Master's thesis defence

Option pricing using functional data-parallel languages

Philip Carlsen Martin Dybdal

Computer Science
University of Copenhagen

22. March 2013

Overview

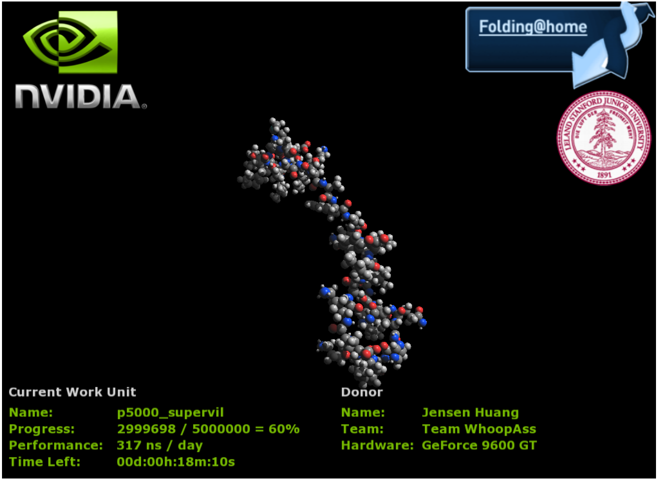
- Data-parallel language survey
  - Comparison through implementation of option pricing algorithms
  - Identification of shortcomings

Overview

- Data-parallel language survey
  - Comparison through implementation of option pricing algorithms
  - Identification of shortcomings
- Proposed extensions
  - Not completely implemented

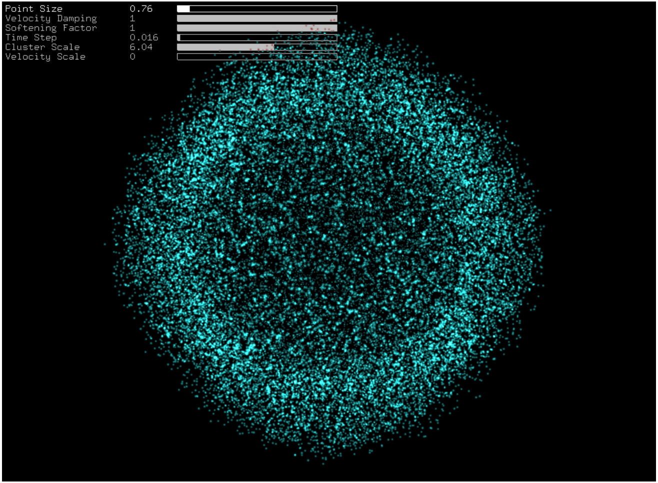
Data-parallel problems: Finance

Data-parallel problems: Bioinformatics



Protein Folding

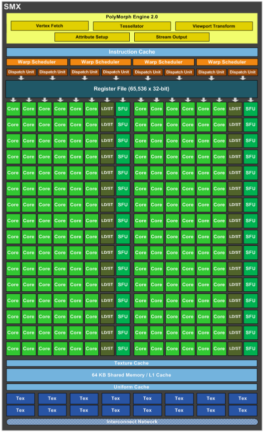
Data-parallel problems: Scientific simulations



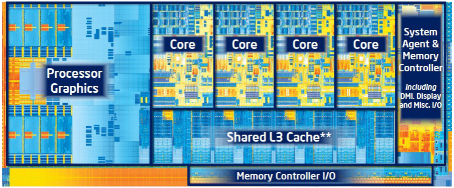
N-body simulation

GPUs vs. CPUs

NVIDIA GK104 GPU Multiprocessor



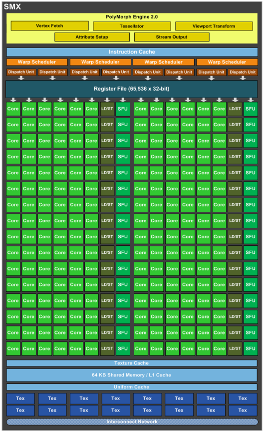
Intel Ivy Bridge CPU



- Computation vs. caching
- High throughput vs. low latency

GPU programming

NVIDIA GK104 GPU
Multiprocessor

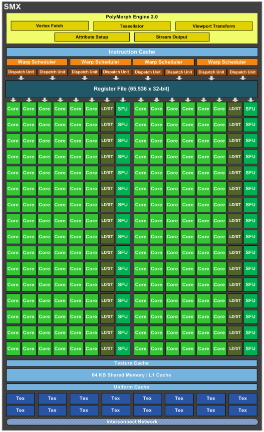


Things to be aware of:

- Memory access coordination

GPU programming

NVIDIA GK104 GPU Multiprocessor

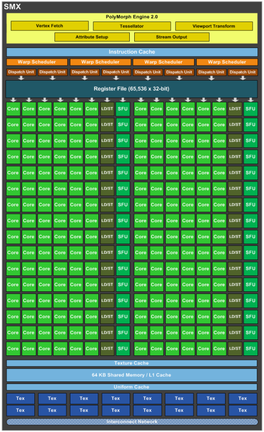


Things to be aware of:

- Memory access coordination
- Avoid memory accesses (perform program fusion)

GPU programming

NVIDIA GK104 GPU Multiprocessor



Things to be aware of:

- Memory access coordination
- Avoid memory accesses (perform program fusion)
- Schedule enough work

GPU programming languages

What we want:

- Avoid manual memory management
- Avoid manual fusion
- Fusion with library code
- Good abstractions
- Hardware independent

Data-parallel programming languages

- Feldspar
- Obsidian (GPU)
- Data-parallel Haskell
- Copperhead (GPU)
- Nikola (GPU)
- Accelerate (GPU)
- Repa
- Data.Vector

Later we discovered: Theano, Bohrium, CnC-CUDA and R+GPU

Data-parallel programming languages

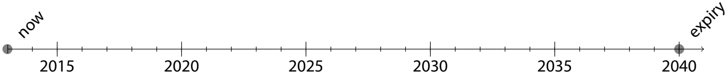
- Feldspar
- Obsidian (GPU)
- Data-parallel Haskell
- Copperhead (GPU)
- **Nikola (GPU)**
- **Accelerate (GPU)**
- **Repa**
- **Data.Vector**

Later we discovered: Theano, Bohrium, CnC-CUDA and R+GPU

Options

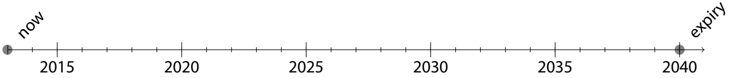
See blackboard

Option styles

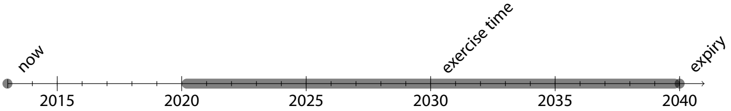


European option

Option styles

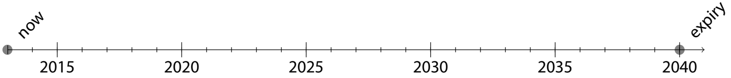


European option

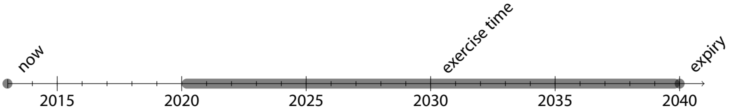


American option

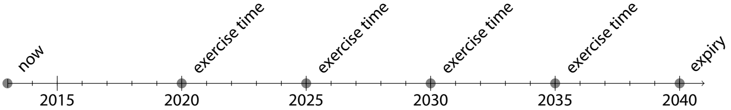
Option styles



European option

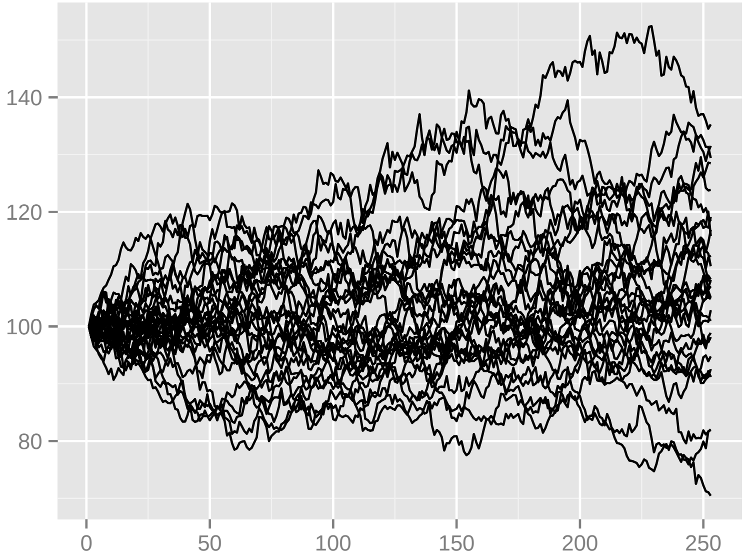


American option

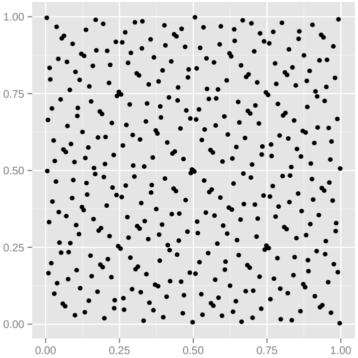


Bermudan option

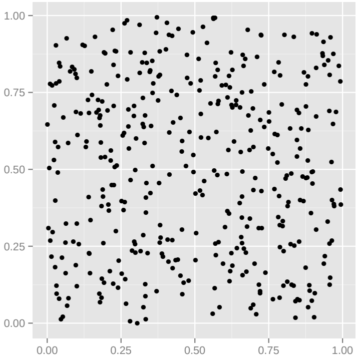
Option pricing: Least-Squares Monte Carlo



Sobol-sequence generation



2D Sobol-sequence



2D Mersenne Twister Sequence

Sobol-sequence generation

What we want:

$$\mathsf{SobolInductive}\ v\ i \;=\; \textbf{reduce}\ (\oplus)\ 0\ (\textbf{zipWith}\ (\cdot)\ v\ (\mathsf{ToBitVector}\ i))$$

$$\left(\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{32} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \right) \overset{\oplus}{\downarrow} = x_3$$

Sobol-sequence generation

What we want:

$$\mathsf{SobolInductive}\; v\; i \;=\; \textbf{reduce}\,(\oplus)\,0\,(\textbf{zipWith}\,(\cdot)\,v\,(\mathsf{ToBitVector}\;i))$$

$$\left(\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{32} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix}\right) \overset{\oplus}{\downarrow} = x_3$$

$$\mathsf{SobolSequence1D}\; m\; v \;=\; \textbf{parmap}\,(\mathsf{SobolInductive}\; v)\,[1..m]$$

$$\left(\begin{pmatrix} v_1 & v_1 & \cdots \\ v_2 & v_2 & \cdots \\ \vdots & \vdots & \\ v_{32} & v_{32} & \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 & \cdots \\ 0 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \end{pmatrix}\right) \overset{\oplus}{\downarrow} = \begin{pmatrix} x_1 & x_2 & x_3 & \cdots \end{pmatrix}$$

Sobol-sequence generation

What we want:

$$\mathsf{SobolInductive}\; v\; i \;=\; \textbf{reduce}\; (\oplus)\; 0\; (\textbf{zipWith}\; (\cdot)\; v\; (\mathsf{ToBitVector}\; i))$$

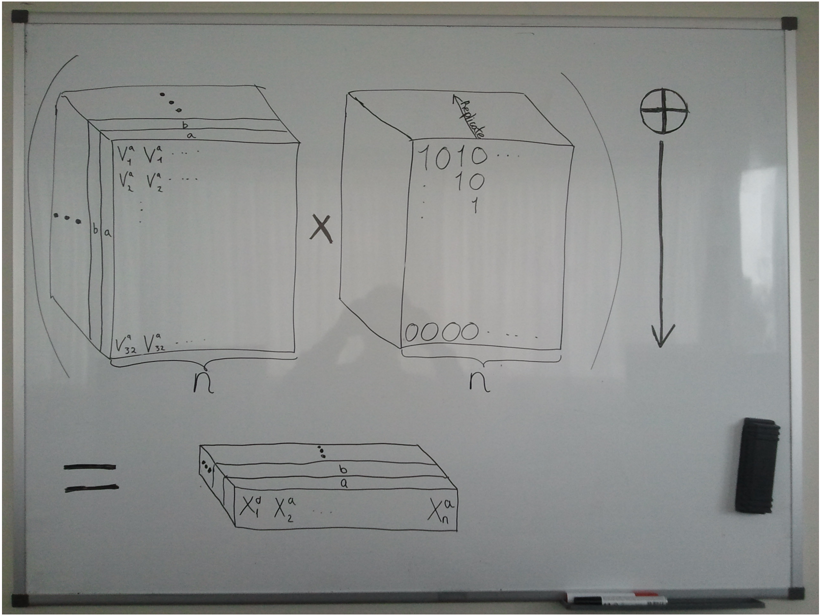
$$\left(\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{32} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \right) \overset{\oplus}{\downarrow} = x_3$$

$$\mathsf{SobolSequence1D}\; m\; v \;=\; \textbf{parmap}\; (\mathsf{SobolInductive}\; v)\; [1..m]$$

$$\left(\begin{pmatrix} v_1 & v_1 & \cdots \\ v_2 & v_2 & \cdots \\ \vdots & \vdots & \\ v_{32} & v_{32} & \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 & \cdots \\ 0 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \end{pmatrix} \right) \overset{\oplus}{\downarrow} = \begin{pmatrix} x_1 & x_2 & x_3 & \cdots \end{pmatrix}$$

$$\mathsf{SobolSequenceND}\; m\; vs \;=\; \textbf{parmap}\; (\mathsf{SobolSequence\text{-}1D}\; m)\; vs$$

Sobol-sequence generation



Implementation observations

- Rely on fusion to avoid memory overhead

Implementation observations

- Rely on fusion to avoid memory overhead
- Incompositional

Implementation observations

- Rely on fusion to avoid memory overhead
- Incompositional
- Thus, library development seems infeasible

Option pricing: Binomial method

$$S(t+\Delta t) = \begin{cases} S(t)u & \text{with probability } q \\ S(t)d & \text{with probability } 1-q \end{cases}$$

$$\begin{tikzcd}[column sep=small]
S(t_0) = & & & & S(t_0) \arrow[dl] \arrow[dr] & & & \\
S(t_1) = & & & dS(t_0) \arrow[dl] \arrow[dr] & & uS(t_0) \arrow[dl] \arrow[dr] & & \\
S(t_2) = & & d^2S(t_0) \arrow[dl] \arrow[dr] & & S(t_0) \arrow[dl] \arrow[dr] & & u^2S(t_0) \arrow[dl] \arrow[dr] & \\
S(t_3) = & d^3S(t_0) & & dS(t_0) & & uS(t_0) & & u^3S(t_0)
\end{tikzcd}$$

Binomial lattice for three periods ($u \cdot d = 1$)

Implementation observations

Accelerate and Nikola:

- Synchronization has to happen across all blocks.
- We cannot force the use of just a single block.
- We cannot reuse already allocated memory.
- Writing portfolio pricer requires irregular arrays.

Implementation observations

Accelerate and Nikola:

- Synchronization has to happen across all blocks.
- We cannot force the use of just a single block.
- We cannot reuse already allocated memory.
- Writing portfolio pricer requires irregular arrays.

BinomialPortfolio = **parmap** (λopt. **foldl** (prev opt) (final opt) [n..1])

Nested array operations

The same operations can execute in several ways:

- **mapS** - sequential loop on the device
- **mapP** - independent parallel threads
- **mapSeqPar** - execute the kernels in strict sequence

Nested array operations

The same operations can execute in several ways:

- **mapS** - sequential loop on the device
- **mapP** - independent parallel threads
- **mapSeqPar** - execute the kernels in strict sequence

We can now write:

$$\text{SobolInductive}\, v\, i \;=\; \textbf{reduceS}\; (\oplus)\; 0\; (\textbf{zipWithS}\; (\cdot)\; v\; (\text{ToBitVector}\; i))$$

$$\text{SobolSequence1D}\, m\, v \;=\; \textbf{mapP}\; (\text{SobolInductive}\; v)\; [1..m]$$

$$\text{SobolSequenceND}\, m\, vs \;=\; \textbf{mapSeqPar}\; (\text{SobolSequence-1D}\; m)\; vs$$

Sequential

- This selection should be automated.
- We suggest using a marking construct

Sequential

- This selection should be automated.
- We suggest using a marking construct

$$\mathsf{SobolInductive}\,v\,i \;=\; \textbf{reduce}\;(\oplus)\;0\;(\textbf{zipWith}\;(\cdot)\;v\;(\mathsf{ToBitVector}\;i))$$

$$\mathsf{SobolSequence1D}\,m\,v \;=\; \textbf{map}\;(\underline{\textbf{sequential}}(\mathsf{SobolInductive}\,v))\;[1..m]$$

$$\mathsf{SobolSequenceND}\,m\,vs \;=\; \textbf{map}\;(\mathsf{SobolSequence\text{-}1D}\,m)\;vs$$

Sequential

- This selection should be automated.
- We suggest using a marking construct

$$\text{SobolInductive}\, v\, i \;=\; \textbf{reduce}\, (\oplus)\, 0\, (\textbf{zipWith}\, (\cdot)\, v\, (\text{ToBitVector}\, i))$$

$$\text{SobolSequence1D}\, m\, v \;=\; \textbf{map}\, (\underline{\textbf{sequential}}(\text{SobolInductive}\, v))\, [1..m]$$

$$\text{SobolSequenceND}\, m\, vs \;=\; \textbf{map}\, (\text{SobolSequence-1D}\, m)\, vs$$

$$\Downarrow$$

$$\text{SobolInductive}\, v\, i \;=\; \textbf{reduceS}\, (\oplus)\, 0\, (\textbf{zipWithS}\, (\cdot)\, v\, (\text{ToBitVector}\, i))$$

$$\text{SobolSequence1D}\, m\, v \;=\; \textbf{mapP}\, (\text{SobolInductive}\, v)\, [1..m]$$

$$\text{SobolSequenceND}\, m\, vs \;=\; \textbf{mapSeqPar}\, (\text{SobolSequence-1D}\, m)\, vs$$

Block level synchronization

Several additional levels, depending on the architecture:

- **mapB** - independent parallel threads using only in a single block
- **mapSeqParB** - sequential map with device synchronization between each map.

BinPortfolio = **map** (**foldl** (**map** ...) (**map** ...) [n..1])
BinPortfolio = **mapP** (**foldlSeqParB** (**mapB** ...) (**mapB** ...) [n..1])

Ordering

- **S** can occur within **P** or **B** operations
- **B** can occur within **SeqPar**, **P** or **SeqParB** operations
- **SeqParB** can occur within **SeqPar**, **P** or **SeqParB** operations
- **P** can occur within **SeqPar** operations
- **SeqPar** can occur within **SeqPar**

Sobol-sequences take #2

Recursive formulation: $x_{i+1} = \; x_i \oplus v_{\mathrm{lsb}(i)}$

unfold :: (Int $\rightarrow$ a $\rightarrow$ a) $\rightarrow$ Int $\rightarrow$ a $\rightarrow$ [a]

Sobol-sequences take #2

Recursive formulation: $x_{i+1} = x_i \oplus v_{\mathrm{lsb}(i)}$

unfold :: (Int $\rightarrow$ a $\rightarrow$ a) $\rightarrow$ Int $\rightarrow$ a $\rightarrow$ [a]

unfold :: (Int $\rightarrow$ a $\rightarrow$ a) $\rightarrow$ Int $\rightarrow$ [a] $\rightarrow$ [[a]]

Future work

- Implementation and theoretical development
  - Will this work out in practice?
  - How should memory be managed?
  - How are irregular arrays handled?
  - Can we do this in the type system?
  - **mapP**'s and **foldlS**'s as a target language in other situations

Future work

- Implementation and theoretical development
  - Will this work out in practice?
  - How should memory be managed?
  - How are irregular arrays handled?
  - Can we do this in the type system?
  - **mapP**'s and **foldlS**'s as a target language in other situations
- Extend the survey
  - Test additional languages
  - Implement complete LSM
  - Evaluate implementations of additional algorithms