A Prototype Framework for Parallel Valuation and Risk Calculation

Presentation at the 2014 HIPERFIT Workshop

December 10, 2014

Martin Elsman, Associate Professor, PhD

Danil Annenkov, PhD Student

Department of Computer Science
University of Copenhagen (DIKU)

Why a Prototype Framework

Goal: *In two years time, say, 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.*

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)

Requirements/drawbacks

1. Some startup cost
2. Active development required
3. Project management
4. Low short term publ. payoff
5. Work outside of domain
6. Ownership?

HIPERFIT Projects & Vision

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

Automatic Parallelization of Loop Structures (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 Semantics for Nested Data Parallelism (DIKU)
Reduce space complexity of "embarrassingly parallel" functional computations by streaming.

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

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

Futhark

A High-Level, Parallel, Functional Language

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

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

Big Data – Efficient queries (DIKU, SimCorp)
Parallelize big data queries.

Optimal Decisions in Household Finance (IMF, Nykredit, FinE)
Investigate and develop quantitative methods to solve individual household's financial decision problems.

HIPERFIT Projects & Vision

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

Automatic Parallelization of Loop Structures (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 Semantics for Nested Data Parallelism (DIKU)
Reduce space complexity of "embarrassingly parallel" functional computations by streaming.

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

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

Futhark
A High-Level, Parallel, Functional Language

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

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

Big Data – Efficient queries (DIKU, SimCorp)
Parallelize big data queries.

Optimal Decisions in Household Finance (IMF, Nykredit, FinE)
Investigate and develop quantitative methods to solve individual household's financial decision problems.

Component 1: A Certified Contract Management Engine

- 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 (correctness).
- **Certified management code** extracted from the Coq implementation (fixings, decisions).
- **Valuation/pricing**: payoff functions extracted from contracts.

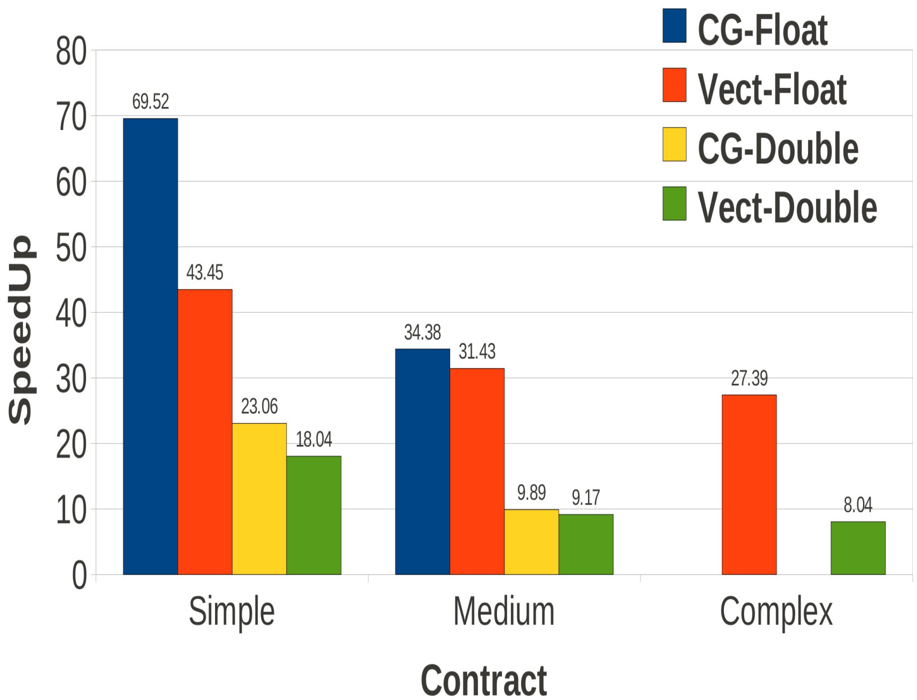
```
callOption =
  scale 1000          -- nominal
   (transl maturity
    (scale carlsb (transfOne EUR "DB" "me")))
  where
   maturity = oneYear -- 365 , using ACT/365
   strike = 50.0
   carlsb = max 0.0 (obs("Carlsberg",0)-strike)

barriertouch =     -- FX barrier touch option
  checkWithin (1.0 < obs("EUR/USD",0)) oneYear
   (scale 1000.0 (transfOne EUR "Nordea" "you"))
   zero
```

Semantics inside

[1] Patrick Bahr, Jost Berthold, and Martin Elsman. **Towards Certified Management of Financial Contracts**. In *Proceedings of the 26th Nordic Workshop on Programming Theory (NWPT'14)*. October, 2014.

Component 2: A Parallel Pricing Engine

- **Parallelized** version of LexiFi pricing engine [2,3].
- Code ported to **OpenCL**, targeting **GPGPU**s.
- Extracted payoff function fused into OpenCL kernel.



[2] Cosmin Oancea, Jost Berthold, Martin Elsman, and Christian Andreetta. **A Financial Benchmark for GPGPU Compilation**. In *18th International Workshop on Compilers for Parallel Computing (CPC'15)*. January 2015.

[3] Cosmin E. Oancea, Christian Andreetta, Jost Berthold, Alain Frisch, and Fritz Henglein. **Financial software on GPUs: between Haskell and Fortran**. In *Proceedings of the 1st ACM SIGPLAN workshop on Functional high-performance computing (FHPC '12)*. Copenhagen 2012.

Component 3: Calculating Risk

- Contract key-ratios (i.e., the greeks) calculated based on **automatic differentiation** techniques [4].

- Parallelization of portfolio **MC VaR calculations** [5].

- **Potential Future Exposure** (PFE) and **CVA** calculations:
  - Multi-party contract manipulations
    (one portfolio → one contract)
  - Algebraic manipulations/analyses (future work)

[4] Esben Bistrup Halvorsen. **Calculating Key Ratios for Financial Products using Automatic Differentiation and Monte Carlo Simulation**. DIKU M.Sc. Student Project. December 2012.

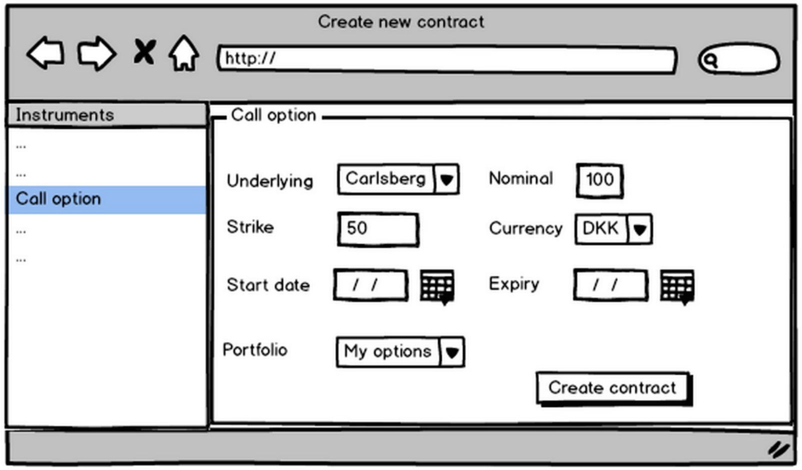
[5] Casper Holmgreen. **A Parallel Haskell Library for Computing Value-at-Risk**. M.Sc. Student Project. November 2014.

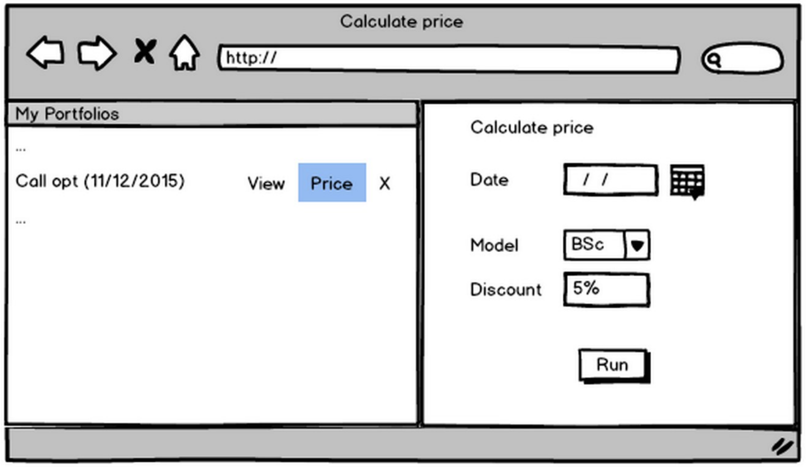
The "Low Tech" Glue – the GUI

A simple web GUI

- Instrument manager
- Portfolio manager
- Market data manager
- Pricing form

A micro-version of SimCorp Dimension / LexiFi Apropos

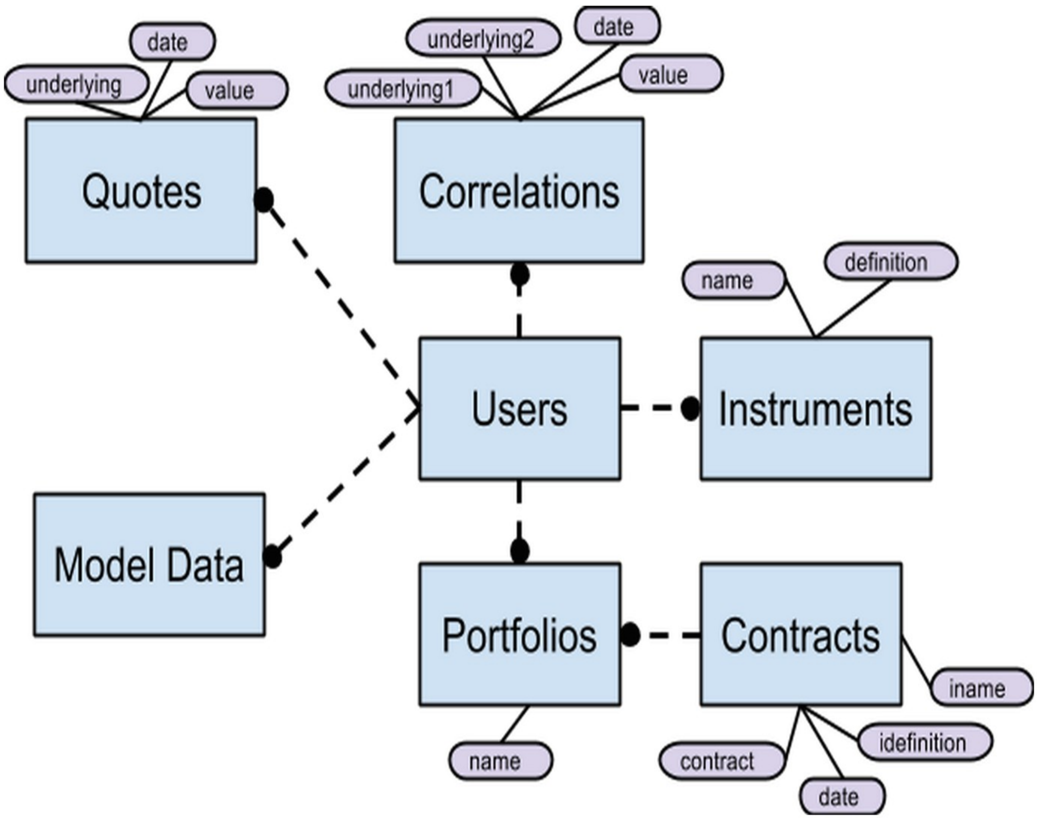




The "Low Tech" Glue – the Database

A simple DB schema
- User information
- Market data (quotes, correlations)
- Model data (calibr. data)
- Instrument templates
- Portfolio data

A micro-version of SimCorp Dimension / LexiFi Apropos

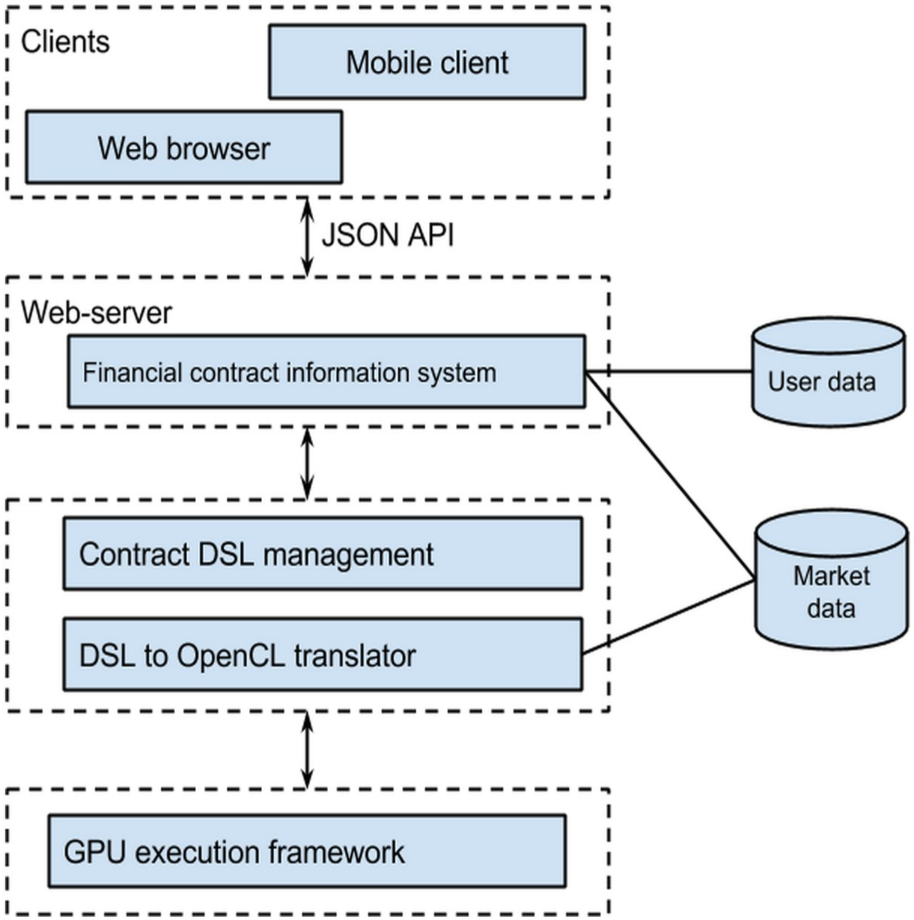


The "Low Tech" Glue – the Architecture

A simple flexible arch.

Use a scaffolding framework for getting started quickly.

A micro-version of SimCorp Dimension / LexiFi Apropos



Future Work

- Construct prototype with a rudimentary GUI.
- Expand work on risk (Greeks, CVA, PFE).
- Formulate student projects on visualization, simulation, ...
- Use ***Futhark*** implementation as the basis for pricing and risk calculations [6-8].

[6] Troels Henriksen and Cosmin E. Oancea. **A T2 Graph-Reduction Approach To Fusion**. In *2nd ACM SIGPLAN Workshop on Functional High-Performance Computing*. Boston, Massachusetts. September 2013.

[7] Troels Henriksen and Cosmin E. Oancea. **Bounds Checking: An Instance of Hybrid Analysis**. In *ACM SIGPLAN International Workshop on Libraries, Languages and Compilers for Array Programming (ARRAY'14)*. Edinburgh, UK. June, 2014.

[8] Troels Henriksen, Martin Elsman, and Cosmin E. Oancea. **Size Slicing - A Hybrid Approach to Size Inference in Futhark**. In *Proceedings of the 3rd ACM SIGPLAN workshop on Functional High-Performance Computing (FHPC'14)*. Gothenburg, SE. September, 2014.

Questions?