Global Systems Science meets Programming Languages and Systems

Presentation at Second Global Systems Science Conference
Stanhope Hotel, Brussels

June 11, 2013

Martin Elsman
Associate Professor, PhD
HIPERFIT Research Center Manager
Department of Computer Science
University of Copenhagen

Global Systems Science – ICT Challenges

The Past

Big data

Data grows!

walking backwards through time

sight

The Future

Predict consequences

Big simulations

Increasing complexity

Complex modeling

Learn from history:

- model calibration
- analytics

You/Society

Decision Making

ICT problems

"*The free lunch is over*":
Halt in CPU clock cycles

Moore's Law still holds some years to come → more transistors

To the rescue:

Parallel computing

Functional and ***declarative*** programming

Exploit ***domain knowledge***

Some Motivation – examples

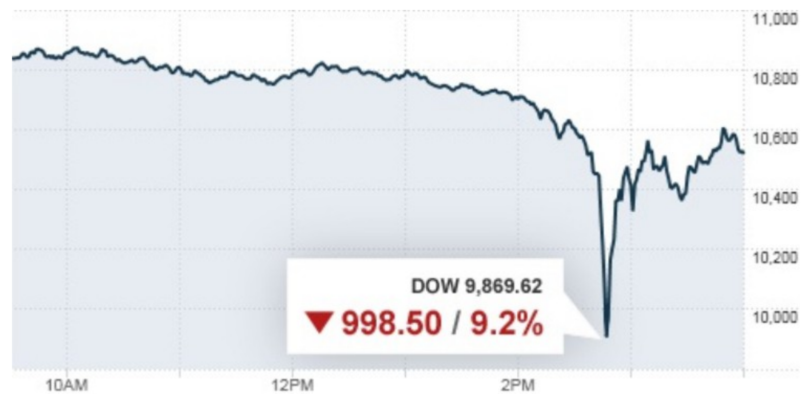
Ex 1: The Credit Crunch…

Worldwide recession in 2008:
US house market collapse.

Rating ignored interdependencies and accumulated failure.

Ex 2: The Flash Crash…

Dow Jones Index on May 6, 2010:



Almost 10% drop within minutes.

Drop almost recovered minutes later.

Systemic effect of algorithmic trading at high volume and frequency.

The 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
C F I R

HIPERFIT: 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.

6 PhD + 3 post-doctoral positions (CS and Mathematics).

Additional funding for collaboration with small/medium-sized businesses.

HIPERFIT Principle: "Less is More"

Transparency
Understand **more** from **shorter** code!

Understand the computation as a mathematical formula with clear semantics.

Productivity
Express **more** with **fewer** lines of code!

Write high-level specifications, not low-level code.

Performance
Compute **more faster**!

Apply domain-specific methods for parallel hardware.

Capture domain-specific parallelism in DSLs.

The Trick
Skip the indirection of imperative software architecture.

Do not build upon sequentialized inherently parallel operations!!

Use Functional Programming Language techniques!

Vision

L

A High-Level, Parallel,
Functional Language

Vision

Financial Contract Specification (DIKU, IMF, Nordea)
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.

Automatic 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.

CVA (IMF, DIKU, Nordea)
Parallelize calculation of exposure to counterparty credit risk.

L

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.

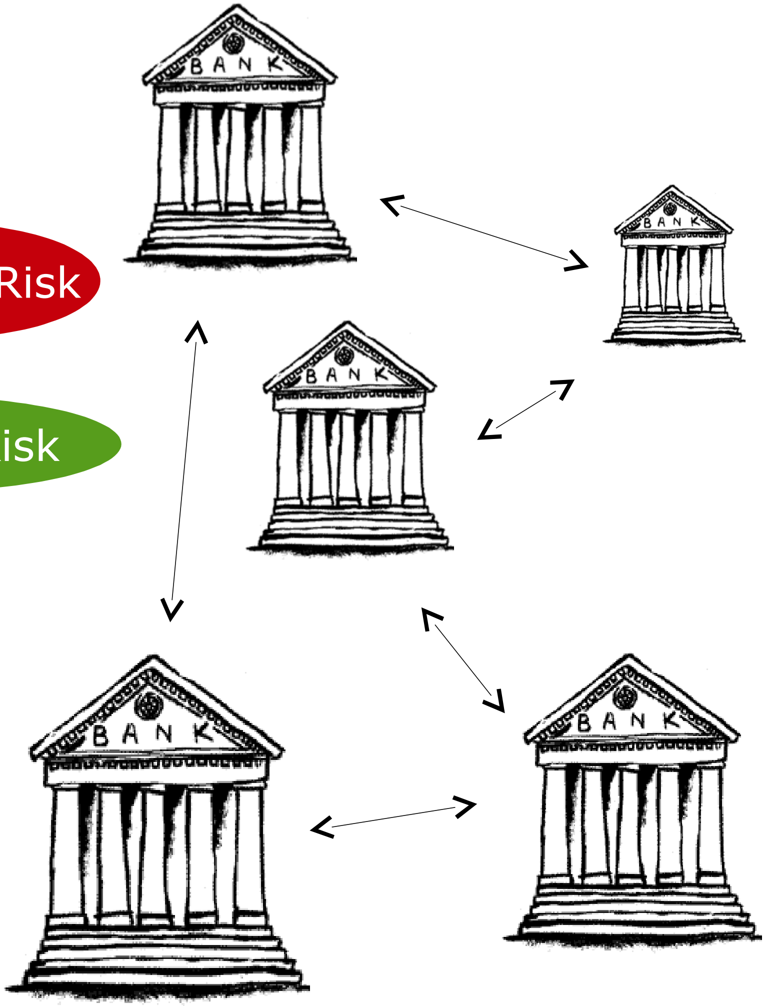
Big Data – Efficient queries (DIKU, SimCorp)
Parallelize big data queries.

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

Project: Financial Contract Specification

Banks (and other financial institutions) use financial contracts for both

- Speculation — Increase Risk
- Insurance (hedging) — Decrease Risk



Many contracts are ”**Over The Counter**” (OTC) contracts, which are negotiated agreements between a bank and another bank (its counter party).

Many Types of Contracts are Traded

- FX Swaps
- Inflation Swaps
- Catastrophy Bonds
- Interest Rate Swaps
- Bermudan Options
- Variance Certificates
- Plain Vanilla Options
- Indexed Linked Bonds
- Rainbow Options
- Chooser Options
- Credit Default Swaps (CDS)
- Himalyan Options
- FX Options
- Forward Rate Agreements
- Callable Range Accrual Notes

How do Banks and Society Keep Track?

Algebraic Properties, simple reasoning

Many Problems

- Financial contracts need management
  fixings, ***decisions***, ***corporate actions***, …
- Banks must **report daily** on their total value of assets
- Banks must **control risk** (counterparty risk, currency risk, …)
- Banks need to know about future cash flows, …

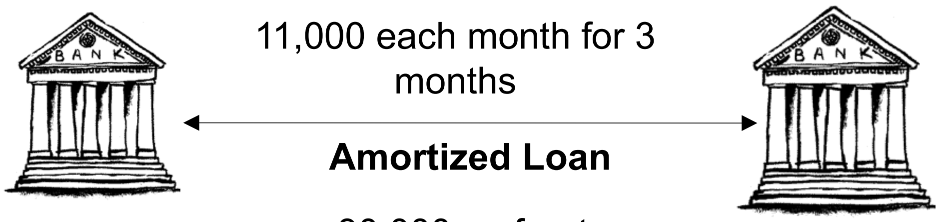
A Solution:

- Specify financial contracts in a **domain specific language**
- Use a **functional programming language** (e.g., ML)

Financial industry has already recognized the value of FP:

- LexiFi (ICFP'00 paper by Peyton-Jones, Eber, Seward)
- SimCorp A/S (uses LexiFi technology)
- Jane Street Capital (focus on electronic trading)
- Societe Generale, Credit Suisse, Standard Chartered
- Contract ”Pay-off” specifications are often functional in style

Example Financial Contracts in an Embedded DSL




```
(* Simple amortized loan *)
val ex1 =
    let val coupon = 11000.0
        val principal = 30000.0
    in All [Give(flow(?"2011-01-01",principal,EUR)),
            flow(?"2011-02-01",coupon,EUR),
            flow(?"2011-03-01",coupon,EUR),
            flow(?"2011-04-01",coupon,EUR)]
    end

(* Cross currency swap *)
val ex2 =
    All [Give(
           All[flow(?"2011-01-01",7000.0,DKK),
               flow(?"2011-02-01",7000.0,DKK),
               flow(?"2011-03-01",7000.0,DKK)]),
         flow(?"2011-01-01",1000.0,EUR),
         flow(?"2011-02-01",1000.0,EUR),
         flow(?"2011-03-01",1000.0,EUR)]
```

Notice: flows in Different currencies

A More Complex Example...

```
(* Call option on "Carlsberg" stock *)
val equity = "Carlsberg"
val maturity = ?"2012-01-01"
val ex4 =
    let val strike = 50.0
        val nominal = 1000.0
        val obs =
            Max(Const 0.0,
                Sub(Underlying(equity,maturity),
                    Const strike))
    in Scale(Const nominal,
             Acquire(maturity,Scale(obs,One EUR)))
    end
```

Meaning: Acquire at maturity the amount (in EUR), calculated as follows (*P* is price of Carlsberg stock at maturity):

nominal * max(0, *P* – *strike*)

What can we do with the contract definitions?

- Report on the **expected future cash flows**
- Perform **management operations**:
  - Advancement (simplify contract when time evolves)
  - Corporate action (stock splits, merges, catastrophic events, …)
  - Perform fixing (simplify contract when an underlying becomes known)
- Report on the **value (price) of a contract**
  - Stochastic models; Monte-Carlo simulation
- Calculate **risk** (Key-Ratios, MC Var, CVA)

Project: Compiling APL

APL is in essence a functional language

APL has arrays as its primary data structure

APL “requires a special keyboard”!

Examples:

```
a ← ι 8          ⍝ array [1..8]
b ← +/ a         ⍝ sum of elements in a
f ← {2+ω×ω}      ⍝ function x^2 + 2
c ← +/ f¨a       ⍝ apply f to all elements of a and sum the elements
```

Reduce

Map

> APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.
>
> Edsger Dijkstra

Compiling APL – An Example

APL Code:

```
diff ← {1↓ω−¯1⌽ω}
signal ← {¯50⌈50⌊50×(diff 0,ω)÷0.01+ω}
+/ signal ι 100000
```

Notice: The APL Compiler has removed all notions of arrays!

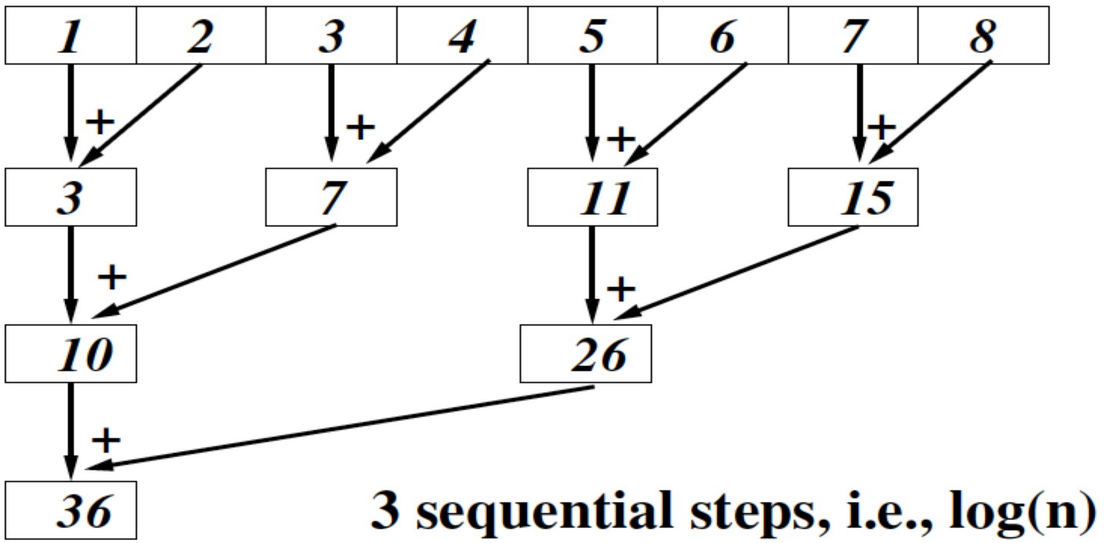
Generated C Code:

```
double kernel(int n14) {
  double d13 = 0.0;
  for (int n82 = 0; n82 < 100000; n82++) {
    d13 = (max(-50.0,min(50.0,(50.0*(i2d(((1+n82)-((n82<1) ? 0 : n82)))/
              (0.01+i2d((1+n82)))))))+d13);
  }
  return d13;
}
```

We Can Do Better!

map is "embarressingly parallel"!

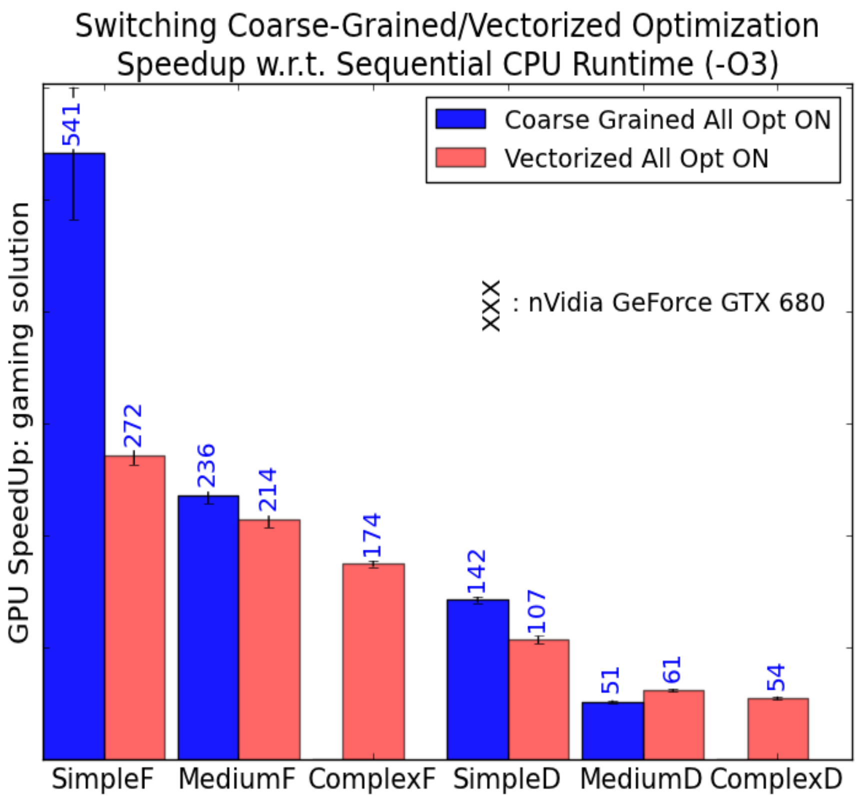
Also **reduce** (+/) can be parallelized:



By making use of **map** and **reduce** and a few other combinators (e.g., **scan**), a high degree of parallelism can be obtained.

Project: Pricing Financial Contracts on GPGPUs

Experiments made by HIPERFIT postdocs C. Oancea and C. Andreetta:



Three kinds of contracts: Simple, Medium, Complex

F/D: Floats/Doubles

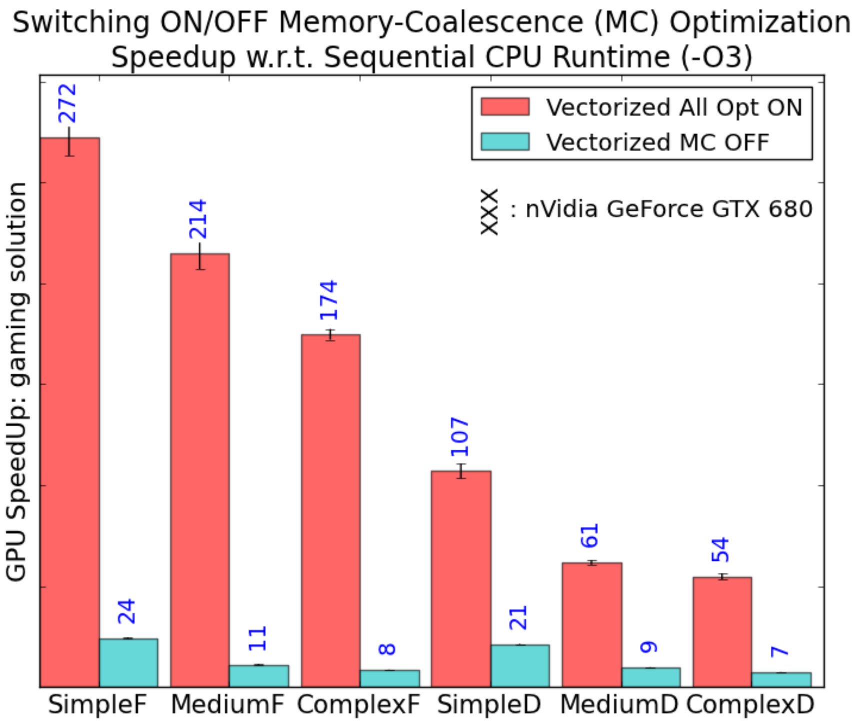
Coarse Grained: One outer map (use of map fusion)

Vectorized: Map distribution

Note: Complete setup with parallel sobol sequence generation, brownian bridge, payoff function...

GPGPUs Requires New Kinds of Optimizations

Accessing global memory in a "coalesced way" may lead to dramatic speedups!



Coalesced: consecutive threads must access consecutive global memory slots...

Often a change of algorithm is needed for ensuring coalescing (e.g., matrix transposition)...

Some Conclusions

Functional programming:

- Is **declarative**: **Focuses on what** instead of how
- Eases **reasoning** (formal as well as informal) and **parallel processing**

Open Question

The modern computation model is **highly parallel**:

- Computation everywhere simultaneously
- Grand challenge: **How to program it**?
- What is a good programming model/**cost-model**?

Quote (Bill Dally, senior vice president of NVIDIA Research)

"Making it easy to program a machine that requires 10 billion threads to use at full capacity is [also] a challenge. ... We need to move toward **higher-level programming models** where the programmer describes the algorithm with all available parallelism and locality exposed, and tools automate much of the process of efficiently mapping and tuning the program to a particular target machine."