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

Additional information about the HIPERFIT Research Center and the HIPERFIT Portfolio Management Prototype
February, 2016

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

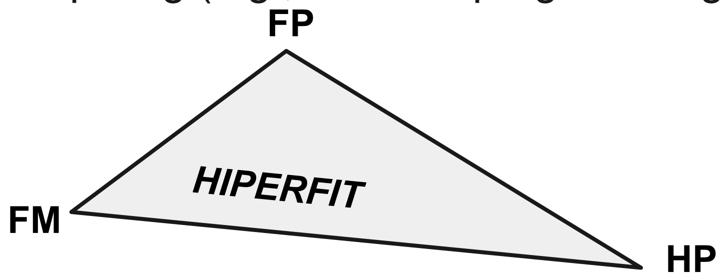
The HIPERFIT Context

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)



<u>The Prototype:</u> 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 a many possibilities for extending the usefulness of the system - look at the next slides for possible projects...

What is HIPERFIT?

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

Department of Computer Science

Department of Mathematical Sciences

Niels Bohr Institutet

Danske Bank

LexiFi

Nordea

Nykredit

SimCorp

CFIR

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

- Six years lifespan:

2011	2012	2013	2014	2015	2016

- 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.

Futhark
A Functional Data-Parallel Programming Language

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

Key-Ratios by AD (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 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.

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

Futhark
A Functional Data-Parallel Programming Language

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

Key-Ratios by AD (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 multi-core processors.

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

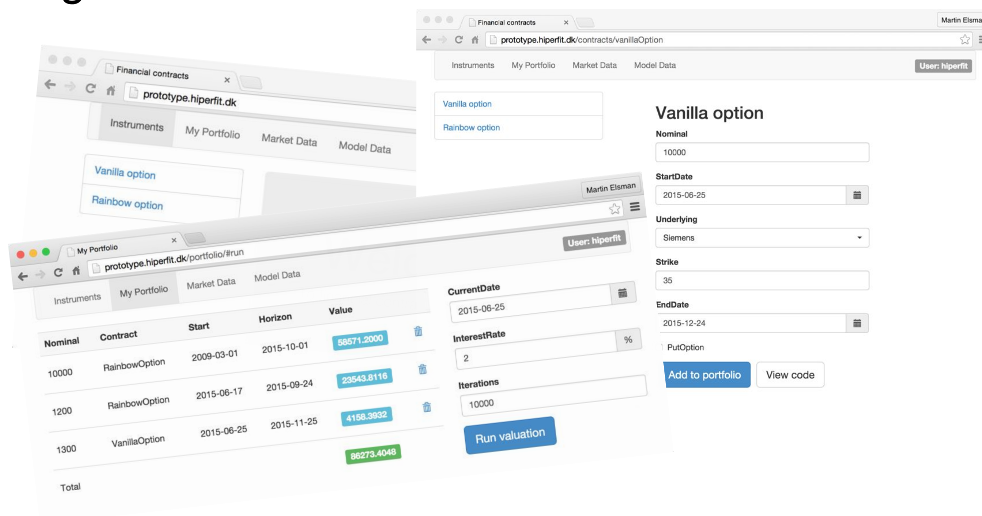
Optimal Decisions in Household Finance (Math, Nykredit, FinE)
Develop quantitative methods to solve individual household's financial decision problems.

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**… :)

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 ⇒ 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)

obs(l,d)	observe the value of l (a label) at time d
acc(f,d,e)	accumulate function f over the previous d days

Contracts (c)

∅	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 × c	multiply all obligations in c by e
d↑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:

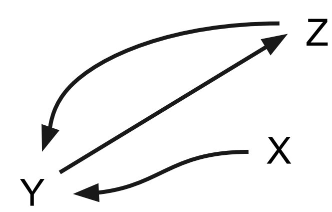
C_{cds} = (10 × DKK(Y→Z)) &

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

$e1 \times (e2 \times c)$	$\simeq$	$(e1 \cdot e2) \times c$
$d1 \uparrow (d2 \uparrow c)$	$\simeq$	$(d1 + d2) \uparrow c$
$d \uparrow (c1 \,\&\, c2)$	$\simeq$	$(d \uparrow c1) \,\&\, (d \uparrow c2)$
$e \times (c1 \,\&\, c2)$	$\simeq$	$(e \times c1) \,\&\, (e \times c2)$

$d \uparrow \emptyset$	$\simeq$	$\emptyset$
$r \times \emptyset$	$\simeq$	$\emptyset$
$0 \times c$	$\simeq$	$\emptyset$
$c \,\&\, \emptyset$	$\simeq$	c
$c1 \,\&\, c2$	$\simeq$	$c2 \,\&\, c1$

With a netting semantics:

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

Other benefits:

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

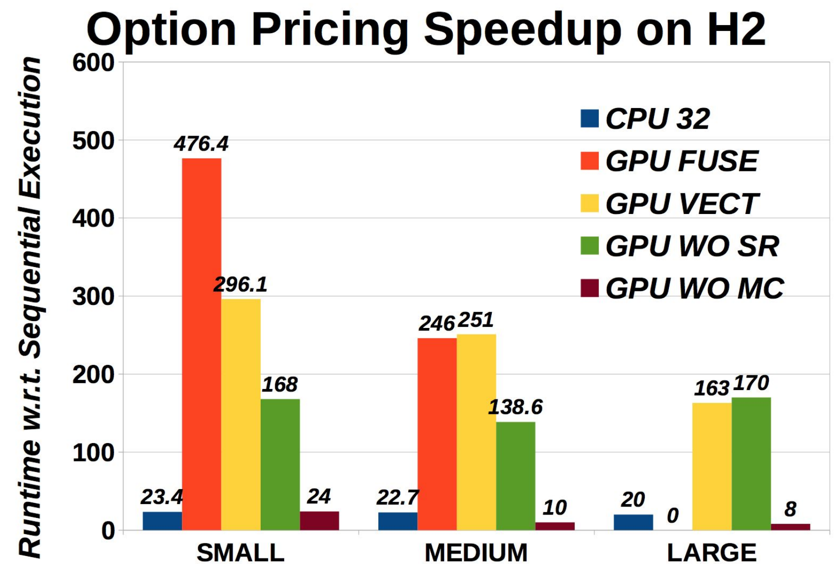
II: A Parallel Pricing Engine

Parallelized version of LexiFi pricing engine [2,3].

Code ported to **OpenCL**, targeting **GPGPU**s.

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.

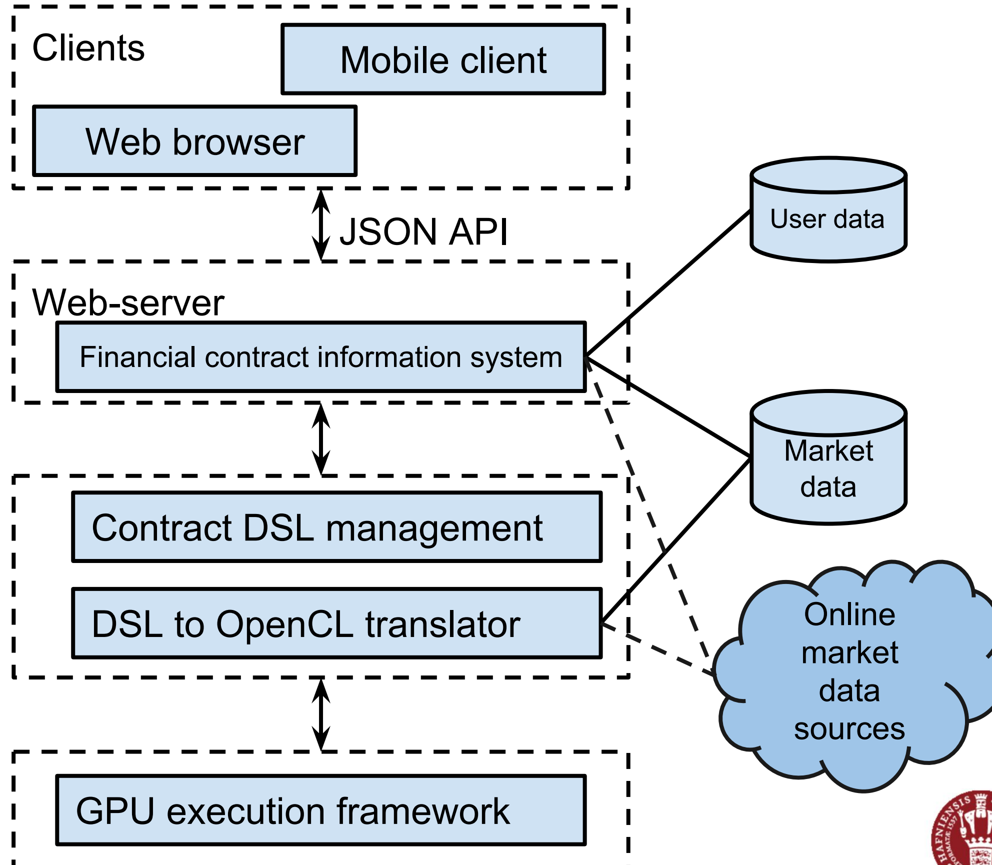
[4] Cosmin E. Oancea, Christian Andreetta, Jost Berthold, Alain Frisch, and Fritz Henglein. **Financial software on GPUs: between Haskell and Fortran**. In *Proceedings of the 1st ACM SIGPLAN workshop on Functional high-performance computing (FHPC '12)*. Copenhagen 2012.

Architecture overview

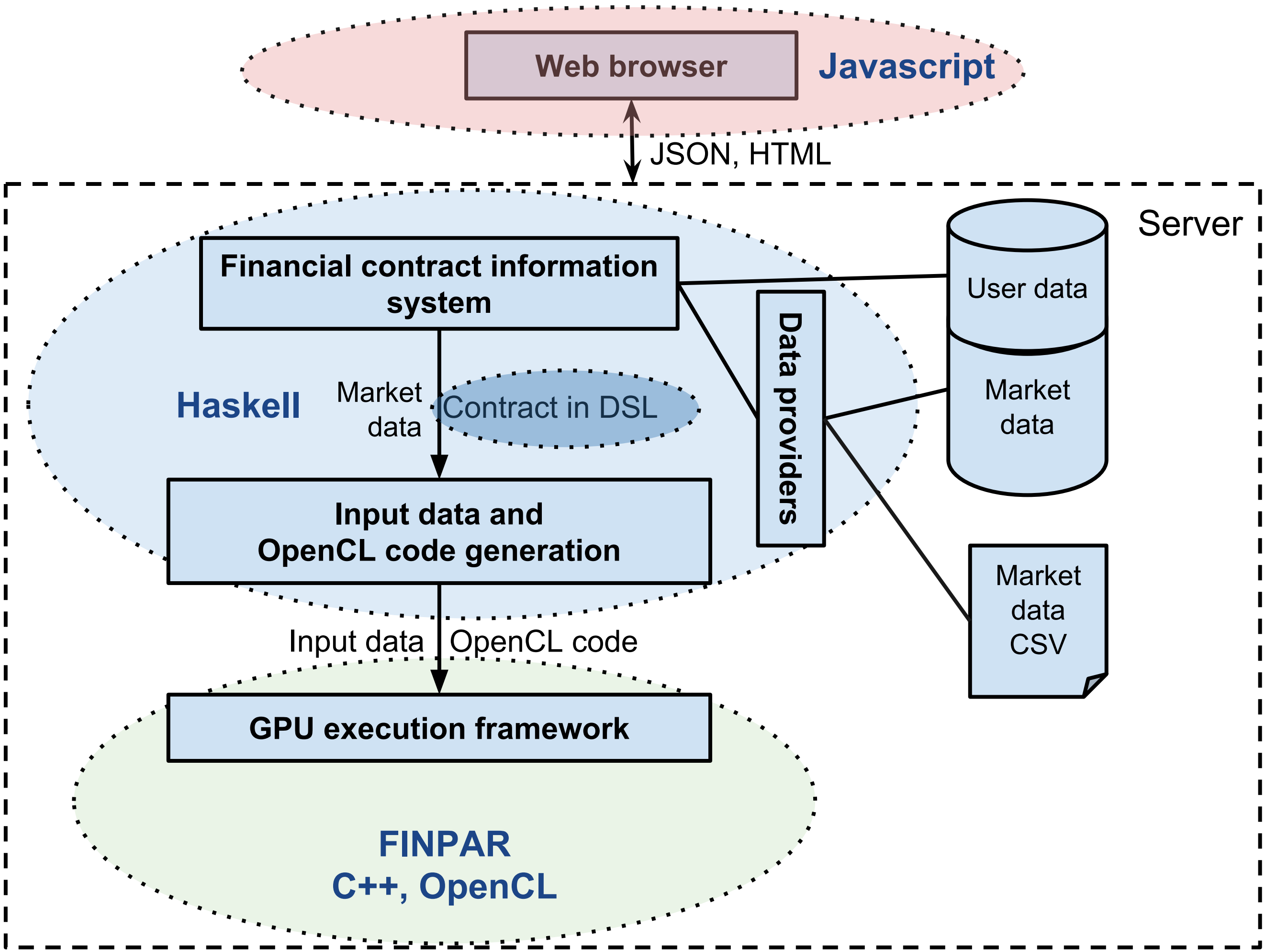
The prototype architecture is **simple**, yet **flexible**.

3 (4) tier architecture:

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



Architecture of current implementation



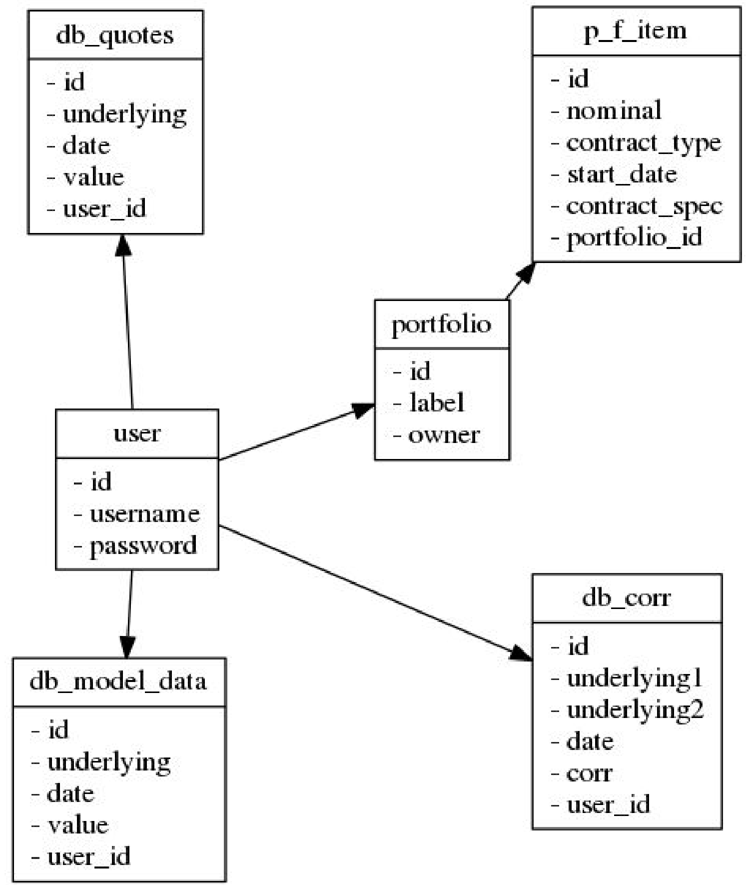
Database

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

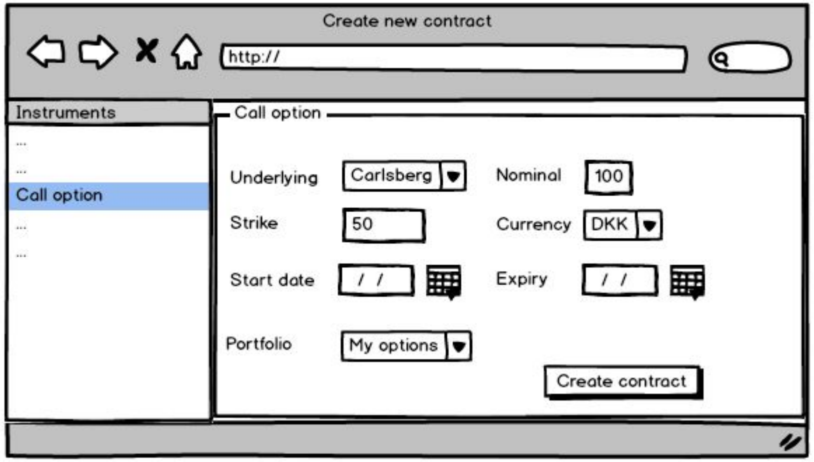
1. An *instrument* maps instrument-specific parameter data to contracts.

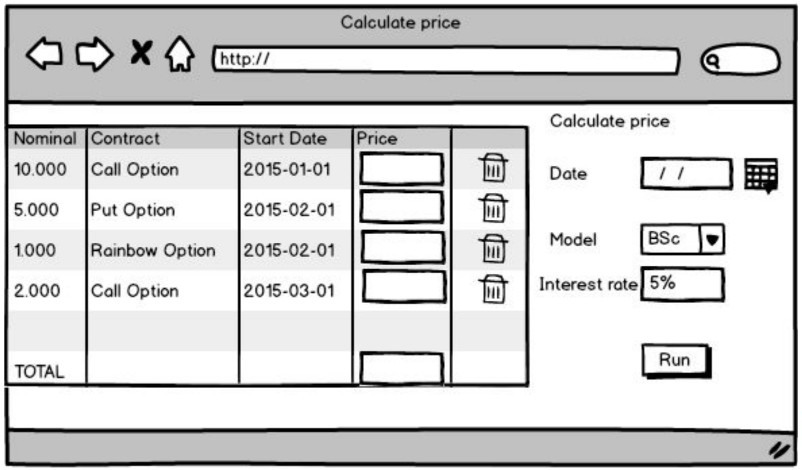
2. Available instruments include, a Call option and a Rainbow option.

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).





Prototype DEMO

prototype.hiperfit.dk

Prototype developed primarily by
HIPERFIT PhD Student Danil Annenkov

Implementation

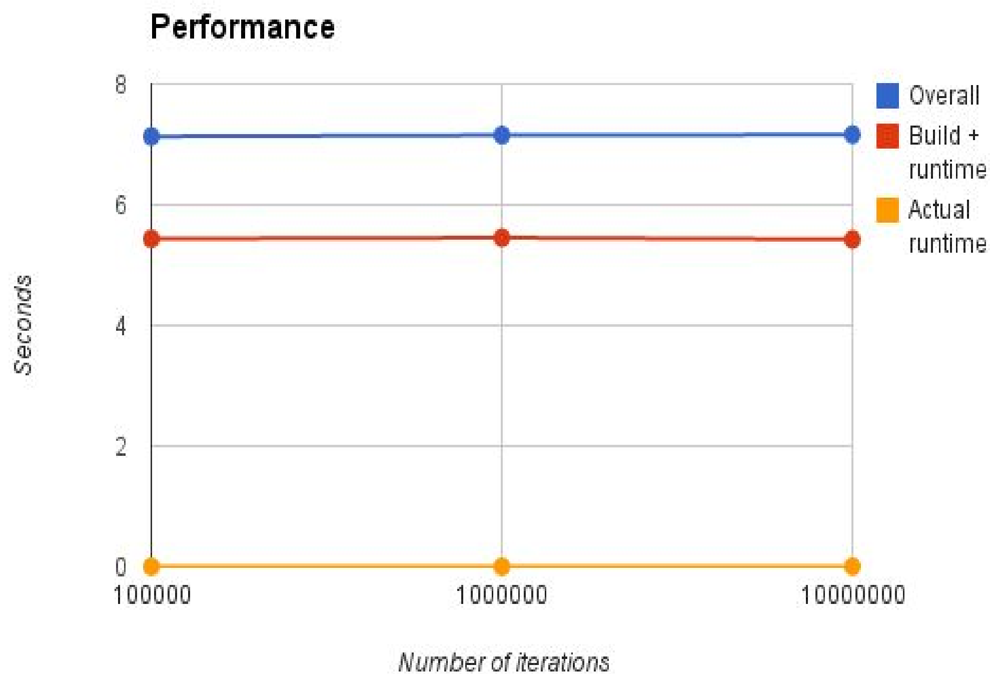
Fork me on GitHub

- 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 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.