General Purpose GPU Computing @ Programming Languages & Systems

Manuel M. T. Chakravarty

University of New South Wales, Sydney

PLS

Our secret weapon: Functional Programming

- Haskell, ML, Lisp, etc.
- Not so secret anymore. . . the next big thing?

Programming Languages & Systems @ CSE

Our secret weapon: Functional Programming

- Haskell, ML, Lisp, etc.
- Not so secret anymore. . . the next big thing?

Specifically, Haskell

- Mature & widely used
- Excellent, fully featured & highly optimising implementation
- Lots of libraries and a great community
- `http://haskell.org/`

PLS

Programming Languages & Systems @ CSE

Our secret weapon: Functional Programming

- Haskell, ML, Lisp, etc.
- Not so secret anymore. . . the next big thing?

Specifically, Haskell

- Mature & widely used
- Excellent, fully featured & highly optimising implementation
- Lots of libraries and a great community
- `http://haskell.org/`

What we do

- Improving programming language designs and implementations
- Solve interesting problems with cutting-edge language technology

PLS

Parallelism and Functional Programming

Functional programming and parallel programming are fundamentally linked!

Parallelism and Functional Programming

Functional programming and parallel programming are fundamentally linked!

Parallel programming

- Central problem: unpredictable execution order of parallel computations
- Why is that a problem?

Parallelism and Functional Programming

Functional programming and parallel programming are fundamentally linked!

Parallel programming

- Central problem: unpredictable execution order of parallel computations
- Why is that a problem? Side effects!

Parallelism and Functional Programming

Functional programming and parallel programming are fundamentally linked!

Parallel programming

- Central problem: unpredictable execution order of parallel computations
- Why is that a problem? Side effects!

Functional programming

- Controls side effects — `map func [d1, ..., dn]`
- Always safe to execute `(func d1)` to `(func dn)` in parallel

Parallelism and Functional Programming

Functional programming and parallel programming are fundamentally linked!

Parallel programming

- Central problem: unpredictable execution order of parallel computations
- Why is that a problem? Side effects!

Functional programming

- Controls side effects — `map func [d1, ..., dn]`
- Always safe to execute `(func d1)` to `(func dn)` in parallel

CUDA

- Kernel functions must be externally side-effect free
- `kernel<<< n, m, r >>>(...)` is a parallel map

High-Level Programming Model for GPUs

Embedding GPU computations in Haskell

- Embedded domain-specific language for dense array computations — working on irregular, nested computations
- Aggregate array operations are compiled to CUDA code
- Automatic, type-based separation of device and CPU code
- No ad-hoc constraints on control flow constructs

PLS

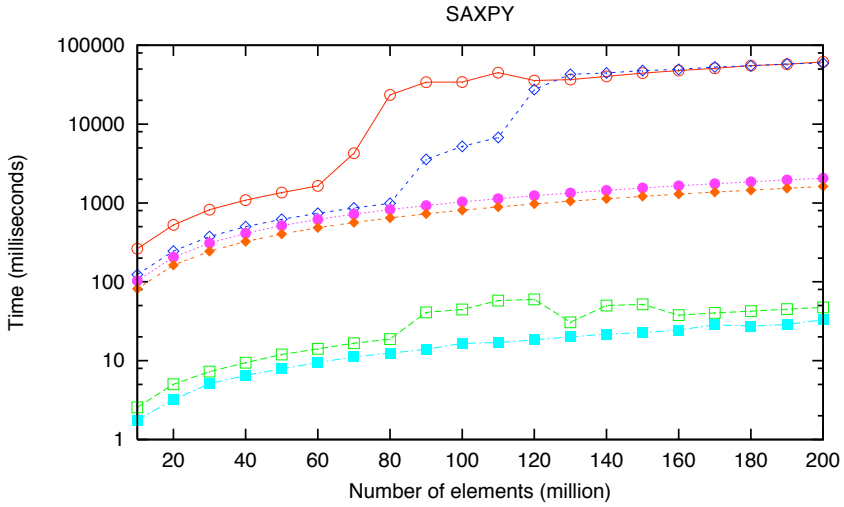
High-Level Programming Model for GPUs

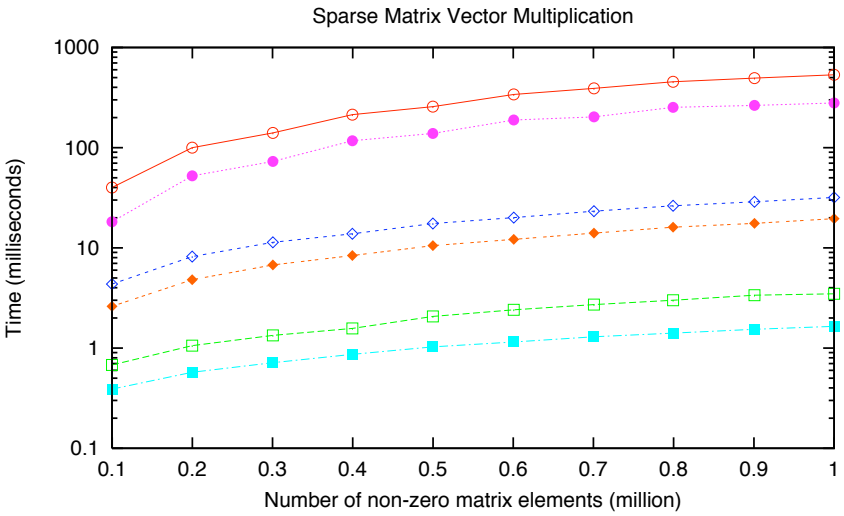
Embedding GPU computations in Haskell

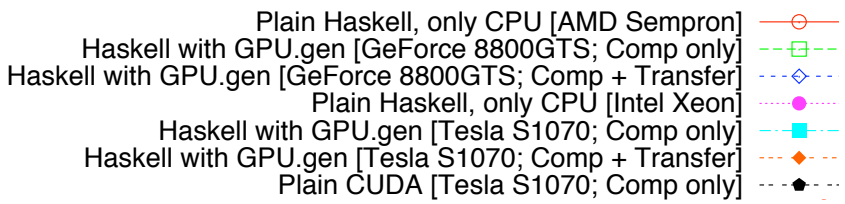
- Embedded domain-specific language for dense array computations — working on irregular, nested computations
- Aggregate array operations are compiled to CUDA code
- Automatic, type-based separation of device and CPU code
- No ad-hoc constraints on control flow constructs

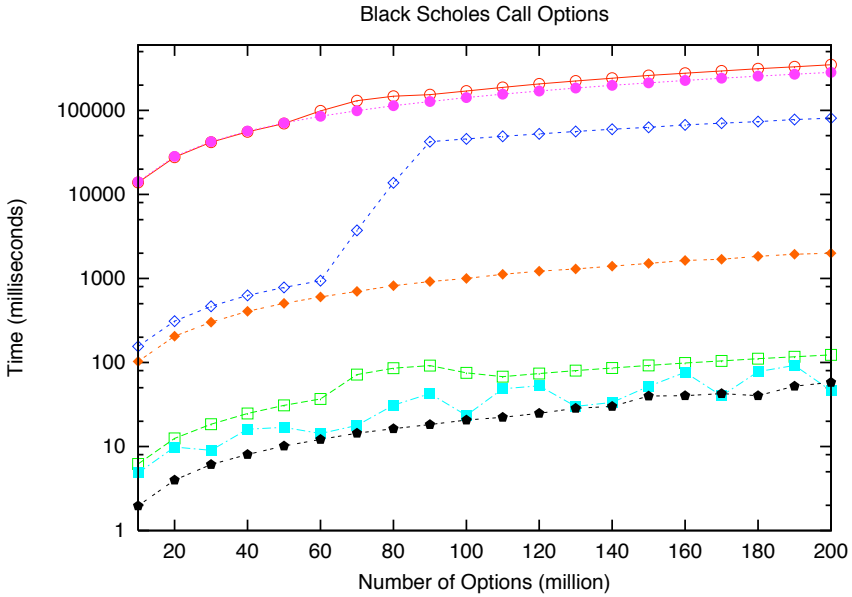
```
type Vector = Array DIM1 Float

saxpy :: GPU.Exp Float → Vector → Vector → Vector
saxpy alpha xs ys
  = GPU.run $ do
      xs' ← use xs
      ys' ← use ys
      GPU.zipWith (λx y → alpha*x + y) xs' ys'
```


SAXPY
Time (milliseconds)
Number of elements (million)
100000
10000
1000
100
10
1
20
40
60
80
100
120
140
160
180
200


Sparse Matrix Vector Multiplication
Time (milliseconds)
Number of non-zero matrix elements (million)
1000
100
10
1
0.1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

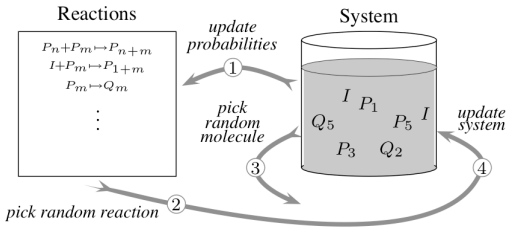

Plain Haskell, only CPU [AMD Sempron]
Haskell with GPU.gen [GeForce 8800GTS; Comp only]
Haskell with GPU.gen [GeForce 8800GTS; Comp + Transfer]
Plain Haskell, only CPU [Intel Xeon]
Haskell with GPU.gen [Tesla S1070; Comp only]
Haskell with GPU.gen [Tesla S1070; Comp + Transfer]
Plain CUDA [Tesla S1070; Comp only]


Black Scholes Call Options
Time (milliseconds)
Number of Options (million)
100000
10000
1000
100
10
1
20
40
60
80
100
120
140
160
180
200

Two Applications

Polymerisation kinetics

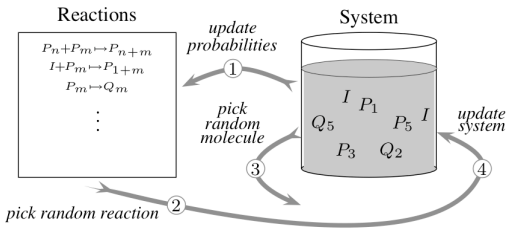
- Collaborated with CAMD @ UNSW
- World's fastest Monte-Carlo simulator
- Runs on multicore CPUs and clusters
- Specialising simulator generator in Haskell generates highly optimised C



Two Applications

Polymerisation kinetics

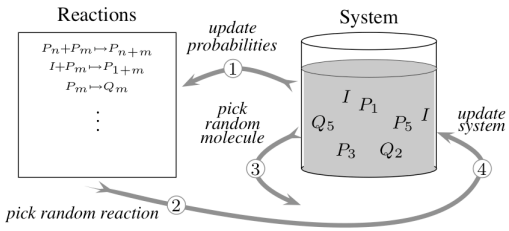
- Collaborated with CAMD @ UNSW
- World's fastest Monte-Carlo simulator
- Runs on multicore CPUs and clusters
- Specialising simulator generator in Haskell generates highly optimised C
- Currently re-targeting to CUDA!



Two Applications

Polymerisation kinetics

- Collaborated with CAMD @ UNSW
- World's fastest Monte-Carlo simulator
- Runs on multicore CPUs and clusters
- Specialising simulator generator in Haskell generates highly optimised C
- Currently re-targeting to CUDA!



Mass spectrometry

- In collaboration with Jason Wong of Medical Sciences
- Identification of proteins in complex proteomics samples
- Started the development of new dataparallel algorithms
- Targeting CUDA

PLS