The HIPERFIT Contract Prototype: A Fully Certified Financial Contract Engine

for Managing and Parallel Pricing of Over-the-Counter Portfolios

HIPERFIT / CFIR / DIKU Business Club event
June 25, 2015

Martin Elsman
Department of Computer Science
University of Copenhagen (DIKU)
What is HIPERFIT?

Research Center funded by the Danish Council for Strategic Research (DSF) in cooperation with financial industry partners:

**HIPERFIT**: Functional High Performance Computing for Financial IT.

- Six years lifespan:
- Funding volume: **5.8M EUR**.
- 78% funding from DSF, 22% from partners and university.
- 8(10+) Ph.D. + 2(0) post-doctoral positions (CS and Mathematics).
- Funding for collaboration with small/medium-sized businesses.
HIPERFIT Projects and Vision

**Financial Contract Specification (DIKU, IMF)**
Use declarative combinators for specifying and analyzing financial contracts.

**Automatic Loop Parallelization (DIKU)**
Outperform commercial compilers on a large number of benchmarks by parallelizing and optimizing imperative loop structures.

**Parallelization of Financial Applications (DIKU, LexiFi)**
Analyze real-world financial kernels, such as exotic option pricing, and parallelize them to run on GPGPUs.

**Streaming Nested Data Parallelism (DIKU)**
Reduce space complexity of "embarrassingly parallel" functional computations by streaming.

**Risk (IMF, DIKU, SimCorp)**
Parallelize calculation of VaR and exposure to counterparty credit risk.

**Bohrium (NBI)**
Collect and optimize bytecode instructions at runtime and thereby efficiently execute vectorized applications independent of programming language and platform.

**APL Compilation (DIKU, Insight Systems, SimCorp)**
Develop techniques for compiling arrays, specifically a subset of APL, to run efficiently on GPGPUs and multi-core processors.

**Key-Ratios by AD (DIKU)**
Use automatic differentiation for computing sensibilities to market changes for financial contracts.

**Optimal Decisions in Household Finance (Math, Nykredit, FinE)**
Develop quantitative methods to solve individual household's financial decision problems.

**Big Data – Efficient queries (DIKU, SimCorp)**
Parallelize big data queries.
HIPERFIT Projects and Vision

**Financial Contract Specification (DIKU, IMF)**
Use declarative combinators for specifying and analyzing financial contracts.

**Automatic Loop Parallelization (DIKU)**
Outperform commercial compilers on a large number of benchmarks by parallelizing and optimizing imperative loop structures.

**Parallelization of Financial Applications (DIKU, LexiFi)**
Analyze real-world financial kernels, such as exotic option pricing, and parallelize them to run on GPGPUs.

**Streaming Nested Data Parallelism (DIKU)**
Reduce space complexity of "embarrassingly parallel" functional computations by streaming.

**Risk (IMF, DIKU, SimCorp)**
Parallelize calculation of VaR and exposure to counterparty credit risk.

**Bohrium (NBI)**
Collect and optimize bytecode instructions at runtime and thereby efficiently execute vectorized applications independent of programming language and platform.

**APL Compilation (DIKU, Insight Systems, SimCorp)**
Develop techniques for compiling arrays, specifically a subset of APL, to run efficiently on GPGPUs and multi-core processors.

**Key-Ratios by AD (DIKU)**
Use automatic differentiation for computing sensibilities to market changes for financial contracts.

**Optimal Decisions in Household Finance (Math, Nykredit, FinE)**
Develop quantitative methods to solve individual household's financial decision problems.

**Futhark**
A Functional Data-Parallel Programming Language

**Big Data – Efficient queries (DIKU, SimCorp)**
Parallelize big data queries.
Why a HIPERFIT Prototype Framework?

**Motivation:** Develop a framework that allows for experimenting with solutions to key challenges in the financial industry, including contract management, portfolio analytics, and parallel Monte Carlo techniques for contract and portfolio evaluation and for calculating risk measures.

**Benefits of a Prototype**
1. Research results to the test
2. Projects unite
3. Visibility
4. Student activities
5. Giving back to society (open source)

**HIPERFIT Goal:** In two years time, we would like our partners, and industrial peers, to look towards HIPERFIT to find parallel (i.e., scalable) techniques for solving demanding computational problems within the domain of finance.
Prototype Ideas

Main idea: Build a solution for managing and pricing over-the-counter (OTC) financial contracts.
( resembling LexiFi Apropos and SimCorp’s XpressInstruments)

1. Build the system around the concept of a “live” portfolio of contracts.

2. As time goes by, the portfolio evolves according to a reduction semantics for contracts.

3. Allow the portfolio to be priced (i.e., valuated) at any chosen point in time (e.g., yesterday, now, or tomorrow).

4. Give the user good performance and loads of features… : )
LexiFi/SimCorp style **contract combinators** for specifying financial derivatives [1].

**Contract kernel** written in Coq, a functional language and proof assistant for establishing program properties (verified correctness wrt a cash-flow denotational semantics).

**Certified management code** extracted from the Coq implementation (fixings, decisions).

**Valuation/pricing**: payoff functions extracted from contracts (input to stochastic pricing engine).

---

**American Option contract in natural language:**

At any time within the next 90 days, party X may decide to buy USD 100 from party Y, for a fixed rate $r=6.65$ of Danish Kroner.

**Specified in the contract language:**

\[
\text{if } \text{obs}(X \text{ exercises option) } \text{ within 90 then}
\]

\[
100 \times (\text{USD}(Y \rightarrow X) \& 6.65 \times \text{DKK}(X \rightarrow Y))
\]

\[
\text{else } \emptyset
\]

---


The Contract Language

Features:

**Compositionality**
Contracts are time-relative \(\Rightarrow\) compositionality

**Multi-party**
Possibility for specifying portfolios

**Contract management**
Contracts can be managed (fixings, splits, …)
Contracts gradually reduce to the empty contract

**Contract utilities (symbolic)**
Contracts can be analysed in a variety of ways
(find horizon, potential cash-flows, …)

Assumptions
- \(d\) integer (specifies a number of days)
- \(p\) ranges over parties (e.g., YOU, ME, X, Y)
- \(a\) assets (e.g., USD, DKK)

Expressions (extended expressions for reals and booleans)
- \(\text{obs}(l,d)\) observe the value of \(l\) (a label) at time \(d\)
- \(\text{acc}(f,d,e)\) accumulate function \(f\) over the previous \(d\) days

Contracts (\(c\))
- \(\emptyset\) empty contract with no obligations
- \(a(p_1 \rightarrow p_2)\) \(p_1\) has to transfer one unit of \(a\) to \(p_2\)
- \(c_1 \& c_2\) conjunction of \(c_1\) and \(c_2\)
- \(e \times c\) multiply all obligations in \(c\) by \(e\)
- \(d^\uparrow c\) shift \(c\) into the future by \(d\) days
- \(\text{let } x = e \text{ in } c\) bind today’s value of \(e\) to \(x\) in \(c\)

if \(e\) within \(d\) then \(c_1\) else \(c_2\) behave as \(c_1\) when \(e\) becomes true
if \(e\) does not become true within \(d\) days, behave as \(c_2\)
Asian Option

\[ 90 \uparrow \text{if } \text{obs}(X \text{ exercises option}) \text{ within } 0 \text{ then } \]
\[ 100 \times (\text{USD}(Y \rightarrow X) & (rate \times DKK(X \rightarrow Y))) \]
\[ \text{else } \emptyset \]

\text{where}
\[ rate = \frac{1}{30} \cdot \text{acc}(\lambda r. r + \text{obs}(\text{FX USD/DKK}), 30, 0) \]

\text{Notice:} the special acc-construct is used to compute an average rate.

Simple Credit Default Swap (CDS)

\textbf{The bond:}
\[ C_{\text{bond}} = \text{if } \text{obs}(X \text{ defaults, } 0) \text{ within } 30 \text{ then } \emptyset \]
\[ \text{else } 1000 \times DKK(X \rightarrow Y) \]

\textbf{Insurance:}
\[ C_{\text{cds}} = (10 \times DKK(Y \rightarrow Z)) \& \]
\[ \text{if } \text{obs}(X \text{ defaults, } 0) \text{ within } 30 \text{ then } \]
\[ 900 \times DKK(Z \rightarrow Y) \]
\[ \text{else } \emptyset \]

\textbf{Entire Contract:}
\[
\begin{align*}
C &= C_{\text{bond}} \& C_{\text{cds}}
\end{align*}
\]
Benefits of the Formal Framework

Some contract equivalences

\[ e_1 \times (e_2 \times c) \equiv (e_1 \cdot e_2) \times c \]
\[ d_1 \uparrow (d_2 \uparrow c) \equiv (d_1 + d_2) \uparrow c \]
\[ d \uparrow (c_1 \& c_2) \equiv (d \uparrow c_1) \& (d \uparrow c_2) \]
\[ e \times (c_1 \& c_2) \equiv (e \times c_1) \& (e \times c_2) \]
\[ d \uparrow \emptyset \equiv \emptyset \]
\[ r \times \emptyset \equiv \emptyset \]
\[ 0 \times c \equiv \emptyset \]
\[ c \& \emptyset \equiv c \]
\[ c_1 \& c_2 \equiv c_2 \& c_1 \]

With a netting semantics:

\[(e_1 \times a(p_1 \rightarrow p_2)) \& (e_2 \times a(p_1 \rightarrow p_2)) \equiv (e_1 + e_2) \times a(p_1 \rightarrow p_2)\]

Other benefits:

- Type system for causality
- Correctness of contract evolution (reduction semantics)
- Calendar support using observables
Parallelized version of LexiFi pricing engine [2,3].

Code ported to OpenCL, targeting GPGPUs.

Extracted contract payoff function fused into OpenCL kernel.

Market data provided by framework.

![Option Pricing Speedup on H2](image)


The prototype architecture is **simple**, yet **flexible**.

3 (4) tier architecture:
- Front-end (web client)
- Web server
- Contract management
- GPU server
Architecture of current implementation

Web browser

JavaScript

JSON, HTML

Server

Financial contract information system

Haskell

Market data

Contract in DSL

Input data and OpenCL code generation

Data providers

User data

Market data

Market data CSV

Input data

OpenCL code

GPU execution framework

FINPAR

C++, OpenCL

Data providers
A **simple** database schema for an extensible framework.

Basic entities (tables):
- user
- portfolio
- market data (db_corr, db_quotes)
- model data

Schema generated from Haskell’s Persistent library, which explains the weird naming...
1. An instrument maps instrument-specific parameter data to contracts.

2. Available instruments include, a Call option and a Rainbow option.

3. A portfolio is a set of contracts (no strategies assigned).

4. Contracts are added by instantiating instruments with parameter data (e.g., start date, strike)

5. A portfolio and its contained contracts are priced based on a pricing date and an interest rate for discounting (and for Black-Scholes drift).
Prototype DEMO

prototype.hiperfit.dk

Prototype developed primarily by HIPERFIT PhD Student Danil Annenkov
Implementation

- Available for public forking on github…

- Uses HIPERFIT’s contract and finpar github repositories as sub-modules.

- Uses the GHC generics library for generating GUIs for instruments based on instrument parameter types.

- Uses scotty web framework (based on WAI and Warp - fast Haskell web-server)

- Uses blaze-html eDSL for markup and Clay eDSL for CSS

- Uses Persistent library for type-safe database access
Performance

Pricing itself is very fast, but compiling kernel code and running it takes quite a lot of time.

The web server is also quite fast and adds almost no overhead.

<table>
<thead>
<tr>
<th># of iterations</th>
<th>Overall (finpar)</th>
<th>Build + runtime</th>
<th>Actual runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>7.134</td>
<td>5.437685</td>
<td>0.001296</td>
</tr>
<tr>
<td>1000000</td>
<td>7.156</td>
<td>5.454617</td>
<td>0.001423</td>
</tr>
<tr>
<td>10000000</td>
<td>7.165</td>
<td>5.426234</td>
<td>0.003808</td>
</tr>
</tbody>
</table>
Prototype Student Projects

- Visualize contracts, their evolution and generate textual term sheets.
- Pricing load-balancing and multi-GPU utilization.
- More instruments.
- Model implementations:
  a. FX (Garman-Kohlhagen)
  b. American optionality
  c. Hull-White
- Graphs for visualizing time series and portfolio development.
- User-defined instruments and instrument templating.
- Support for market conventions and calendars.
- Support pricing in the browser.
- Calculate correlations based on historical stock quotes.
Prototype Future Work

- Expand work on risk (Greeks, CVA, PFE).
- Formulate detailed student projects on visualization, simulation, …
- Use *Futhark* as the basis for pricing and risk calculations [6-8].
- Interface with an online stock quote API.

---

