

Bachelor Projects in the HIPERFIT Research Center

Developing and Improving the “HIPERFIT Portfolio Management Prototype”

The “HIPERFIT Portfolio Management Prototype” is a System for Managing and Pricing Portfolios of Over-the-Counter Financial Contracts

A Bachelor's Project Teaser!
August, 2015

Danil Annenkov and Martin Elsmann (supervisors)
Department of Computer Science
University of Copenhagen (DIKU)



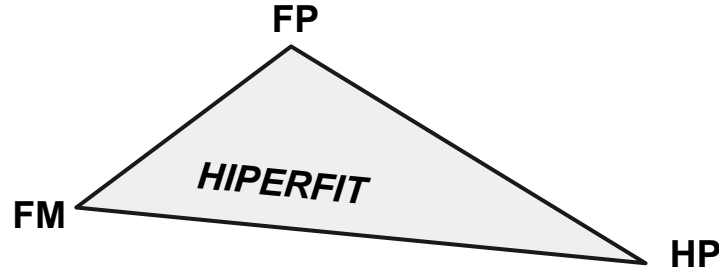
The HIPERFIT Context

The HIPERFIT Research Center conducts research (with input from our industrial partners) in the span between

FP: Functional programming and programming language semantics (e.g., type systems and domain-specific languages)

FM: Finance mathematics

HP: High-performance and parallel computing (e.g., GPGPU programming)



The Prototype: The *HIPERFIT Portfolio Management Prototype* integrates a number of HIPERFIT research efforts and seeks to demonstrate the application of the research in practice, including:

- Domain-specific languages for contract management
- Certified program development
- High-performance numeric computations on GPGPUs

Come join the HIPERFIT team in developing the system further! There are many possibilities for extending the usefulness of the system - look at the next slides for possible projects...



About the Projects

The projects emphasize the combination of

1. using core computer science techniques for solving technical computer science problems; and
2. applying some software engineering disciplines for working with group members and other groups with the aim of coordinating cross-project development (e. g., work with github and github “issues”) and cross-project design.

Students are expected to work in groups (>1) and to develop the concrete project descriptions in coordination with the supervisors.

Supervisors:

Danil Annenkov (PhD stud.)
daan@di.ku.dk

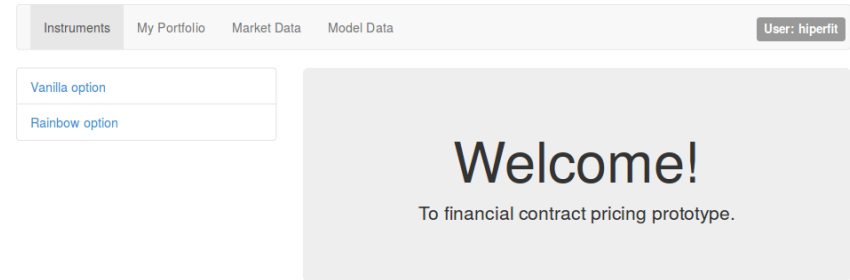
Martin Elsmann (Assoc. Prof.)
mael@di.ku.dk

**Look at the next slides for
concrete project
proposals...**



Project 1. Quotes and model parameters

- Fetch historical data from a public source available in the Internet (e.g. Google Finance API).
- Compute model input based on historical data (correlations, volatilities).
- Make the functionality available on the portfolio valuation screen in the Prototype web-interface.
- Display quote grid showing quotes stored in database.



The screenshot shows the 'Market Data' page of the financial contract pricing prototype. The user is identified as 'User: hiperfit'. The page features a navigation bar with tabs for 'Instruments', 'My Portfolio', 'Market Data', and 'Model Data'. The 'Quotes' tab is selected, and the 'Correlations' sub-tab is active. A table displays a list of quotes with columns for 'Underlying', 'Date', and 'Price'. An 'Add' button is visible next to the table.

| Underlying | Date | Price | |
|----------------------|----------------------|----------------------|-----|
| <input type="text"/> | <input type="text"/> | <input type="text"/> | Add |
| Siemens | 2008-09-01 | 46.000 | |
| BASF | 2008-09-01 | 48.000 | |
| Carlsberg | 2015-02-22 | 10.000 | |
| BASF | 2015-02-23 | 44.000 | |



Project 2. Portfolio evolution

- Fetch historical data from a public source available in the Internet (e.g. Google Finance API).
- Compute Profit and Loss (P&L) for portfolio and individual contracts.
- Display P&L on the portfolio valuation screen in the Prototype web-interface.
- Display graph showing evolution of portfolio P&L over time.

| Instruments | | | | | My Portfolio | Market Data | Model Data | User: hiperfit |
|-------------|---------------|------------|------------|--------------|--------------|-------------|------------|----------------|
| Nominal | Contract | Start | Horizon | Value | | | | |
| 1000 | VanillaOption | 2008-09-01 | 2009-04-01 | Not valuated | | | | |
| 50 | VanillaOption | 2015-01-10 | 2016-01-10 | Not valuated | | | | |
| 1 | RainbowOption | 2015-03-16 | 2015-08-06 | Not valuated | | | | |
| Total | | | | | | | | |

CurrentDate: 2015-08-25

InterestRate: 2 %

Iterations: 10000

Run valuation



Project 3. More user functionality

POWERED BY



Instruments My Portfolio Market Data Model Data User: hiperfit

- Vanilla option
- Rainbow option

Vanilla option

Nominal

StartDate

Underlying

Strike

EndDate

PutOption

Add to portfolio

View code

```
data VanillaOption = VOpt {
  underlying :: Underlying
, strike     :: Double
, endDate   :: Day
, putOption  :: Bool
} deriving (Show, Generic, Typeable)

vanillaOption =
  GUIRepr { guiLabel = "Vanilla option"
, formFields = gtoForm (Proxy :: Proxy (Rep VanillaOption))
, url       = "vanillaOption" }

makeContract startDate optData =
  let
    maturity = fromIntegral $ diffDays (endDate optData) startDate
    theObs   = obs (underlying optData, 0)
    strk    = r $ strike optData
    val     = if putOption optData
              then strk - theObs
              else theObs - strk
  in trans1 maturity (scale (maxx val 0)
    (transfOne EUR "you" "me"))
```

- Improve portfolio management in Prototype.
- Improve authentication.
- User management and access rights.
- User-specific data (quotes uploaded by user).
- Support for market conventions and calendars.
- Support for stocks in portfolio.



Additional Topics

- Generate textual term sheets.
- Pricing load-balancing and multi-GPU utilization.
- Extend instruments repository.
- User-defined instruments and instrument templating.
- Support pricing in the browser.
- Model implementations:
 - a. FX (Garman-Kohlhagen)
 - b. American optionality
 - c. Hull-White



Interested?

Please contact:

Danil Annenkov (PhD student)
daan@di.ku.dk

Martin Elsman (Associate Professor)
mael@di.ku.dk

For more information about HIPERFIT and the research conducted in the center, you may find information on the following slides and on

hiperfit.dk

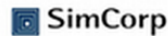


What is HIPERFIT?

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



LexiFi



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.

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.

F

uthark

A Functional Data-Parallel Programming Language

Key-Ratios by AD (DIKU)

Use automatic differentiation for computing sensitivities 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 multi-core processors.

Big Data – Efficient queries (DIKU, SimCorp)

Parallelize big data queries.

Optimal Decisions in Household Finance (Math, Nykredit, FinE)

Develop quantitative methods to solve individual household's financial decision problems.



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.

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.

Big Data – Efficient queries (DIKU, SimCorp)

Parallelize big data queries.

F

uthark

A Functional Data-Parallel Programming Language

Bohrium (NBI)

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

Key-Ratios by AD (DIKU)

Use automatic differentiation for computing sensitivities 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.

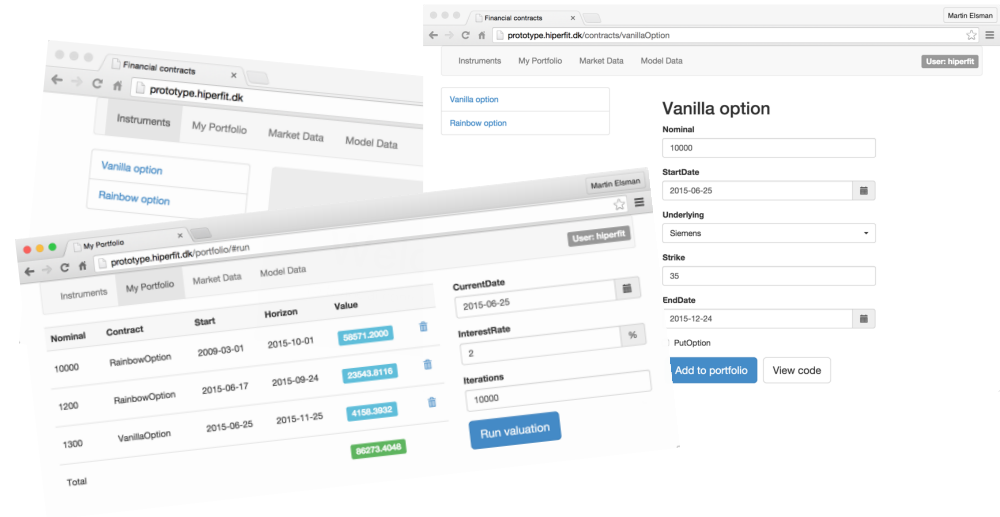


Why a HIPERFIT Prototype Framework?

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., valued) at **any chosen point in time** (e.g., yesterday, now, or tomorrow).
4. Give the user **good performance** and loads of **features... :)**



I: 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 (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:

```
if obs(X exercises option) within 90 then  
  100 × (USD(Y→X) & 6.65 × DKK(X→Y))  
else ∅
```

[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.

[2] Patrick Bahr, Jost Berthold, and Martin Elsman. **Certified Symbolic Management of Financial Multi-Party Contracts**. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP'15)*. September, 2015.



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 |
| let $x = e$ 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



Expressibility: More Contract Examples

Asian Option

90 ↑ **if** obs(X exercises option) **within** 0 **then**
 100 × (USD(Y→X) & (rate × DKK(X→Y)))
else ∅

where

rate = 1/30 · acc(λr.r + obs(FX USD/DKK), 30, 0)

Notice: the special acc-construct is used to compute an average rate.

Simple Credit Default Swap (CDS)

The bond:

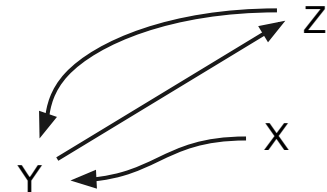
C_{bond} = **if** obs(X defaults, 0) **within** 30 **then** ∅
else 1000 × DKK(X→Y)

Insurance:

C_{cds} = (10 × DKK(Y→Z)) &
if obs(X defaults, 0) **within** 30 **then**
 900 × DKK(Z→Y)
else ∅

Entire Contract:

C = C_{bond} & C_{cds}



Benefits of the Formal Framework

Some contract equivalences

$$e1 \times (e2 \times c) \approx (e1 \cdot e2) \times c$$

$$d1 \uparrow (d2 \uparrow c) \approx (d1 + d2) \uparrow c$$

$$d \uparrow (c1 \& c2) \approx (d \uparrow c1) \& (d \uparrow c2)$$

$$e \times (c1 \& c2) \approx (e \times c1) \& (e \times c2)$$

$$d \uparrow \emptyset \approx \emptyset$$

$$r \times \emptyset \approx \emptyset$$

$$0 \times c \approx \emptyset$$

$$c \& \emptyset \approx c$$

$$c1 \& c2 \approx c2 \& c1$$

With a netting semantics:

$$(e1 \times a(p1 \rightarrow p2)) \& (e2 \times a(p1 \rightarrow p2)) \approx (e1 + e2) \times a(p1 \rightarrow p2)$$

Other benefits:

- Type system for causality
- Correctness of contract evolution (reduction semantics)
- Calendar support using observables



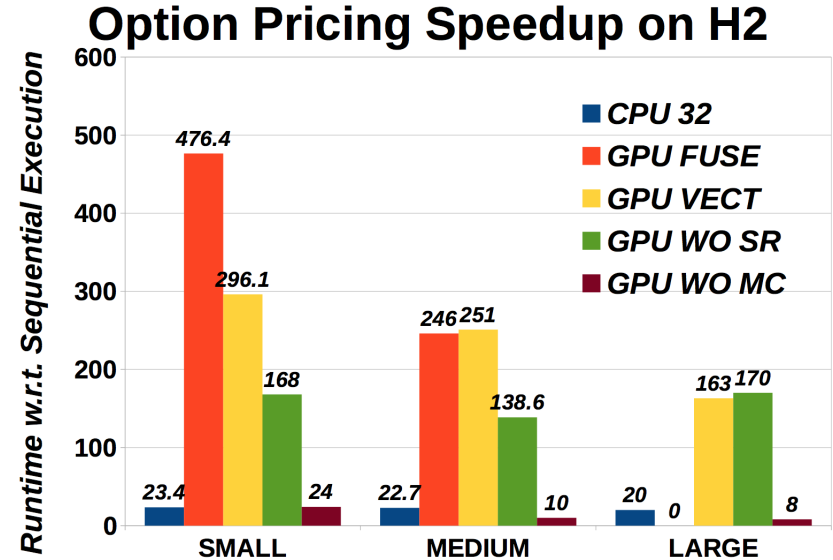
II: A Parallel Pricing Engine

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.



[3] 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.

[4] 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.

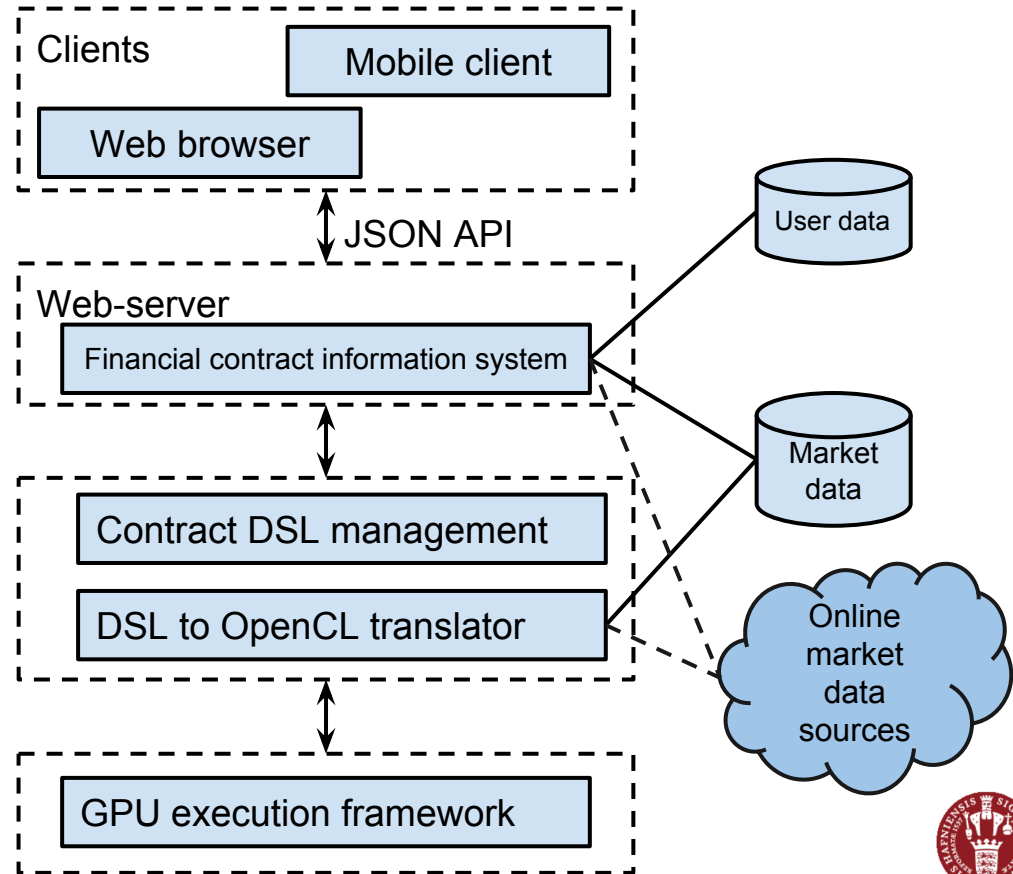


Architecture overview

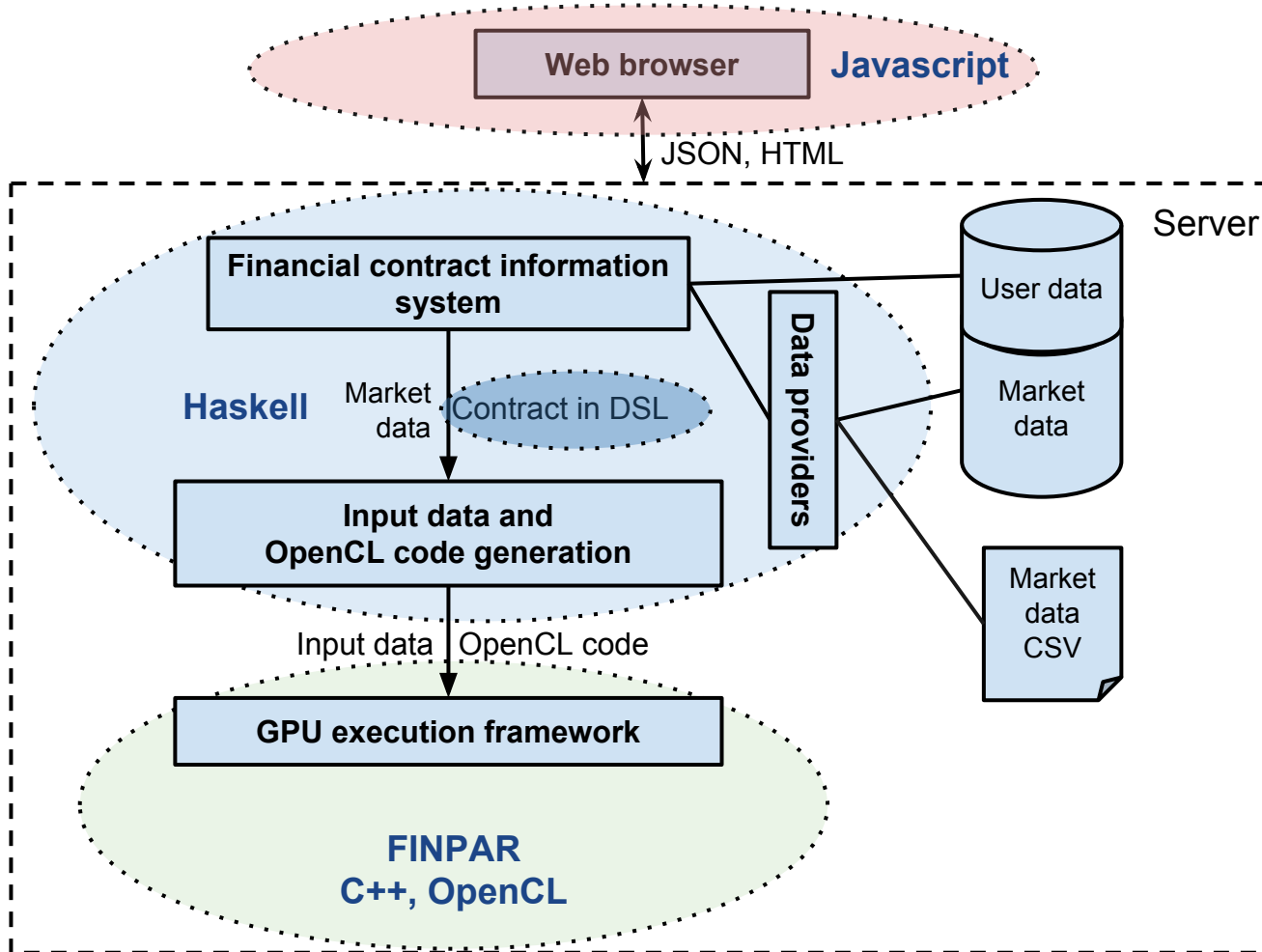
The prototype architecture is **simple**, yet **flexible**.

3 (4) tier architecture:

- Front-end (web client)
- { Web server
Contract management }
- GPU server



Architecture of current implementation



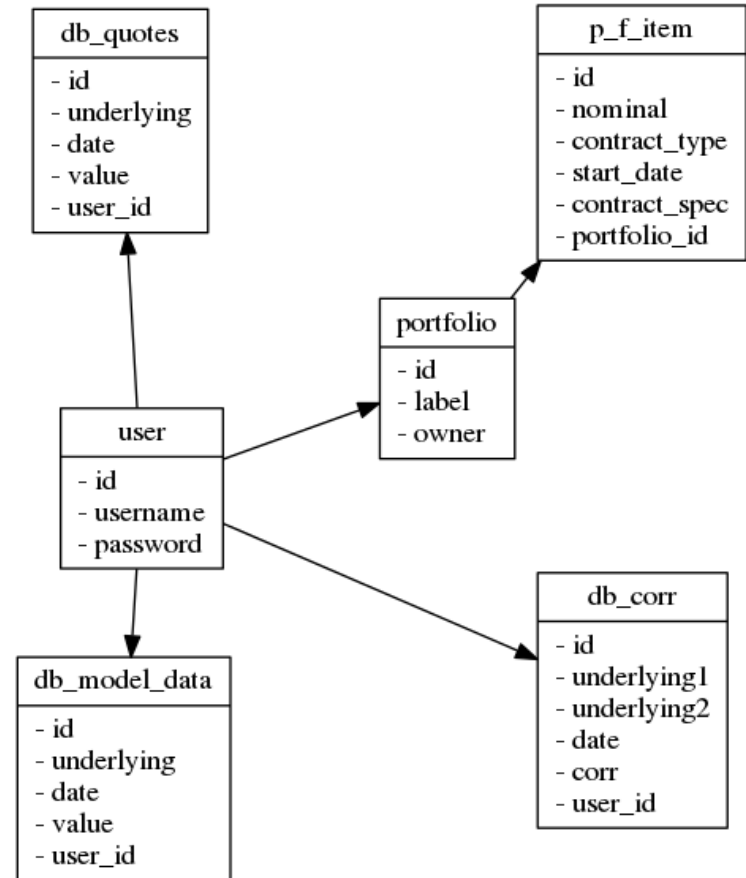
Database

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...



GUI Mockups

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).

The screenshot shows a web browser window titled "Create new contract". The address bar contains "http://". On the left, there is a sidebar labeled "Instruments" with a list containing "Call option" (highlighted in blue) and "Rainbow option". The main area is titled "Call option" and contains the following fields: Underlying (dropdown menu with "Carlsberg" selected), Nominal (input field with "100"), Strike (input field with "50"), Currency (dropdown menu with "DKK" selected), Start date (calendar icon), Expiry (calendar icon), and Portfolio (dropdown menu with "My options" selected). A "Create contract" button is located at the bottom right.

The screenshot shows a web browser window titled "Calculate price". The address bar contains "http://". On the left, there is a table with the following data:

| Nominal | Contract | Start Date | Price | |
|---------|----------------|------------|-------|--|
| 10.000 | Call Option | 2015-01-01 | | |
| 5.000 | Put Option | 2015-02-01 | | |
| 1.000 | Rainbow Option | 2015-02-01 | | |
| 2.000 | Call Option | 2015-03-01 | | |
| TOTAL | | | | |

On the right, there are input fields for "Date" (calendar icon), "Model" (dropdown menu with "BS" selected), and "Interest rate" (input field with "5%"). A "Run" button is located at the bottom right.

Prototype DEMO

prototype.hiperfit.dk

**Prototype developed primarily by
HIPERFIT PhD Student Danil Annenkov**



Implementation

Fork me on GitHub

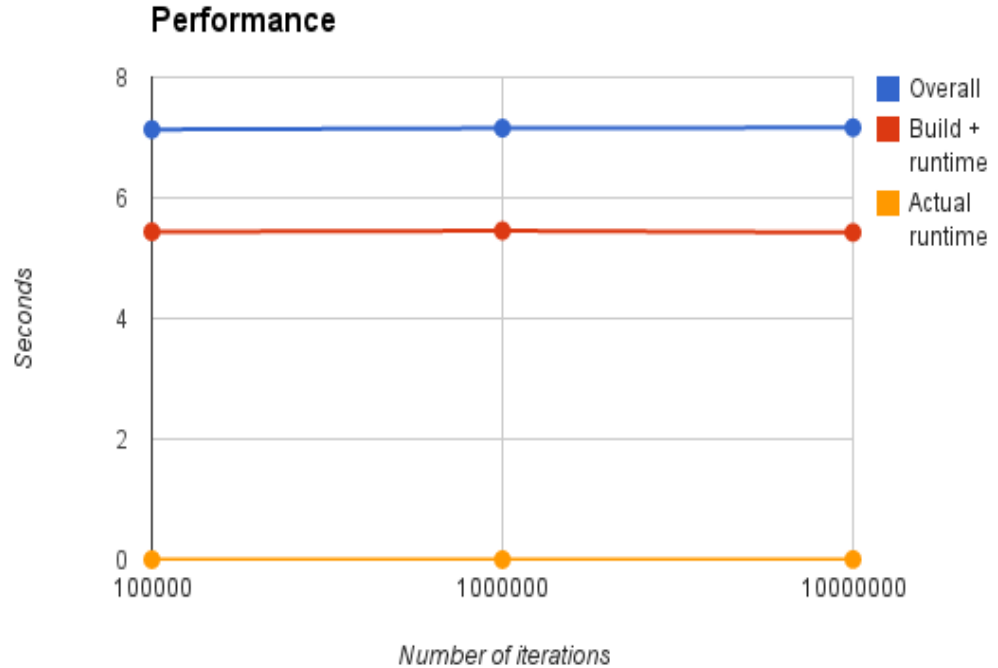
- Available for public forking on github...
- Uses HIPERFIT's contract and finpar github repositories as sub-modules.
- Uses the GHC generics library for generating GUIs for instruments based on instrument parameter types.
- Uses the scotty web framework (based on WAI and Warp - fast Haskell web-server)
- 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.



| # of iterations | Overall (finpar) | Build + runtime | Actual runtime |
|-----------------|------------------|-----------------|----------------|
| 100000 | 7.134 | 5.437685 | 0.001296 |
| 1000000 | 7.156 | 5.454617 | 0.001423 |
| 10000000 | 7.165 | 5.426234 | 0.003808 |



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.

[5] 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.

[6] 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.

[7] Troels Henriksen, Martin Elsmann, 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.

