitel 2 er ofte meget store og tyndtbefolkede. Hvis alle elementer opbevares eksplicit, kan der opstå lagerproblemer. Ved at udnytte at matricerne er tyndtbefolkede, kan 0-elementer opbevares implicit. På denne måde opnås pladsbesparelse og der spares beregningstid (idet det er trivielt at regne på 0-elementer i matricerne).

3.5.1 Pivoting og repræsentation

En måde at løse disse problemer ligger i rækkefølgen, hvori operationer udføres - pivoterings rækkefølgen. Ved at ombytte rækkerne, pivotere, på passende vis under LU faktorisering, kan det sikres, at ligningssystemet konvergerer, og at præcisionen bliver optimal.

Når der tales om tyndtbefolkede ligningssystemer, vil naiv Gauss Elimination og LU faktorisering introducere nye ikke 0-elementer i matricerne. For at kunne bevare de tiltalende egenskaber ved de tyndtbefolkede matricer introduceres den optimale pivotordning. Ved pivotordning forstås den rækkefølge, som rækkeoperationerne i Gauss eliminationerne udføres. Den optimale pivotordning er den ordning, som skaber færrest nye ikke 0-elementer.

Uheldigvis kan det at udføre pivotering være meget dyrt. Beregningen af hvilke rækker, som skal byttes om, kan være kompliceret og kræver globalt kendskab til matricen. Medmindre matricen opbevares i en fornuftig repræsentation, kan det være dyrt at bytte om på rækker i matricerne.

Hvis ligningssystemerne stammer fra en Newton-Raphson iteration, og en række ligningssystemer løses med ikke-optimal præcision, vil den pris, som betales være en nedsat konvergenshastighed i iterationen. Afhængig af den valgte løsning kan det måske betale sig helt at undlade pivotering. Hvis løsningen til Newton-Raphson iterationen divergerer, vil det være nødvendigt at pivotere eller lappe på proceduren på anden vis.

Det er vigtigt, at repræsentationen på en effektiv måde understøtter rækkeoperationerne, som kræves af LU faktorisering. Hvis den valgte algoritme udfører pivotering, vil det være en stor fordel at understøtte de operationer som kræves (både læsning og skrivning) effektivt. De basale rækkeoperationer er ombytning, addition med en konstant og multiplikation med en konstant. Ikke overraskende har de forskellige løsningsmetoder hver yderligere krav til at tilgå matricernes elementer. For at lave en effektiv repræsentation til en algoritme, er det derfor nødvendigt at se på i hvilken rækkefølge algoritmerne tilgår elementerne. Effektive repræsentationer er beskrevet i [WJM88, kap. 3.1].

Når 0-elementer repræsenteres implicit, kræves det, at indsættelse af 0-elementer sker på en effektiv måde. Lagres eksempelvis alle ikke 0-elementer fortløbende efter hinanden, vil indsættelse af et ikke 0-element kræve en flytning af alle eller dele af de øvrige elementer. En anden ofte brugt mulighed er at forudberegne hvor i matricen, der under beregningerne vil opstå ikke 0-elementer og reservere disse på forhånd. Ved på forhånd at kende pivoteringsrækkefølgen af matricen kan det forudsiges, hvor der vil opstå nye ikke 0-elementer.

Kapitel 4

Arkitektur

I det foregående kapitel blev løsning af lineære ligningsystemer ud fra et matematisk synspunkt diskuteret. I dette kapitel diskuteres fordele og ulemper ved forskellige hardwareimplementationer af en ligningsløser.

Der vil blive fokuseret på en række nye og gamle idéer indenfor processor udviklingen, som kan være interessante til at løse lineære ligningsystemer. Det er langt fra en udtømmende liste, men den dækker alligevel store dele af området. Senere kommer en mere uddybende beskrivelse af de enkelte arkitekturer, men her en kort oversigt:

- Uniprocessor
 - En modificeret RISC kerne med mange eksekveringsenheder til flydende tal i kernen.
Her benyttes en speciel processor, der ligner en ganske almindelig RISC processor set fra programmørens synspunkt, men det overlades til selve processoren at finde nyttigt arbejde.
 - VLIW-kerne med mange eksekveringsenheder til flydende tal i kernen.
Til forskel fra den modificeret RISC kerne, hvor skeduleringen overlades til selve processoren, er det her oversætterens opgave.
 - En vektor processor.
En vektor processor fungerer, som navnet antyder, ved at arbejde på vektorer af data i stedet for registre. Det er dermed muligt med enkelte instruktioner at arbejde effektivt med store mængder data.
- Multiprocessor
 - En samling almindelige COTS (Commercial Off-The-Shelf) maskiner forbundet via et netværk.

En fattigmandsversion af en multiprocessormaskine. Ved at bruge billig standard hardware opnås mange små maskiner istedet for én stor kraftig.

- En multiprocessormaskine med fælles lager - en supercomputer. For løsning af nogle problemer kan det være en fordel, at alle processorer har lige hurtig adgang til det samme lager.
- Massiv parallelisering
En helt ny teknologi som endnu ikke har set kommerciel anvendelse.
 - Flere RISC-kerner på én chip.
Idéen er, at bygge et antal processorer sammen på en chip i et ultra tæt koblet netværk med en yderst simpel kommunikationsprotokol, således at kommunikationsomkostningerne minimeres.
 - Et systolisk array.
Dette er et specielt bygget mønster af eksekveringsenheder sat sammen på en måde, at de er i stand til at løse et givet problem. Idéen i det systoliske array er at programmere datavejene direkte ind i hardware.

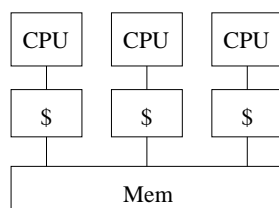
I de følgende afsnit, vil der fokuseres på detaljerne i ovenstående arkitekturer, men først et afsnit om lageret.

4.1 Lager

For løsningen af meget store lineære ligningssystemer er der to faktorer, som gør lagertilgangen til den begrænsende faktor.

Løsning af ligningssystemer er en meget båndbreddeforbrugende proces med en meget regulær tilgang til data. I mange arkitekturer er lagersystemet optimeret til tilgangshastighed af tilfældige data, men i dette tilfælde burde optimeringen være efter båndbredde. Ydermere er det sådan, at arkitekturer ofte har en meget optimeret central beregningsenhed, der er flere faktorer hurtigere end det omkringliggende lagersystem - derfor opnås kun optimal ydelse, hvis der beregnes meget mellem lagertilgange. Løsning af ligningssystemer er ikke en af disse problemer, og derfor ødelægges balancen mellem tilgang til lageret og beregninger. Det skal dog nævnes, at det er muligt at cacheoptimere en implementation, så det aktive datasæt passer i cachen og dermed opnå større udnyttelse af maskinen. En sådan implementation vil naturligvis være platformsføhængig. BLAS implementationen ATLAS[ATINET], er et eksempel på en implementation.

For multiprocessor arkitekturer deler organisationen af lageret sig primært i to lejre: Distribueret og fælles, hver med utallige varianter. Vi ser på 3 ofte brugte varianter:



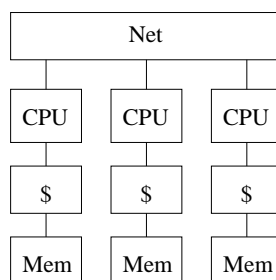
Figur 4.1 – Fælles lager. Alle processorer har samme adresserum og tilgangshastighed. \$ er cache.

- Fælles lager (Figur 4.1). Alle arbejder på det samme lager, og tilgangen er den samme for alle. For lagerkrævende opgaver, som løsning af lineære ligningssystemer, er denne ikke den optimale, når antallet af processorer stiger. Grunden er de stigende omkostninger til håndtering af konsistens og overensstemmelse mellem de forskellige processorer opfattelse af lagerets indehold.
- Distribueret lager (Figur 4.2). Denne model har et hurtigt lager ved hver af de enkelte processorer, men tilgangen til andre dele af lageret end den lokale er meget langsom. Det er derfor hensigtsmæssigt at håndtere udvekslingen af delte elementer eksplicit.
- Dancehall (Figur 4.3). Er en kombination af de to forrige. I denne model er lageret distribueret i række lagerbanke på en sådan måde, at alle processorer har adgang til alle banke. Således kan en bank levere data uafhængigt af de øvrige, så den samlede båndbredde øges væsentligt. Navnet kommer af, at en processor eksklusivt allokerer en lagerbank, som når en mand danser med en kvinde. For multiprocessor systemer kan det antages, at data er konsistent for alle processorer.

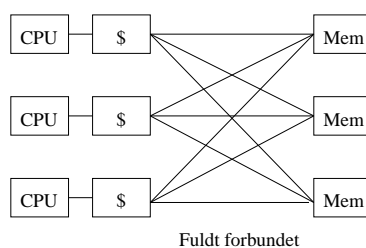
En multiprocessor implementation vil i høj grad afhænge af hvilket lagersystem, der bliver brugt. Varianter af de tre nævnte løsninger bruges i dag kommercielt i mange maskiner. På DIKU findes Mokka-kalfe (En Sun Enterprise 6500 med 24 processorer og 24GB RAM) som benytter en variant af det fælles lager og Dancehall. Desuden har DIKU Kvaser (16 Pentium maskiner sat sammen i et hurtigt SCI netværk, en såkaldt klynge), som benytter en variant af det distribuerede lager.

4.2 Uniprocessor

Som den mest gennemprøvede og velkendte teknologi beskrives en række muligheder for at understøtte løsning af lineære ligningssystemer i et uniprocessor system.



Figur 4.2 – Distribueret lager. Processorerne har separat adresserum og er forbundet i et netværk. \$ er cache.



Figur 4.3 – Dancehall lager. Lageret er adskilt i banke, men dog stadig med fælles adresserum og tilgangshastighed. \$ er cache.

Langt den mest fremherskende metode i dag er på den ene eller anden måde at udvikle en hurtig processor. Man er i øjeblikket ved at nå et stadie, hvor arkitekturene er blevet så store og komplicerede, at det er svært at udvide dem til at køre hurtigere. Et godt eksempel er Intels Pentium 4, som langt fra kan yde det, som dens skabere havde håbet på.

4.2.1 Én modificeret RISC-kerne

Pointen med denne arkitektur er, at det skal være muligt for processoren selv at finde nyttigt arbejde. På denne måde kan allerede eksisterende programimplementationer bruges.

En forudsætning for at denne type arkitektur opnår optimal udnyttelse, er naturligvis at programmerne udviser tilstrækkelig parallelisme til, at processoren kan starte så mange instruktioner, at alle eksekveringsenheder holdes i gang.

For at kunne finde afhængigheder mellem fortløbende instruktioner og tilgang til lageret kræves dog en hel del hardware. For instruktionsdelen alene kræver det $O(N^2)$ tidskritiske kredsløb, hvor N er antallet af instruktioner, som kan køres parallelt. Desværre vil det være et stort problem at få processoren til at køre med en høj klokfrekvens på grund af denne store kompleksitet.

En meget stor fordel for udviklingen af denne arkitektur er, at der ikke benyttes nogen nye og uafprøvede koncepter. Alle dele er kendt teknologi. Med

lidt held og en god implementation er software delen trivielt, da en almindelig uniprocessor implementation eller en lettere modificeret version kan benyttes med godt resultat.

4.2.2 VLIW-kerne

Til forskel fra ovenstående RISC processor bygger VLIW på at overlade det til oversætteren at skedulere instruktionerne. Det er derfor i lighed med den foregående arkitektur vigtigt, at software implementationen udviser en sådan parallelisme.

VLIW arkitekturen angriber RISC arkitekturens skeduleringsproblemer, da mange afhængigheder kan afgøres på oversættelsestidspunktet. Ved at overlade det til software at foretage skeduleringen af instruktionerne forsimples hardwaren væsentligt, og det bliver derfor langt nemmere at optimere hardwaren således, at den overordnede ydelse stiger.

Problemet, i forbindelse med løsning af lineære ligningssystemer, er at denne type program består af nogle meget små løkker, som det er simpelt at fjerne afhængighederne af. Derfor ville denne type processor ikke tilbyde noget i forhold til en RISC maskine.

En største ulempe ved denne arkitektur er, at den baserer sig på helt ny og uafprøvet teknologi. Den oversætterteknologi, som skal skedulere instruktionerne effektivt, er endnu ikke udviklet i en kvalitet, der er god nok. Samtidig har faktiske implementationer af processoren, endnu ikke set dagens lys og der er dermed ingen fortilfælde at læne sig op ad. Intel har i samarbejde med Hewlett Packard projektet Itanium [ITINET] undervejs, men dags dato er den ikke offentligt tilgængeligt på markedet før om ca. en måned.

4.2.3 En vektor processor

RISC arkitekturen og VLIW benytter sig som regel af SISD (Single Instruction Single Data), hvorimod vektor computeren arbejder med SIMD (Single Instruction Multiple Data). SISD betyder, at der for hver instruktion kun indgår et enkelt dataelement, mens SIMD betyder at en operation kan arbejde på lang række dataelementer.

Kernen i vektorprocessoren er vektorregistre. Disse registre fyldes med data fra lageret og en given operation kan udføres på samtlige elementer i registret samtidigt.

Et eksempel på hvordan SISD arbejder i forhold til SIMD kan ses i algoritme 4.1 og 4.2:

LØKKE I SISD (4.1)

- (1) label:
- (2) **load** R1,0[R4]
- (3) **mul** R1,R2,R1

- (4) **add** R1,R3,R1
- (5) **store** R1,0[R4]
- (6) **add** R4,R4,4
- (7) **add** R5,R5,-1
- (8) **jumpnotzero** R5,label

LØKKE I SIMD

(4.2)

- (1) **label**:
- (2) **loadvec** V1,R4
- (3) **mulvec** V1,R2,V1
- (4) **addvec** V1,R3,V1
- (5) **storevec** V1,R4
- (6) **add** R4,R4,VLR
- (7) **add** R5,R5,-VLR
- (8) **jumpnotzero** R5,label

Eksemplet udfører operationen $\bar{X} = a * \bar{X} + b$. a ligger i register 2, b ligger i register 3 og antallet af elementer, der skal regnes på ligger i register 5. Data hentes fra adressen gemt i register 4 og fremefter. VLR betegner vektorcomputerens størrelse af vektorregister.

Den store forskel bliver at vektorprocessoren udfører løkken R4/VLR gange, hvor VLR er antallet af indgange i vektorregistrene, typisk 64-256 elementer, og R4 er register 4 fra eksemplet. SISD processoren derimod udfører løkken R4 gange, og det er svært at forudsige for processoren hvad der sker i fremtiden.

Da vektorprocessoren arbejder på denne måde, er det muligt at optimere lagertilgangen, da denne bliver meget regulær. Den lagermodel, som oftest benyttes, er dancehall. Ved at lade hver indgang i et vektorregister allokere en lagerblok opnås at kunne fylde et vektorregister med den fulde båndbredde fra lageret.

En vektorcomputer er velegnet til løsning af lineære ligningssystemer, da det er muligt at lave en implementation, som udnytter vektor computerens fordele effektivt og opnår tæt på 100% af den teoretiske hastighed. Derfor har vektor computere ofte været anvendt til at løse denne type problemer, når det var hastighed der var brug for.

En af de store fordele ved denne løsning er, at der kun skal udvikles software. Endvidere eksisterer der allerede oversættere, som kan optimere et program til en vektorcomputer. Der findes adskillige kommercielle produkter på markedet, hvis pris afspejler de store udviklingsomkostninger og få anvendelser.

4.3 Multiprocessor

Når der er brug for betydelig regnekraft, kan det ofte betale sig at kaste sig over parallelismen i et givet problem. Derfor baserer alle store supercomputere sig i dag på på flere processorer, og multiprocessorparadigmet er langsomt ved

at finde anvendelse i almindelige arbejdsstationer.

Løsning af fulde lineære ligningssystemer er en opgave, som paralleliserer fint og med en god implementation - næsten ideelt. Derfor er denne type maskiner ofte set anvendt til dette problem.

4.3.1 En multiprocessormaskine med fælles lager

En multiprocessormaskine med fælles lager kaldes i daglig tale ofte for en supercomputer. Pointen med denne type maskine er, at det er muligt at kombinere flere hurtige processorer til at løse ligningssystemet. Hver af disse processorer kunne være en af de forgående, f.eks. mange vektorprocessorer.

En anden tiltalende egenskab er, at supercomputeren i mange tilfælde vil have adgang til store mængder lager, hvormed problemstørrelsen ikke er en hindring. Ligeledes vil der oftest være en program implementation til rådighed. Desværre er et fælles lager ikke hurtigt når antallet af processorer stiger, da lageret vil få problemer med konsistens, som omtalt i afsnit 4.1.

Udviklingsomkostninger

Da der kan benyttes et allerede tilgængeligt kommercielt produkt, er det ikke nødvendigt at udvikle hardware. Ligeledes er programmet også kommercielt tilgængeligt. Derfor er dette en stærk, gennemafprøvet og hurtig løsning. Prisen på disse maskine er dog meget stor og derfor, vil denne løsning kun være tilgængeligt for en lille gruppe mennesker.

4.3.2 COTS maskiner

I stedet for at investere i store specialiserede supercomputere kan det ofte betale sig at gå efter en COTS maskine (Commercial Of-The-Shelf). En sådan maskine består, som navnet antyder, af komponenter der er tilgængelige i computerforretninger.

De seneste årtiers udvikling indenfor almindelige arbejdsstationer har i dag gjort COTS maskiner utrolig hurtige til prisen. Ved køb af disse maskiner får man, som situationen er i dag, langt mere processorkraft for pengene end ved mere specialiserede arkitekturer.

Et antal kraftige arbejdsstationer kan sættes sammen i et LAN, en såkaldt klynge, som efterligner arkitekturen af en supercomputer med distribueret lager. Lageret i systemet er distribueret ved, at hver maskine er født med egen lagerenhed. Kommunikationsomkostningerne er ekstreme i forhold til de tidligere arkitekturer, da man her kommunikerer gennem et LAN. På grund af det distribuerede lager vil det være nødvendigt at bruge processortid på eksplicit

at kommunikere elementer som skal deles. Det er derfor ikke utænkeligt, at en eller flere maskiner vil løbe tør for arbejde, mens de venter på kommunikation.

Software udvikling på disse maskiner adskiller sig ikke væsentligt fra udviklingen til andre parallelmaskiner. Derfor kan den forskning, som er lavet til supercomputere overføres næsten direkte til implementation på et sådan system. Specielt hvis implementationen ikke er meget kommunikationskrævende.

4.4 Massiv parallelisering

Processor designs er i dag ved at nå en sådan kompleksitet, at det begynder at være svært at finde på nye idéer, som forøger ydelsen væsentligt i forhold til antal forbrugte transistorer. Dette er ikke noget nyt problem - RISC og VLIW teknologierne er begge opstået, da det har været nødt til at "starte forfra" og finde på noget nyt. Inden for en overskuelig fremtid vil det være muligt at klemme flere hundrede millioner transistorer ned på en enkelt chip. Med så mange transistorer er det usandsynligt, at RISC og VLIW skalerer godt. Igen er der brug for nytænkning. Den idé, som nogle forskere arbejder med, er massiv on-die parallelisering: Ved at koble beregningsenheder sammen i et tæt koblet netværk på en chip, håbes det fortsat at kunne øge ydelsen af mikroprocessorer.

Det mest kendte projekt inden for denne tankegang er MITs RAW arkitektur. RAW[MITINET] arbejder med en idé, som de kalder "reconfigurable computing" [EW97], som går ud på at placere mange processorer i et on-the-fly rekonfigurerbart netværk på én chip.

Firmaet Star Bridge Systems[SBSINET] udvikler maskiner ud fra FPGA teknologi. En FPGA er bygget op af en lang række makroceller i et kommunikationsnetværk, som kan omprogrammeres on-the-fly. Hver makrocelle kan programmeres som en gate-funktion, således at FPGA chippen kan simulere et kredsløb. Idéen i Star Bridge Systems PensaTM-maskine er at udvide makroceller til at indeholde mere intelligens f.eks. en kombineret multiplikations- og additionsenhed eller en hel processor.

4.4.1 Flere RISC-kerner på én chip

Ved at placere flere RISC processorer på samme chip reduceres kommunikationsomkostningerne til et minimum. Herudover er princippet i denne arkitektur det samme som for andre multiprocessorer maskiner.

Arkitekturen kunne implementeres med alle typer lagermodeller, med hver deres fordele og ulemper. Forestiller man sig en række processorer med hver sit lager i et tætkoblet netværk, vil tilgangen til egen og naboers lager være hurtig. Et tætkoblet netværk med stor båndbredde kan potentielt optage meget areal på chippen, og hvis kommunikationsvejene bliver for lange, kan det gå ud over

klokkfrekvensen. Hvis der i stedet benyttes fælleslager er det åbenlyst, at båndbredden kan blive en begrænsning. Tilgås processorerne lageret gennem en fælles bus, kunne denne bus ligeledes blive en begrænsning. Båndbreddeproblemet kunne afhjælpes ved at vælge en lagerarkitektur med tilstrækkelig båndbredde såsom dancehall.

Denne arkitektur baserer sig på RISC kerner sat sammen med kendt netværksteknologi og udgør tilsammen alligevel en ny teknologi, som det ikke er usandsynligt, vi kommer til at se mere til fremover.

4.4.2 Et systolisk array

Et systolisk array er en samling af ens elementer, der er sat sammen således, at de er i stand til at løse et matematisk problem utrolig effektivt. Arrayet består af mange gentagne elementer, som gør det let at bygge. Ideen er, i stil med hjertets systole, at pumpe data igennem arrayet i veldefinerede trin. Det var et af de store hit, da VLSI designs kom frem. Det viste sig dog, at der var begrænsninger: Antallet af transistorer ville være for stort til at datidens teknologi kunne bruges, og en udvidelse til at lægge hvert element på hver sin chip ville gøre det for dyrt og for langsomt. Derfor kunne det bedre betale sig at bruge pengene på "general purpose" maskiner. Da der, som tidligere omtalt, i dag kan være adskillige millioner transistorer på en chip er idéen igen meget interessant, da det nu er muligt at trykke hele arrayet på én chip.

Årsagen til at løsningen af et lineært ligningssystem, ved hjælp af et systolisk array, kan lade sig gøre er, at det følgende gælder (en omskrivning af algoritme 3.2 på side 11):

$$a_{ij}^{(1)} = a_{ij} \quad (4.1)$$

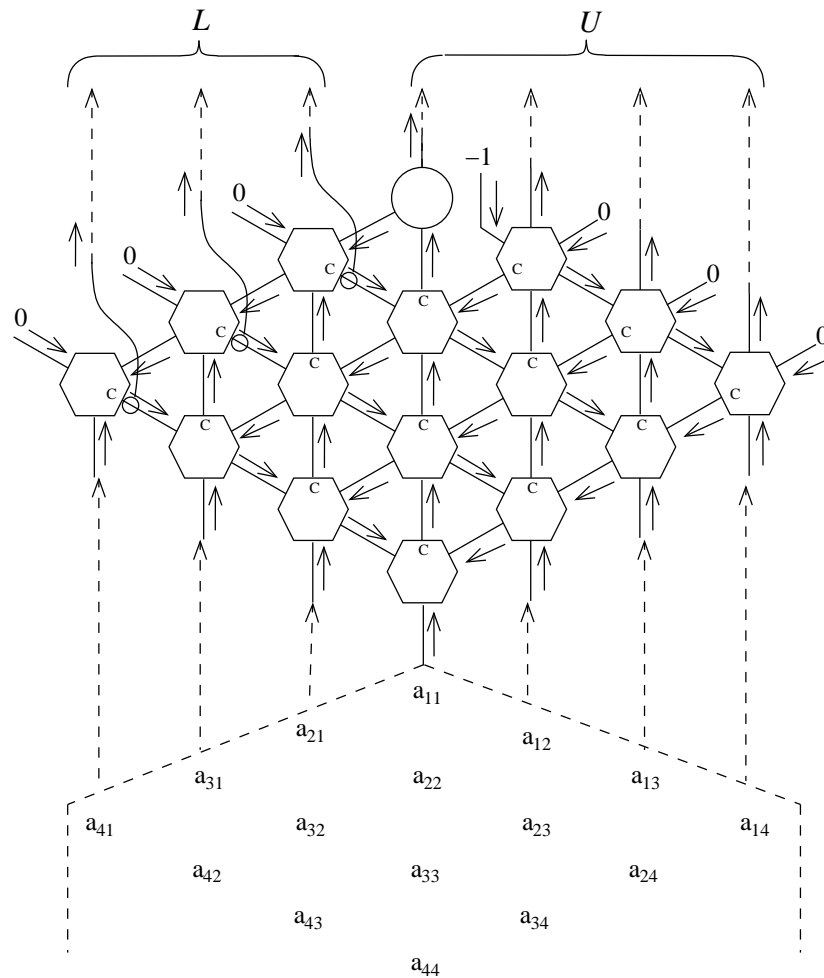
$$a_{ij}^{(n+1)} = a_{ij}^{(n)} + l_{in}(-u_{nj}) \quad (4.2)$$

$$l_{ik} = \begin{cases} 0 & : i < k \\ 1 & : i = k \\ a_{ik}^{(k)} u_{kk}^{-1} & : i > k \end{cases} \quad (4.3)$$

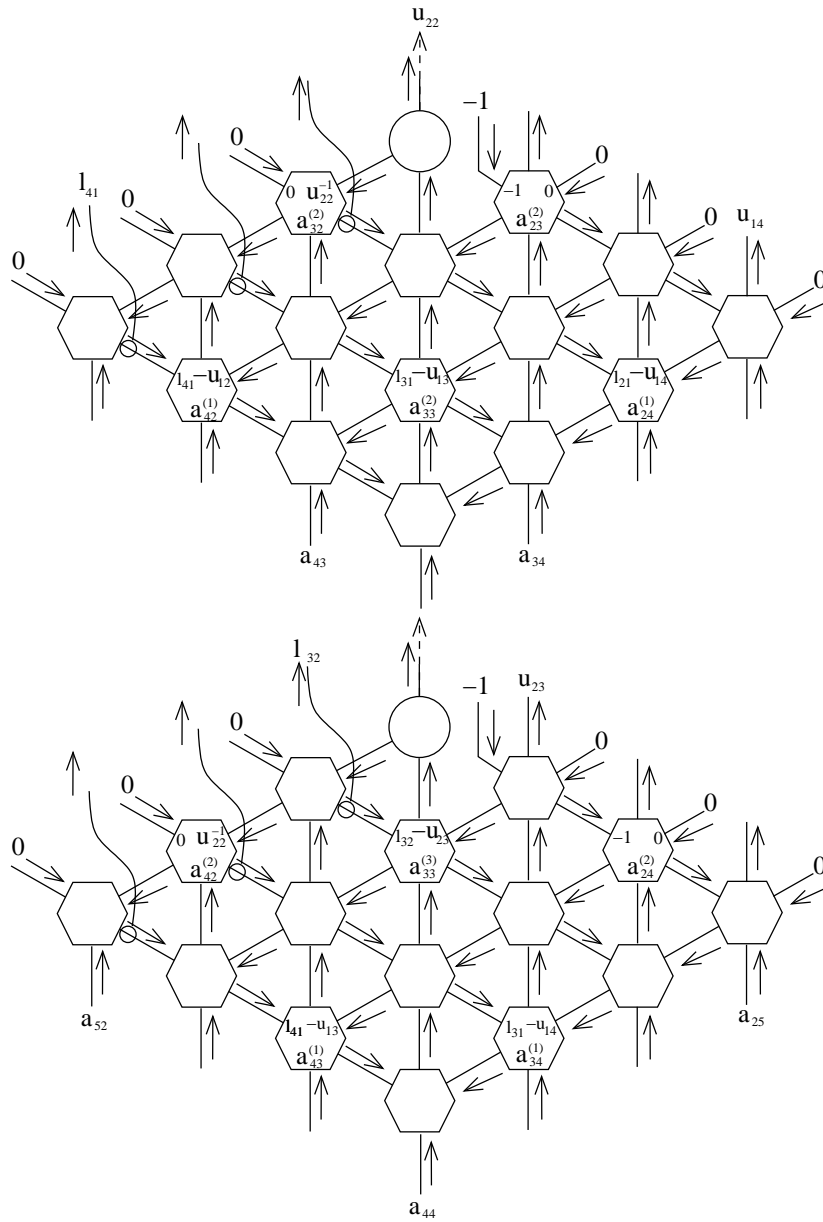
$$u_{kj} = \begin{cases} 0 & : k > j \\ a_{kj}^{(k)} & : k \leq j \end{cases} \quad (4.4)$$

Hvor l_{ij} og u_{ij} er indgangen i L hhv. U matricen.

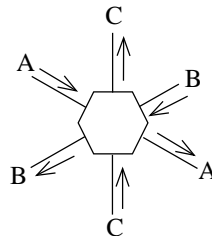
På figur 4.4 ses et systolisk array, der kan LU-transformere en matrix med kardinalitet 4. De sekskantede elementer er celler, der er i stand til at beregne det indre produkt: $C = C + A * B$ (se figur 4.6). Den øverste venstre række i arrayet er drejet 120 grader med uret således, at C vender ind mod arrayet, ligeledes er øverste højre kant drejet 120 grader mod uret. Den øverste runde celle er en såkaldt reciprokcelle, idet den beregner reciprokken af inddata. På figur 4.5 illustreres 2 trin af processen.



Figur 4.4 – Et systolisk array [CM80, s.282]. c fra figur 4.6 er markeret så orienteringen af elementerne står frem.



Figur 4.5 – Et systolisk array i aktion [CM80, s.283].



Figur 4.6 – Celle til at beregne indre produkt [CM80, s.273].

Når reciprokcellen modtager et element u_{kk} gør den 2 ting: Sender u_{kk} ud af arrayet som resultat samt beregner og sender u_{kk}^{-1} til sin venstre nabo.

Øverste venstre kant beregner ligning 4.3 med u_{kk}^{-1} fra reciprokcellen og resultaterne $a_{ik}^{(k)}$ fra de indre celler. Resultatet l_{ik} sendes ud af arrayet.

Når cellerne i øverste højre kant modtager et element beregnes ligning 4.4. Ydermere beregnes $-u_{jk}$, som sendes ind i arrayet.

De indre celler producerer elementerne, der fremgår af ligning 4.2.

Da ligning 4.1-4.4 er det samme som en LU transformation (se algoritme 3.2 på side 11) gælder, at et systolisk array er i stand til at LU transformere A ifølge argumentationen oven for. På grund af det systoliske arrays opbygning følger det, at LU transformationen forløber i lineær tid og ikke forbruger mere plads end resultatet [CM80].

Arrayet er opbygget af en række simple celler, som kopieres i stort antal og sat sammen i et meget regulært netværk. Således vil det være nemt at automatgenere et systolisk array af en given størrelse. Samtidig findes automatgeneratoren til multiplikationsenheder, der indgår i arrayets celler. Altså vil det være meget nemt at konstruere elektronikken til det systoliske array, med undtagelse af grænseflade til lageret eller omkringliggende maskineri.

Kapitel 5

Opbygning af systolisk array

Flere af de beskrevne arkitekturer kunne være interessante at undersøge i forbindelse med løsning af lineære ligningssystemer. Flere processorer på én chip er en spændende idé, som kan vise at være meget hurtig. Den arkitektur som ser mest lovende ud med hensyn til ydelse, er dog det systoliske array - da det virker besnærende at et system bygget udelukkende med formålet at løse lineære ligningssystemer, må være uhyre effektivt.

Dette kapitel vil derfor koncentrere sig om nogle af de problemstillinger, der er med opbygning af et systolisk array.

Der omtales i kapitlet en fiktiv klokperiode, denne klokperiode er ikke defineret. En given implementation vil fastlægge klokperioden for det systoliske array.

5.1 Generelle problemer

Det systoliske array, som beskrevet i sidste kapitel, har nogle problemer:

Arrayet bruger kun 1/3 af sine elementer ad gangen. Ved at lave hver celle som en lille tilstandsmaskine, kan dette udnyttes til at spare en trediedel af cellerne.

Arrayet kan ikke håndtere tyndtbefolkede matricer på en smart måde. Der er flere løsninger til dette problem. Der kan vælges simpelthen ikke at tage højde for, tyndtbefolkede matricer og dermed løse dem som almindelig matricer. Denne løsning er ikke optimal, hvis matricen har mange 0-elementer, da der bruges tid på unødige udregninger.

Alternativt kan en anden matematisk metode vælges således, at det er muligt at dele løsningen af matricen op i flere underproblemer, der hverisær består af

mindre lineære ligningssystemer.

Arrayet håndterer kun LU-transformation. Det er muligt at bygge systoliske array, der laver matrix multiplikation, matrix addition, tilbagesubstitution mm. svarende til LU faktoriseringen (se [CM80]). Dermed er det muligt, at benytte en af de tidligere omtalte arkitekturer, som RAW, til dynamisk at omkonfigurere det systoliske array til at klare alle opgaver.

Arrayet skalerer slet ikke. Arrayet er ikke i stand til at løse LU faktoriseringen af et problem, der er større, end det det er bygget til. Derfor kunne hvert af de systoliske celler udbygges til at håndtere M tal hver, hvilket ville presse køretiden op på $O(N^3/K^2)$ [CM80], hvor $K = N/M$ og K angiver antallet af systoliske celler. Det ville dog ikke lave om på det faktum, at arrayet stadig kun er i stand til at løse en matrix, der har en maksimal størrelse.

Hvis et ligningssystem derimod er mindre end størrelsen af det systoliske array, er det trivielt at udvide det til et større ved at tilføje ligegyldige rækker og søjler. Det systoliske array vil dog arbejde med ligegyldige data, hvilket nedsætter effektiviteten.

Arrayet tager ikke højde for numerisk stabilitet. Da det ikke er muligt for det systoliske array, at tage højde for pivotering af elementer, kan der opstå problemer med den numeriske stabilitet. En måde at omgå dette på ville være at skifte nøjagtighed af elementerne. Hvis de elementer, der skal regnes på, er f.eks. 64 bit, kunne nøjagtigheden internt i det systoliske array forøges til 128 bit og konvertere tilbage, når beregningen er færdig. Der kan dog selv med større præcision risikeres, at løsningen divergerer. Endvidere ville det ikke være muligt at håndtere diagonalelementer som er nul.

5.2 Lagertilgang

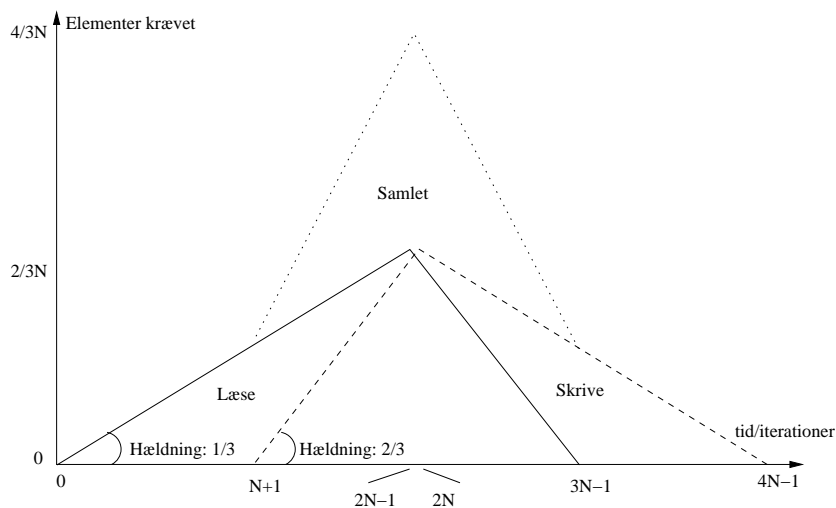
For at udlede køretiden for et systolisk array, er det nødvendigt at udlede hvordan det systoliske array tilgår elementerne. Figur 5.1 viser dette både for læsning og skrivning.

Ud fra figuren udledes læsebåndbreddeforbruget for hver iteration. Det viser sig, at tilgangen er meget regulær når man ser på hver tredje iteration. Følgende tabel viser, hvordan båndbreddeforbruget ændrer sig i løbet af beregningen. Tabellen beskriver i kompakt form hvordan båndbreddeforbruget ændrer sig fra begyndelsen over maksimumforbrug til hele ligningssystemet er indlæst i det systoliske array.

1	3	5	7	9	11	13	15	17	19
3	4	6	8	10	12	14	16	18	20
5	6	7	9	11	13	15	17	19	21
7	8	9	10	12	14	16	18	20	22
9	10	11	12	13	15	17	19	21	23
11	12	13	14	15	16	18	20	22	24
13	14	15	16	17	18	19	21	23	25
15	16	17	18	19	20	21	22	24	26
17	18	19	20	21	22	23	24	25	27
19	20	21	22	23	24	25	26	27	28

11	12	13	14	15	16	17	18	19	20
12	14	15	16	17	18	19	20	21	22
13	15	17	18	19	20	21	22	23	24
14	16	18	20	21	22	23	24	25	26
15	17	19	21	23	24	25	26	27	28
16	18	20	22	24	26	27	28	29	30
17	19	21	23	25	27	29	30	31	32
18	20	22	24	26	28	30	32	33	34
19	21	23	25	27	29	31	33	35	36
20	22	24	26	28	30	32	34	36	38

Figur 5.1 – Tallene angiver i hvilken iteration, elementerne hentes ind i det systoliske array. Den venstre angiver indlæsning og den højre skrivning af resultat. Den 19. iteration er markeret som den, der forbruger mest båndbredde.



Figur 5.2 – Lagerbåndbreddeforbrug af det systoliske array under hele processen.

Iteration	%3=0	%3=1	%3=2
1	-	1	-
2	-	-	0
3	2	-	-
4	-	1	-
5	-	-	2
6	2	-	-
7	-	3	-
8	-	-	2
9	4	-	-
10	-	3	-
11	-	-	4
12	4	-	-
13	-	5	-
14	-	-	4
15	6	-	-
...
$r\%3=0$	$2 * \lceil \frac{r}{6} \rceil$	-	-
$r\%3=1$	-	$2 * \lceil \frac{r+2}{6} \rceil - 1$	-
$r\%3=2$	-	-	$2 * \lfloor \frac{r+1}{6} \rfloor$
...
$\sim 2N$	Maks		
$\sim 2N+3$	Maks-2		
$\sim 2N+6$	Maks-4		
...
$3N$	0	0	0

Det maksimale båndbreddeforbrug afhænger af, hvordan 3 går op i N:

$$\text{Maks} = \left\{ \begin{array}{ll} 2 * \lceil \frac{2N}{6} \rceil & N\%3 = 0 \\ 2 * \lceil \frac{2N+2}{6} \rceil - 1 & N\%3 = 1 \\ 2 * \lfloor \frac{2N+2}{6} \rfloor & N\%3 = 2 \end{array} \right\} \approx \frac{2N}{3}$$

For skrivning gælder det nøjagtig spejlvendt båndbreddeforbrug, toppunktet ligger dog 1 iteration længere fremme end læsningens toppunkt (se figur 5.2). Som det ses af tabellen, kan det nemt forudsiges hvilke elementer, det er nødvendigt at hente, hvilket gør arbejdet meget nemmere for lageret. Samlet leder det frem til, at det systoliske arrays køretid er 4N.

5.2.1 Buffer

Som det tydeligt kan ses af figur 5.2, ændrer båndbreddeforbruget til lageret sig over tid med en stor maksimumværdi. Det kan være et problem for lager-

Arealet er da:

$$A = \frac{1}{2}hg = \frac{1}{2} * \frac{5N}{6} * \frac{5N-6}{3} = \frac{25N^2 - 30N}{36} \approx \frac{7}{10}N^2 \quad (5.7)$$

For et systolisk array af størrelsens 256×256 skal der bruges en buffer på ca. 45.000 elementer. Med elementer på 64 bit er det ca. 350kB buffer. Hvis en statisk ram bit i gennemsnit fylder ca. 6 transistor/bit, fylder bufferen ca. $6 * 64 * 45.000 = 17,4$ mio. transistorer.

5.3 Størrelse

For at sandsynliggøre konstruktionen af et systolisk array indenfor overskuelig fremtid, er det interessant at anslå størrelsen i transistorforbrug. I det følgende antages størrelsen af elementerne til 64 bit.

Hver af de systoliske celler i et array skal kunne udføre en samlet multiplikation og addition $C = C + A * B$. Overordnet kommer hver celle til at bestå af en multiplikations- og additionsenhed og en kontrolenhed. Hver celle i det systoliske array er koblet sammen i et netværk, men forbindelserne er statiske og forbinder kun naboceller. Derfor vil netværket optage et minimalt areal på chippen.

5.3.1 Systolisk celle

Kontrolenheden i hver celle skal være meget simpel og forbruger ganske få transistorer i det samlede billede; det antages, at den kan implementeres på 500 transistorer.

Den afgørende faktor for transistorforbruget vil være multiplikations- og additionsenheden. For at opnå maksimal ydelse er det nødvendigt, at det systoliske array kan behandle elementer i samme hastighed, som lageret kan levere dem. Derfor er der brug for en multiplikations- og additionsenhed, der netop er hurtig nok. Den simpleste enhed består af et shift register og en 64 bit adder, som i løbet af 64 klokperioder kan multiplicere 2 tal. I den anden ende af skalaen ligger en enhed, der er i stand til at beregne det samme på én klokperiode, transistorforbruget stiger dog drastisk.

For at få et overslag på transistorforbruget af en 64 bit enkelt-klokperiode multiplikationsenhed, benyttes en automatgenerator. VLSI værktøjet "Alliance" indeholder programmet "AMG - Array Multiplier Generator", som generer multiplikatorer til heltal af den simple arraytype. En multiplikator til flydendekommatil adskiller sig ikke væsentligt i størrelse fra en heltalsmultiplikator og derfor ses bort fra det yderligere transistorforbrug. Parametre til AMG er vedlagt som bilag; af pladshensyn er datafiler ikke vedlagt.

Klokperioder	Transistorer element	Samlet u. buffer	Max (2N/3t) båndb./klok	Samlet m. buffer	Effektiv (N/2t) båndb./klok
1	91 k	2,00 mia.	171~1,4 kB	2,02 mia.	128~1,0 kB
2	46 k	1,01 mia.	85~0,7 kB	1,02 mia.	64~0,5kB
4	23 k	509 mio.	43~0,3 kB	526 mio.	32~0,3kB
8	12 k	260 mio.	21~0,2 kB	277 mio.	16~0,1 kB
16	6,2 k	135 mio.	11~85 B	153 mio.	8~64 B
32	3,4 k	73,1 mio.	5~43 B	90,5 mio.	4~32 B
64	1,9 k	42,0 mio.	3~21B	59,4 mio.	2~16 B

Tabel 5.2 – Ekstrapoleret transistorforbrug af systolisk array som funktion af køretid af multiplikationsenhed. Antallet af transistorer er angivet for hver celle samt for hele arrayet med og uden buffer. Desuden den nødvendige båndbredde angivet i antal ord/klokperiode og byte/klokperiode.

Den resulterende multiplikator fylder 91144 transistorer. Den sidste additionsenhed, som er nødvendig til multiplikations- og additionsenheden i en systolisk celle, fylder ikke meget i forhold til dette, derfor ses bort fra dette transistorforbrug. Forbruget af transistorer i en multiplikationsenhed aftager tilnærmelsesvis omvendt proportional med antallet af klokperioder. Ud fra arraymultiplikatoren ekstrapoleres transistorforbruget af langsommere multiplikatorer se bilag D.

Hvert systolisk array indeholder én reciprokenhed. Denne enhed vil bruge flere transistorer end multiplikationsenheden, men sammenlignet med det meget store antal multiplikationsenheder, vil det ikke være afgørende. Derimod er det afgørende, at reciprokenheden skal køre med samme frekvens som multiplikationsenhederne. Det er tvivlsomt, om det kan lade sig gøre at bygge en sådan en enhed, som kan køre i én klokperiode, idet det er nytteløst at pipeline enheden. Det er dog sandsynligt, at reciprokenheden kan laves hurtig, for det første fordi den er simplere end en sædvanlig divisionsenhed, for det andet fordi den kan bygges meget stor, idet selv en stor reciprokenhed ikke er afgørende i det samlede regnskab. Det ligger uden for denne rapport, om og så fald hvordan en sådan enhed kan laves.

5.3.2 Samlet transistorforbrug

Antallet af transistorer for en systolisk celle kan ses som en funktion af multiplikationsenhedens hastighed målt i klokperioder se tabel 5.2. Der er i tabellen regnet med et systolisk array, der kan klare et ligningssystem med kardinalitet 256 hvilket vil kræve $256^2/3 = 21845$ celler.

I figuren er en række størrelser er anslået. Moores lov fremskriver, at inden for få år vil det være muligt at presse 1 mia. transistorer ned på en chip. Holder

denne forudsigelse, vil det inden længe være realistisk at implementere et af de givne forslag.

Det systoliske array skal netop være hurtigt nok, til at båndbredden til hukommelsen kan følge med, da der ellers ville bruges tid på at vente på hukommelsen eller på det systoliske array. Derfor kan det nu for et givet hukommelsessystem lade sig gøre at skyde sig ind på hvilken størrelse chip, som skal til for at kunne løse ligningssystemerne optimalt.

Kapitel 6

Ydeevne analyse

I sidste kapitel blev vigtige aspekter af det systoliske arrays opbygning belyst, i dette kapitel vil ydeevnen blive studeret.

For at finde et systoliske arrays hastighedsbegrænsning kunne der tages udgangspunkt i en prototype. Denne kunne forestilles implementeret i et lavniveau kredsløbsbeskrivelsesprog som VHDL eller Verilog. En sådan implementation ville nøjagtigt kunne simulere de enkelte celler eller hele arrayet. Hardware begrænsningerne er dog så forudsigelige, elementerne så simple og tilgangen til lageret så regelmæssig, at det med stor nøjagtighed kan lade sig gøre at forudsige ydeevnen på anden vis. Ydeevne analysen af det systoliske array vil derfor primært være tænkt og ikke basere sig på egentlige målinger.

Ydelsen af det systoliske array afhænger i høj grad af hvilken type ligningssystem som løses, da det systoliske array, som præsenteret i sidste kapitel, ikke er i stand til at håndtere store ligningssystemer uden et utopisk antal transistorer. For små ligningssystemer kan det systoliske array umiddelbart anvendes, mens det ikke er indlysende for store ligningssystemer hvordan det systoliske array kan anvendes.

Som omtalt i kapitlet om kredsløbsanalyse involverer kredsløbssimulering ofte løsning af en mængde lineære ligningssystemer, der er BBD (Block Bordered Diagonal, se afsnit 2.2 på side 4). I andre typer problemer opstår der ligningssystemer, som minder om BBD, men uden rand. Denne type ligningssystemer ligner båndmatricer og kan tolkes som blok tridiagonale. Disse ligningssystemer er dog ofte meget store og derfor introduceres to matematiske metoder til at opdele ligningssystemet, så det systoliske array kan bruges. Ydelsen af det systoliske array analyseres i forbindelse med disse to forskellige typer ligningssystemer og med små ligningssystemer, som det systoliske array kan håndtere uden problemer. For at forenkle analysen ses kun på selve LU faktoriseringen, da langt den tungeste del af løsningsprocessen ligger i LU faktoriseringen.

Da det systoliske array ofte ikke er i stand til at klare hele opgaven alene

antages det, at det sidder i en almindelig arbejdsstation som slaveprocessor. Hovedprocessoren styrer den overordnede fremgangsmåde, mens det systoliske array LU faktorerer efter behov. Derfor antages i det følgende, at størrelsen af det systoliske array er lavet til at håndtere matricer af 256×256 og at præcisionen er 64 bit. Denne størrelse giver ligningssystemer, som ikke er for små og som kan implementeres inden for et fornuftigt antal af transistorer, som vist i sidste kapitel.

Pivotering er udeladt, da det systoliske array omfatter ikke pivotering, og en realistisk sammenligning vil give det systoliske array en fordel i forhold til de metoder, der benytter sig af pivotering.

6.1 Små tætbefolkede ligningssystemer

For disse små ligningssystemer sammenlignes ydelsen af et systoliske array med en traditionel uniprocessor implementation, og der indføres derfor en ny processor - UtopiskHurtig™. Denne processor er i stand til at udføre 16 multiplikationer, additioner eller kombinationen per klokperiode. Alle andre operationer er den i stand til at udføre i 0 tid. Lagersystemet er ideelt, da det er i stand til at vide, hvilke elementer der vil være brug for i de næste beregninger, og båndbredden er ikke et problem. En implementation af LU faktorisering uden pivotering kan realiseres med køretiden [WJM88]:

$$\frac{N^3}{3} + N^2 - \frac{N}{3}$$

Denne processor vil kunne LU faktorisere et ligningssystem på:

$$\text{tid}_{\text{uni}} = \frac{1}{16} * \left(\frac{N^3}{3} + N^2 - \frac{N}{3} \right)$$

Som omtalt LU faktorerer det systoliske array i $4N$ tid, altså:

$$\text{tid}_{\text{sys}} = 4N$$

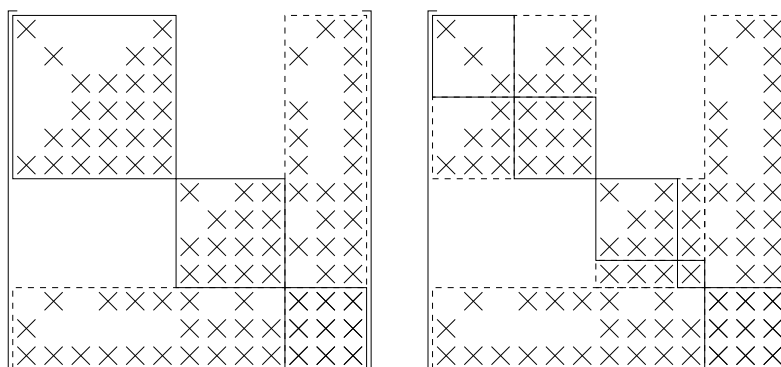
For en matrix af kardinalitet 256 giver dette:

Arkitektur	tid
UtopiskHurtig™	353616
Systolisk array	1024

Hvilket betyder at et systoliske array bygget til kardinalitet 256 vinder for matricer af kardinalitet over

$$\frac{1}{16} * \left(\frac{N^3}{3} + N^2 - \frac{N}{3} \right) = 4 * 256 \Rightarrow N \approx 36$$

Hvis kardinaliteten derimod vokser til over 256, vil det systoliske array straks få problemer.



Figur 6.1 – BBD ligningssystem fra figur 2.1 inddelt i variabel og fast blokstørrelse

Det er ikke nødvendigt at undersøge en mere realistisk uniprocessor, idet denne kun yderligere vil understrege det systoliske arrays styrke, for matricer som er velegnede til det systoliske array.

6.2 BBD ligningssystemer

Det systoliske array kan bruges til at øge hastigheden af løsning af lineære ligningssystemer af typen BBD væsentligt. Til denne type ligningssystem præsenteres en optimeret blok LU faktorisering, der udnytter at matricen er tyndt-befolket.

Som udgangspunkt benyttes blokmetoden fra afsnit 3.3.2 på side 10. Matricen opdeles i blokke efter sin struktur, som LU transformeres en ad gangen. Et BBD ligningssystem har strukturen:

$$A = \begin{bmatrix} A_1 & & & B_1 \\ & A_2 & & B_2 \\ & & \dots & \dots \\ & & & A_m & B_m \\ C_1 & C_2 & \dots & C_m & E \end{bmatrix} \quad (6.1)$$

Problemet med BBD ligningssystemerne er randen. Det er ikke muligt at LU faktorisere A_n blokkene uden også at opdatere randen. Derfor vil LU faktoriseringen bestå af en række LU faktoriseringer og en række opdateringer. Som omtalt i kapitlet om kredsløbsanalyse, er systemerne hierarkisk opbygget, derfor kan algoritmen anvendes rekursivt på hver blok indtil en blokstørrelse, som er passende for det systoliske array, opnås.

BBD ligningssystemet transformeres til:

$$A = \begin{bmatrix} L_1 U_1 & & & & \bar{B}_1 \\ & L_2 U_2 & & & \bar{B}_3 \\ & & \dots & & \dots \\ & & & L_m U_m & \bar{B}_m \\ \bar{C}_1 & \bar{C}_2 & \dots & \bar{C}_m & \bar{E} \end{bmatrix} \quad (6.2)$$

Af afsnit 3.3.2 på side 10 følger:

$$\begin{aligned} A_n &= L_n U_n \quad \text{for } n = 1, 2, \dots, m \\ \bar{C}_n &= C_n U_n^{-1} \quad \text{for } n = 1, 2, \dots, m \\ \bar{B}_n &= L_n^{-1} B_n \quad \text{for } n = 1, 2, \dots, m \\ \bar{E} &= E - \bar{C}_1 \bar{B}_1 - \bar{C}_2 \bar{B}_2 - \dots - \bar{C}_m \bar{B}_m \end{aligned} \quad (6.3)$$

Algoritmen opnår en hastighedsforøgelse frem for den tidligere bloktransformation af to årsager: Der opnås en stor besparelse ved kun at LU faktorisere ikke-0 blokke. Desuden opnås en væsentlig hastighedsforøgelse ved at benytte det systoliske array til LU faktoriseringen.

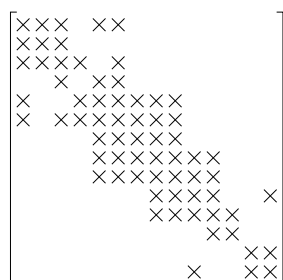
6.2.1 Løsning med et systolisk array

Det systoliske array, som beskrevet i sidste kapitel, kan løse LU faktoreringsdelen af 6.3, mens de øvrige beregninger må klares med hovedprocessoren eller en specielprocessor. I det følgende antages det, at det er hovedprocessorens opgave, der vendes tilbage til en specielprocessorløsningen i afsnit 6.3.3 i forbindelse med løsning af blok tridiagonale ligningssystemer.

Køretiden for LU faktorisering af et ligningssystem, der kan opdeles i blokke af samme størrelse som det systoliske array, kan estimeres meget simpelt. Hvis blokkene ikke passer i det systoliske array, vil køretiden blive en funktion af bredden af blokkene, men et tilsvarende resultat vil være gældende. Strassens algoritme til matrix matrix multiplikation kan klare denne opgave i $O(k^{\log(7)})$. Af ligning 6.3 ses at løsning med denne algoritme kræver m LU faktoriseringer, m matrix subtraktioner ($O(k^2)$), $3m$ matrix matrix multiplikationer.

Uden det systoliske array, betyder dette en kompleksitet på $mk^3 + 3mk^{\log(7)} + mk^2$, idet LU faktorisering er en $O(k^3)$ opgave [WJM88].

Det systoliske kan udføre LU faktoriseringen i $4k$ tid hvilket bringer køretiden ned på $3mk^{\log(7)} + mk^2 + 4mk$ - en ganske betydelig besparelse. I afsnittet om tridiagonale ligningssystemer, vises det hvordan de resterende matrixoperationer, som her udføres af hovedprocessoren, kan udføres ved brug af flere systoliske array, således at besparelsen øges yderligere.



Figur 6.2 – Båndmatrixsystem, kan tolkes som blok tridiagonalt ligningssystem se fig 6.3.

6.3 Blok tridiagonale ligningssystemer

I andre typer problemer, f.eks. styrke simulering af en bådmast, opstår der ligningssystemer, som minder om BBD, men uden rand. Denne type kaldes for blok tridiagonale og et eksempel på en båndmatrix, som kan tolkes som blok tridiagonal, kan ses i figur 6.2. Algoritmen er meget lig den foregående, men har specielle egenskaber der kan bruges til at vise hvordan et systoliske array kan udvides til at løse denne type ligningssystem meget effektivt. I denne forbindelse vises det, hvordan algoritmen kan udnytte flere systoliske array parallelt.

Den beskrevne algoritme er en multiniveau løsningsmetode til et blok tridiagonalt lineært ligningssystem [INH87].

Den matematiske definition af et blok tridiagonalt system ser ud som:

$$A = \begin{bmatrix} A_1 & B_1 & & & \\ C_2 & A_2 & B_2 & & \\ & C_3 & A_3 & B_3 & \\ & \dots & \dots & \dots & \dots \\ & & & C_m & A_m \end{bmatrix} \quad (6.4)$$

hvor

$$\begin{aligned} A_r &\in \mathcal{R}^{n_r \times n_r} && \text{for } r = 1, 2, \dots, m \\ B_r &\in \mathcal{R}^{n_r \times n_{r+1}} && \text{for } r = 1, 2, \dots, m-1 \\ C_r &\in \mathcal{R}^{n_{r-1} \times n_r} && \text{for } r = 2, 3, \dots, m \end{aligned} \quad (6.5)$$

og m er et ulige tal for klarhed skyld.

Metoden blok cyklisk reduktion [DH76] er i stand til at omforme A fra ligning

6.4 til:

$$\tilde{A} = \left[\begin{array}{cccc|cccc} A_1 & & & & B_1 & & & \\ & A_3 & & & C_3 & B_3 & & \\ & & A_5 & & & C_5 & B_5 & \\ & & & \dots & & \dots & \dots & \\ & & & & & & & C_m \\ \hline C_2 & B_2 & & & A_2 & D_2 & & \\ & C_4 & B_4 & & E_4 & A_4 & D_4 & \\ & & C_6 & \dots & & E_6 & A_6 & \dots \\ & & & \dots & & & \dots & \dots \\ & & & \dots & & & \dots & A_{m-1} \end{array} \right] \quad (6.6)$$

Således bliver A separeret i uafhængige blokke A_r for $r = 1, 3, \dots, m$. A_r for $r = 1, 3, \dots, m$ LU faktoriseres og \tilde{A} bliver til:

$$\bar{A} = \left[\begin{array}{cccc|cccc} L_1 U_1 & & & & \bar{B}_1 & & & \\ & L_2 U_3 & & & \bar{C}_3 & \bar{B}_3 & & \\ & & L_5 U_5 & & & \bar{C}_5 & \bar{B}_5 & \\ & & & \dots & & \dots & \dots & \dots \\ & & & & & & & \bar{C}_m \\ \hline \bar{C}_2 & \bar{B}_2 & & & \bar{A}_2 & \bar{D}_2 & & \\ & \bar{C}_4 & \bar{B}_4 & & \bar{E}_4 & \bar{A}_4 & \bar{D}_4 & \\ & & \bar{C}_6 & \dots & & \bar{E}_6 & \bar{A}_6 & \dots \\ & & & \dots & & & \dots & \dots \\ & & & \dots & & & \dots & \bar{A}_{m-1} \end{array} \right] \quad (6.7)$$

hvor

$$\begin{aligned} L_r U_r &= A_r && \text{for } r = 1, 3, \dots, m \\ \bar{B}_r &= L_r^{-1} B_r && \text{for } r = 1, 3, \dots, m-2 \\ \bar{C}_r &= L_r^{-1} C_r && \text{for } r = 3, 5, \dots, m \\ \bar{B}_{r-1} &= B_{r-1} U_r^{-1} && \text{for } r = 3, 5, \dots, m \\ \bar{C}_{r+1} &= C_{r+1} U_r^{-1} && \text{for } r = 1, 3, \dots, m-2 \\ \bar{D}_s &= D_s - \bar{B}_s \bar{B}_{s+1} && \text{for } s = 2, 4, \dots, m-3 \\ \bar{E}_s &= E_s - \bar{C}_s \bar{C}_{s-1} && \text{for } s = 2, 4, \dots, m-1 \\ \bar{A}_s &= A_s - \bar{C}_s \bar{B}_{s-1} - \bar{B}_s \bar{C}_{s+1} && \text{for } s = 2, 4, \dots, m-1 \end{aligned} \quad (6.8)$$

Det bemærkes, at den nederste højre fjerdedel af \hat{A} ligner A fra 6.4. Metoden har navnet multiniveau, da den rekursivt fortsætter med at løse matricerne \bar{A} på samme måde.

Dermed er der fundet en algoritme, der er i stand til at opdele ligningssystemet i mange uafhængige LU faktoriseringer.

Antagelsen m ulige (fra ligning 6.4) er en forenkling. For at kunne håndtere det lige tilfælde kan m gøres ulige ved at samle to blokke (A_r). Alternativt kan den blok cykliske reduktionsalgoritme rettes til at tage højde for det [INH87].

Den beskrevne algoritme tager ikke højde for, at A_r kunne være en singular matrix. En ændring til håndtering af dette ville kræve fundamentale ændringer af algoritmen [INH87]. Der henvises ligeså til [INH87] for en tilbagesubstitutionsalgoritme, da denne ikke er identisk med den beskrevne i afsnit 3.4.

6.3.1 Tilpasning til det systoliske array

For den optimale inddeling af A vælges mængden n_r (fra ligning 6.5) således, at der vil være færrest elementer at arbejde med. Denne opdeling er dog ikke altid velegnet til et systoliske array, da det i den beskrevne form kun er i stand til at håndtere matricer af en fast størrelse. Det blev antaget at det systoliske array blev bygget til en matrix med kardinalitet 256, hvilket vil sige, at n_j (kardinaliteten af A_r) maksimalt kan være 256.

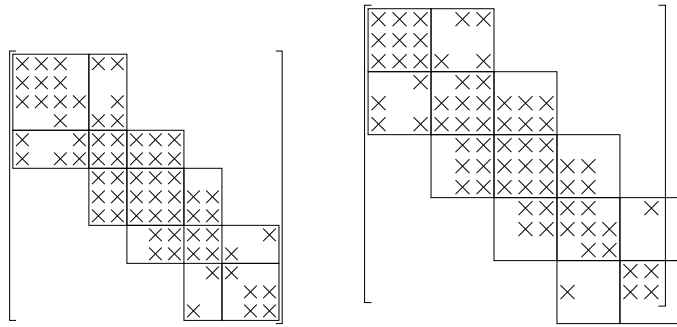
Heldigvis er der flere ting, der hjælper med at få n_r ned på et niveau, hvor det systoliske array kan følge med. For det første vil strukturen af matricen ofte gentage sig hierakisk. Dermed kan der "zoomes" ind på en blok og se strukturen af A gentaget en eller flere gange. På denne måde kan n_r i de fleste tilfælde formindskes. For det andet; hvis A_n er en båndmatrix med diagonalbredde \leq det systoliske arrays størrelse, kan A_n løses direkte med det systoliske array. Dette er muligt, da det systoliske array ikke ser forskel på, om der arbejdes med en fuld matrix eller med en båndmatrix. Derfor vil det systoliske array, i sin enkleste form, kunne LU transformere båndmatricen uden ændring af hardware. Hvis n_r fortsat er for stor, kan der LU faktorerises med den almindelige blok LU faktorisering (fra afsnit 3.3.2).

Figur 6.3 viser, en opdeling af figur 6.2 i blokke svarende til ligning 6.4. Den første figur viser, hvorledes en optimal inddeling kunne se ud, og den anden opdelingen til et systolisk array af størrelsen 3×3 .

6.3.2 Parallelisering

For at parallelisere godt skal et problem kunne deles op i flere mindre uafhængige enheder. Multiniveau metoden opdeler netop problemet således og paralleliserer derfor åbenlyst godt. Det viser sig da også, at denne metode på et stort antal processorer, paralleliserer bedre end standard LU faktorisering [INH87].

Ud fra ligning 6.4 indføres matricerne Q_n for $n = 1, 3, \dots, m$ i stedet for \tilde{A}



Figur 6.3 – Afhængigheder af komponenter fra en Newton-Raphson iteration. Kasserne i venstre figur symboliserer en optimal opdeling og den højre figur en opdeling til et systoliske array. Det er meningen, at kassen omkring ikke passer i højre figur, da det illustrerer en suboptimal inddeling.

(ligning 6.6):

$$\begin{aligned}
 Q_1 &= \begin{bmatrix} A_1 & B_1 \\ C_2 & A_2 \end{bmatrix} \\
 Q_r &= \begin{bmatrix} A_r & B_r & C_r \\ C_{r+1} & A_{r+1} & E_{r+1} \\ B_{r-1} & D_{r-1} & 0 \end{bmatrix} \quad \text{For } r=3,5,\dots,m-2. \\
 Q_m &= \begin{bmatrix} A_m & C_m \\ B_{m-1} & 0 \end{bmatrix}
 \end{aligned} \tag{6.9}$$

Nu kan man parallelt faktorisere Q til matricerne \bar{Q} :

$$\begin{aligned}
 \bar{Q}_1 &= \begin{bmatrix} L_1 U_1 & \bar{B}_1 \\ \bar{C}_2 & \hat{A}_2 \end{bmatrix} \\
 \bar{Q}_r &= \begin{bmatrix} L_r U_r & \bar{B}_r & \bar{C}_r \\ \bar{C}_{r+1} & \hat{A}_{r+1} & \bar{E}_{r+1} \\ \bar{B}_{r-1} & \bar{D}_{r-1} & \tilde{A}_{r-1} \end{bmatrix} \quad \text{For } r=3,5,\dots,m-2. \\
 \bar{Q}_m &= \begin{bmatrix} L_m U_m & \bar{C}_m \\ \bar{B}_{m-1} & \tilde{A}_{m-1} \end{bmatrix}
 \end{aligned} \tag{6.10}$$

med samme definitioner som i ligning 6.8 samt

$$\begin{aligned}
 \hat{A}_s &= A_s - \bar{C}_s \bar{B}_{s-1} \\
 \tilde{A}_s &= -\bar{B}_s \bar{C}_{s+1}
 \end{aligned} \tag{6.11}$$

og derfra genereres

$$\bar{A}_s = \hat{A}_s + \tilde{A}_s \quad \text{for } s = 2, 4, \dots, m-1 \tag{6.12}$$

Processor/ Beregningsniveau	0	1	2	3	4	5	6	7
1	Q_1	Q_3	Q_5	Q_7	Q_9	Q_{11}	Q_{13}	Q_{15}
2	Q_2	-	Q_6	-	Q_{10}	-	Q_{14}	-
3	Q_4	-	-	-	Q_{12}	-	-	-
4	Q_8	-	-	-	-	-	-	-

Figur 6.4 – Processor fordeling ved beregning af en matrix med $m = 15$ med 8 processorer.

På denne måde distribueres multilevel algoritmen på en simpel måde til flere computere eller systoliske array. Beregningen af \bar{Q}_n kan også paralleliseres ved udnyttelse af uafhængighederne mellem komponenterne.

6.3.3 Udvidelse af systolisk array

Når \bar{A}_r beregnes fra 6.9 til 6.10, laves der 5 matrix additioner og 7 matrix multiplikationer for hver LU faktorisering, da beregningen af \bar{B} , \bar{C} , \bar{D} , \bar{E} og \bar{A}_s for $s = 2, 4, \dots, m - 1$ skal udføres.

Det betyder, at selv med en uendelig hurtig LU faktorisering, vil hastigheden ikke blive hjulpet med mere end 1/13 målt i antal matrix operationer. For at hjælpe på dette, kan man opbygge et anderledes systolisk array.

Det er, som nævnt tidligere muligt, at bygge systoliske array, der laver matrix multiplikation og addition svarende til LU faktoriseringsarrayet (se afsnit 5.1). Dermed kan der bygges en mængde systoliske array sat sammen således, at de beregner Q_n . Dette specielle systoliske array kunne enten bestå af flere systoliske array, hvor nogle håndterer matrix multiplikation, andre matrix addition og til sidst en enkelt der LU faktorerer. Alternativt kunne benyttes en af de tidligere omtalte arkitekturer som RAW til dynamisk at omkonfigurere det systoliske array til at klare alle opgaver.

Dermed vil en løsning af et Q_n til \bar{Q}_n kunne foregå på denne måde:

1. Beregn LU faktoriseringen af A_r og gem L og U i et lager på det systoliske array og i hovedlageret.
2. Beregn \bar{B}_r ved hjælp af en matrix multiplikation og gem denne på det systoliske array og i hovedlageret.
3. Beregn som (2) \bar{C}_{r+1} .
4. Beregn \hat{A}_{r+1} ved hjælp af en matrix multiplikation og en matrix addition og gem resultatet i hovedlageret.
5. Beregn som (2) \bar{B}_{r-1} og smid L ud af det systoliske arrays lager.

6. Beregn som (4) \bar{D}_{r-1} og smid \bar{B}_r ud af det systoliske arrays lager.
7. Beregn som (2) \bar{C}_r og smid U ud af det systoliske arrays lager.
8. Beregn som (4) \bar{E}_{r+1} og smid \bar{C}_{r+1} ud af det systoliske arrays lager.
9. Beregn som (4) \bar{A}_{r-1} , men gem resultatet på det systoliske array og smid \bar{C}_r og \bar{B}_{r-1} ud af det systoliske arrays lager.
10. Hvis det nuværende array er udset til at beregne det næste niveau af Q_n beregnes \bar{A}_{r-1} med en matrix addition. Resultatet gemmes i det systoliske array. Hvis dette array ikke er sat til andet, stoppes beregningerne her.
11. Det næste niveau Q_n kan nu beregnes med \bar{A}_{r-1} fra (10) ved at hoppe til (1). Dog spares overførelsen til det systoliske array da matricen allerede ligger i det systoliske arrays lager.

Hvert trin kræver, at der bliver hentet maksimalt én matrix ind til beregningen, alle andre komponenter ligger allerede parat i det systoliske arrays lager. Desuden vil det højst være nødvendigt at have en LU enhed (trin 1), en matrix multiplikation (trin 2, 3, 5, 7), en matrix addition (trin 9) eller både en multiplikation og en addition (trin 4, 6, 8). Der vil maksimalt være brug for at opbevare 5 matricer på det systoliske array. Ved en større båndbredde til lageret ville det være muligt at parallelisere flere af trinene, ganske som de kan paralleliseres ud på flere computere. Hvis der var mere lager kunne flere overførelser fra lageret undgås, da flere af de beregnede elementer end \bar{A}_{r-1} skal bruges i næste iteration.

For at kunne holde disse 5 matricer skal der bruges et lager. Hvis elementernes længde antages at være 64 bit og af matricer af kardinalitet 256 skal der bruges $5 * 64 * 256 * 256 = 20$ Mbit, hvilket er i en størrelsesorden, hvor det sagtens kan ligge på kredsløbet. Desuden kan man nemt gøre båndbredden af dette lager en hel del større, da man ikke har mange af de samme begrænsninger som et standard lager.

Køretiden for løsning af et Q_n er da kommet ned på det mest optimale, set fra antallet af gange elementerne tilgås fra hovedlageret, da kun én læsning af elementerne fra lageret og én skrivning tilbage er nødvendig. Hvilket betyder at det kører i $O(N_Q)$ tid, hvor N_Q er kardinalitet af Q_n .

Det antages, at der findes et ligningssystem H af størrelsen $N = 2^d - 1$. Det er optimalt opbygget, således at opdelingen i blokkene Q_n er perfekt til et systolisk array af størrelse M . Det antages, at det yderste niveau af Q_n allerede er opbygget. Antallet af niveauer der kræves for at løse dette system er $\max(1, d + 2 - M)$. Ved brug af $2^{\max(1, d + 2 - M)}$ systoliske array er det muligt at få køretiden af hvert niveau ned i $O(M)$ tid, men da M er konstant er det $O(1)$. Da der skal løses $O(d)$ niveauer bliver den samlede køretid $O(d) = O(\log_2(N))$.

Kapitel 7

Konklusion

I denne rapport blev der fokuseret på arkitekturen systolisk array til at LU faktorisere lineære ligningssystemer meget effektivt i hardware. Mange af de vigtige problemstillinger omkring en implementation er belyst.

Det er sandsynliggjort hvordan lineære ligningssystemer opstår i kredsløbs analyse og det klassiske matematiske grundlag er blevet gennemgået. Der er diskuteret en række arkitekturer med henblik på at undersøge hvilke, som ville være velegnede til løsning af lineære ligningssystemer.

Det er vist, at et lineært ligningssystem som passer i et systolisk array, kan LU faktoreres med arrayet flere hundrede gange hurtigere end en uniprocessor-implementation.

Der er præsenteret to algoritmer, der ved udnyttelse af egenskaber ved specielle store tyndtbefolkede ligningssystemer kan udnytte et systolisk array til at opnå store hastighedsforøgelser. For den ene algoritme, er der vist en parallel udvidelse, som udnytter flere systoliske array på samme tid og som ideelt kan LU faktoriseringen i $O(\log_2(N))$ tid.

7.1 Yderligere arbejde

Inden for rammerne af dette projekt, er en række spændende problemer og resultater undersøgt. Emnet er ikke udtømmende beskrevet, og der er stadig mange løse ender, der kan arbejdes videre på.

Her er præsenteret to algoritmer til opdeling af store ligningssystemer med en speciel struktur. Det kunne være spændende at undersøge om og hvordan en tilsvarende algoritme kunne udformes til store generelle ligningssystemer, og som kunne udnytte et eller flere systoliske array, kunne udformes.

Den systoliske array idé er ikke kun begrænset til LU faktorisering. Det kunne derfor være interessant, at se på systoliske array til løsning af matrix multiplika-

tion, tilbagesubstitution m.fl. da det er en af komponenterne til yderligere store hastighedsforøgelser.

Det systoliske array til LU faktorisering er kun blevet præsenteret som teori. For at kunne arbejde videre mod en hardware implementation, vil et langt mere udførligt design være nødvendigt. Nogle af de problemstillinger, som kunne arbejdes med er f.eks. implementationen af en reciprokcelle, der er i stand til at klare arbejdet i samme tid som de andre celler. En multiplikationsenhed, der ikke bruger $O(N)$ lag logik, ville også være nærliggende. Desuden kunne der ses på, hvordan cellerne skal udformes for at kunne reducere antallet af celler i et array til en trediedel, som teorien forudsiger det er muligt.

I forbindelse med den numeriske ustabilitet kunne det undersøges hvordan dette problem afhjælpes. Enten ved at se på hvordan den overordnede algoritme, kan sikre at LU faktoriseringen går godt, eller ved at se på hvad en intern udvidelse til flere præcisionsbit vil kunne give.

Hvis de rekonfigurerbare computere bliver en succes, kunne det være interessant, at se på hvordan et systolisk array kunne implementeres i en sådan arkitektur.

Det næste naturlige skridt i udforskningen af LU faktorisering med et systolisk array, vil være at implementere den overordnede software, samt at forsøge at simulere en virkelig hardware implementation af et systolisk array. Samspillet mellem disse to dele er afgørende for, at det systoliske array kan anvendes til at løse store ligningssystemer.

Litteraturliste

Ligningssystemer

- [WJM88] William J. McCalla: "*Fundamentals of Computer-Aided Circuit Simulation*", Kluwer Academic Publishers, 1988, ISBN: 0-89838-248-3.
- [DW96] David Kincaid: "*Numerical analysis*", 2. udgave, Brooks/Cole, 1996, ISBN: 0-534-33892-5.
- [RM93] Robert Messer: "*Linear algebra - Gateway to mathematics*", Harper Collins 1993.
- [ATINET] BLAS: "*Atlas*" <http://www.netlib.org/atlas/>
- [INH87] Ibrahim N. Hajj og Stig Skelbo: "*A Multilevel Parallel Solver for Block Tridiagonal and Banded Linear Systems*", diku tryk, 1987.
- [DH76] D. Heller: "*Some aspects of the cyclic reduction algorithm for block tridiagonal systems*", SIAM J. Numer. Anal. 13, 1976.
- [RD93] Richard C. Dorf (editor-in-chief): "*The electrical engineering handbook*", CRC Press Inc., 1993, ISBN: 0-8493-0185-8
- [TI97] Timothy A. Davis, Iain S. Duff: "*A Combined Unifrontal/Multifrontal Method for Unsymmetric Sparse Matrices*", Technical Report TR-97-016, University of Florida, 1997.

Hardware implementation

- [EW97] Elliot Waingold et. al: "*Baring It All to Software: RAW Machines*", IEEE Computer, September 1997.
- [CM80] Carver A. Mead: "*Introduction to VLSI Systems*", Addison-Wesley, 1980, ISBN: 0-201-04358-0.
- [MITINET] MIT: "*RAW*" <http://www.cag.lcs.mit.edu/raw/>
- [SBSINET] Star Bridge Systems: "*Pensa*" <http://www.starbridgesystems.com/>
- [ITINET] Intel: "*Itanium*" <http://developer.intel.com/design/IA-64/>

Bilag B

Synopsis

Indledning

Dette er synopsisen til Batchelor projektet på Datalogi på Datalogisk Institut Københavns universitet. Nedestående er hvad studienævnet synes denne skal indeholde:

Synopsis skal

- specificere og motivere de studerendes valg af emne
- angive den litteratur, der er fundet
- indeholde en plan over det videre forløb.

Synopsis er primært en hensigtserklæring, der beskriver, hvad gruppen har tænkt sig at bruge resten af forløbet til. Synopsis er ikke bindende for gruppen. Synopsis må gerne indeholde spørgsmål, gruppen vil prøve at besvare i det videre forløb. Synopsis bør ikke indeholde resultater.

Synopsis

Projektets titel er "Hardware implementation af parallel løsning af lineære ligningssystemer". Mere specifikt er det de ligningssystemer man får ved at simulere digitale kredsløb. Vores tese er, at en speciel hardware implementation kan løse et lineært ligningssystem hurtigere end en konventionel processor.

Vi vil ikke beskæftige os med opstilling og beskrivelse af kredsløb. Istedet vil vi benytte os af programmet ESACAP, som til et givet tidspunkt kan aflevere jakobimatricen svarende til et system af knudepunktligninger. Det er disse ligningssystemer vi ønsker at løse. Vi vil gerne undersøge i hvilken grad det er muligt at løse disse ligningssystemer enten iterativt eller direkte.

Kapitel B: Synopsis

Vi vil udnytte at det er muligt at implementere flere CPU'er på en chip sammenkoblet i et netværk, hvilket vil være det samme som at lave samme løsning i parallel på flere maskiner på én gang, men hvor kommunikationsomkostningerne er meget små. Desuden vil vi undersøge muligheden for at lave én processer med en stor mængde eksekveringsenheder i stedet, i dette tilfælde mange flydende kommatalsenheder.

Vi ønsker at designe en chip til hardware implementationen. Vores mål er at lave et design, som er så konkret at vi kan udforme en prototype. Vi overvejer to mulige prototyper. Enten et C++ program, som vil kunne vise vigtige egenskaber ved vores design, eller en implementation i beskrivelsessproget VHDL (et professionelt værktøj til design og implementation af digitale kredsløb alá Sim-Sys). En VHDL løsning vil kræve et langt mere detaljeret design af det omkringliggende netværk end en C++ prototype, men vil kunne give mere realistisk bud på hastighedsfordele ved en hardware implementation. En VHDL løsning vil ydermere kunne simuleres på specifikt udviklings udstyr.

Batchelor projektet udarbejdes i samarbejde med Silicide A/S som stiller udstyr (herunder VHDL udstyr) og arbejdspladser til rådighed.

Tidsplan

Skemaet indeholder vore forløbige tidsplan for projektføreløbet.

Uge	10	11	12	13	14	15	16	17	18	19	20
synopsis	X	X									
synopsisforsvar		X	X								
algoritme analyse/rapport			X	X							
hardware analyse/rapport				X	X						
implementation						X		X	X	X	
páskeferie						X	X				
rapportafslutning/resultater										(X)	X
aflevering											X

Valg af litteratur

Vi har endnu ikke lagt os fast på de endelige kilder og hvad der er relevant i de allerede fundne kilder. Vores litteratursøgning har indtil nu sigtet mod at finde så meget litteratur, som så relevant ud. Derfor er nedenstående liste ikke en komplet oversigt over hvad vi vil læse.

Bøger som indtil nu ser særlig relevante ud:

- **EVV94** Gode afsnit om parallel løsning af denne type problemer

- **ES92** Artikler fra et seminar om løsning af lineære ligningssystemer. F.eks. ”Gaussian Elimination on Distributed Memory”
- **SY98** Blød introduktion til VHDL
- **EW97** God introducerende artikel. Inspirationskilde til arkitekturen.

Løsning af ligningssystemer

EVV94 Eric F. Van de Velde: “*Concurrent Scientific Computing*”, Springer-Verlag, 1994, ISBN: 0-387-94195-9.

ES92 Emilio Spedicato: “*Computer Algorithms for Solving Linear Algebraic Equations*”, Springer-Verlag, 1992, ISBN: 3-540-54187-X.

BW99 Barry Wilkinson: “*Parallel Programming*”, Prentice Hall, 1999, ISBN: 0-13-671710-1.

WJM88 William J. McCalla: “*Fundamentals of Computer-Aided Circuit Simulation*”, Kluwer Academic Publishers, 1988, ISBN: 0-89838-248-3. (Kap 3.1 - Sparse Matrix Storage).

DW96 David Kincaid: “*Numerical analysis*”, 2. udgave, Brooks/Cole, 1996, ISBN: 0-534-33892-5. (afsnit 4.8 analysis of roundoff error in the gaussian algorithm).

MF94 Ming Fang: “*Iterative Methods for...*”, ph.d. afhandling DIKU 1994.

Hardware implementation

SY98 Sudhakar Yalamanchili: “*VHDL Starter’s Guide*”, Prentice Hall, 1998, ISBN: 0-13-519802-X.

SHR99 Seyed H. Roosta: “*Parallel Processing and Parallel Algorithms*”, Springer-Verlag, 1999, ISBN: 0-387-98716-9. (Inter connection networks, mulig kandidat)

EW97 Elliot Waingold: “*Baring It All to Software: RAW Machines*”, IEEE Computer, September 1997.

SS00 Sundararajan Sriram: “*Embedded Multiprocessors*”, Marcel/Dekker, 2000, ISBN: 0-8247-9318-8.

WB01 William Buchanan: “*Advanced PC Architecture*”, Addison-Wesley, 2001, ISBN: 0-201-39858-3. (Noget PCI busser, mulig kandidat).

CM80 Carver A. Mead: “*Introduction to VLSI Systems*”, Addison-Wesley, 1980, ISBN: 0-201-04358-0.

Kredsløbssimulering

LTP94 Lawrence T. Pillage: “*Electronic Circuit and System Simulation Methods*”, McGraw-Hill, 1994.

