Parallelism

Gabriele Keller
University of New South Wales

Programming Languages Mentoring Workshop 2015

undergrad

Karslrul

PhD

Univ[illegible] of T[illegible] tkuba, Japan

Credit Suisse

New York

:w South

University of New South Wales

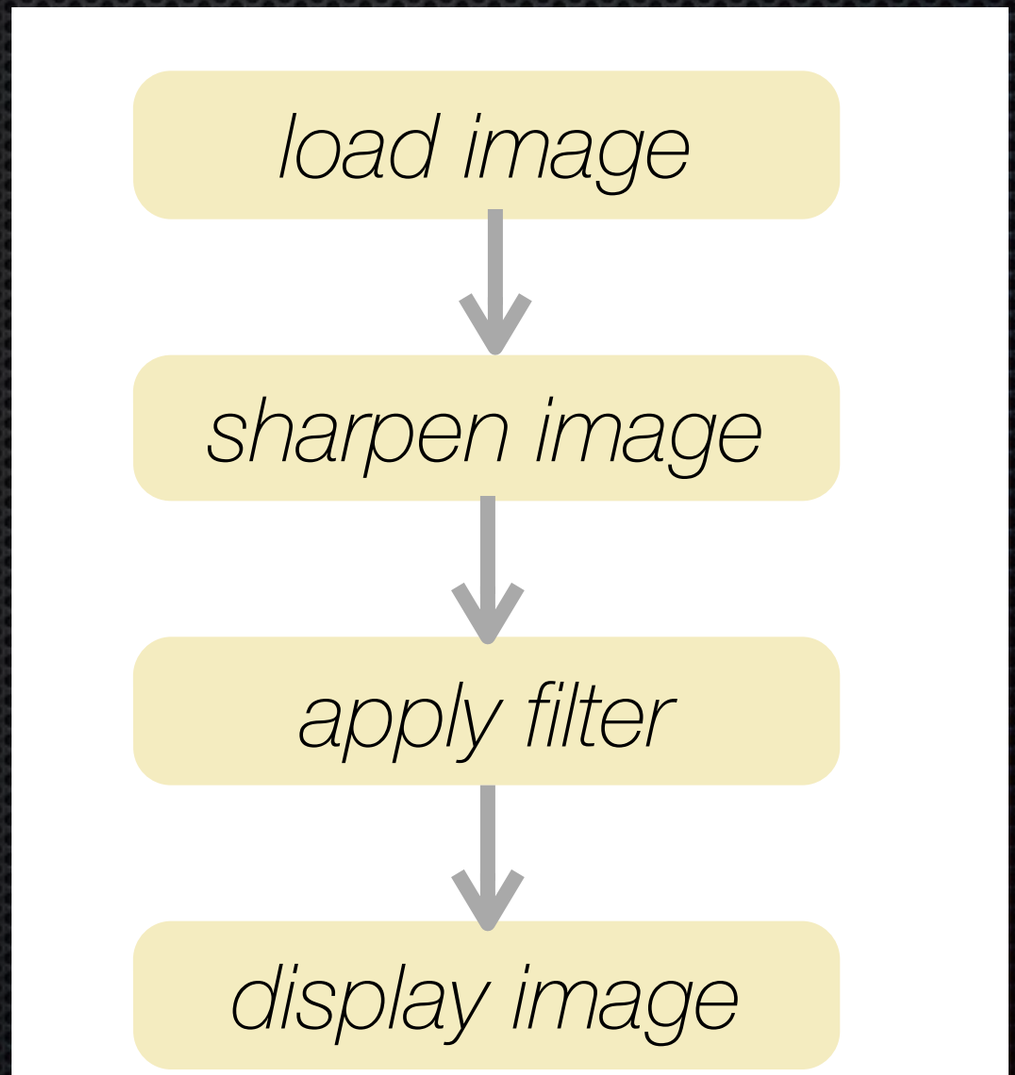
Sydney

What is parallelism?

simultaneous execution of multiple instructions to reduce overall running time

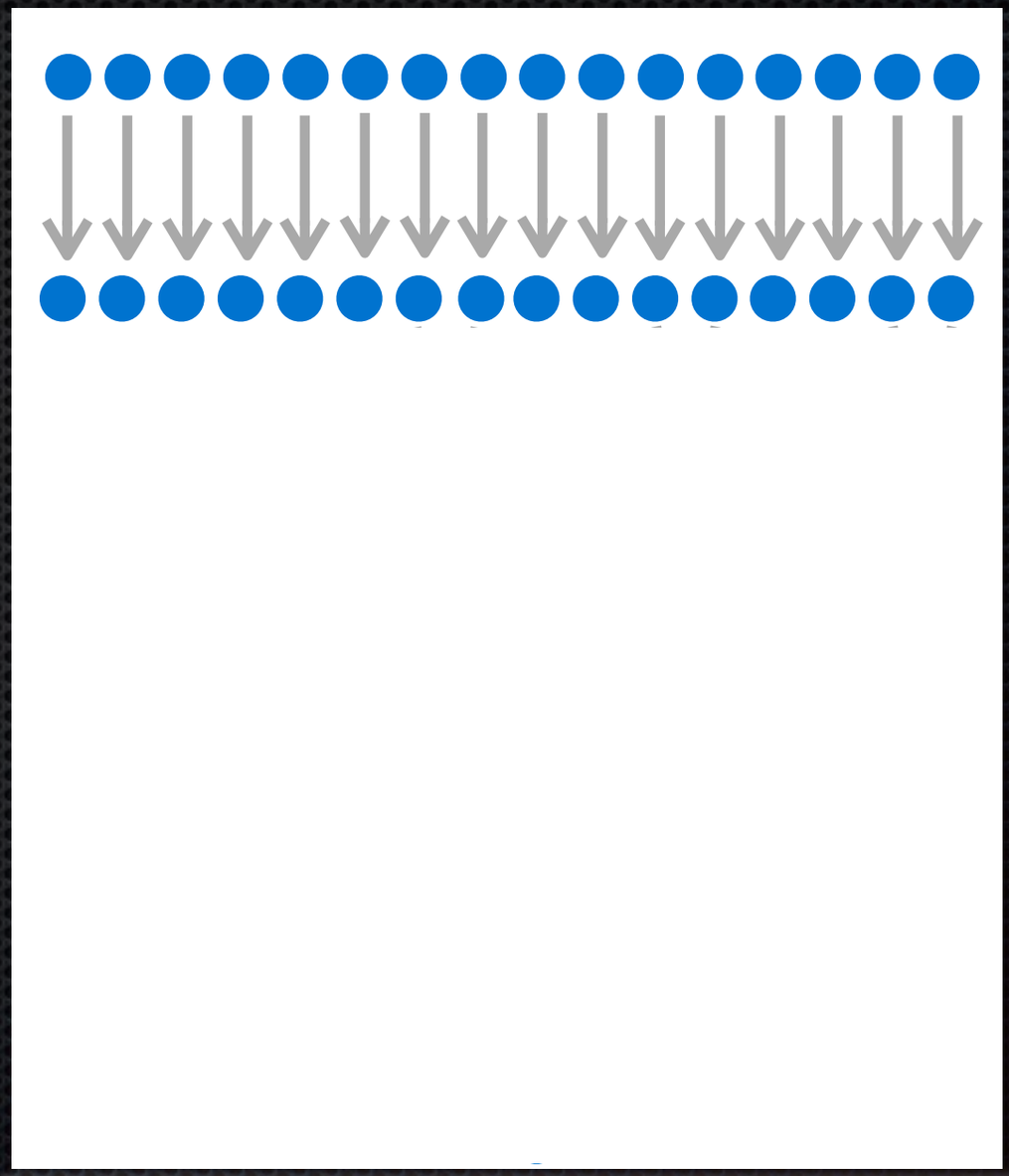
Control/Task Parallelism

- different cooperating tasks are run simultaneously
- close to concurrency wrt to expressiveness, but different aim



Data Parallelism

- same set of instructions applied simultaneously to a collection of data
- regular data parallelism
- irregular/nested data parallelism



Why care about parallel programming?

venerable supercomputer

multicore CPU

http://en.wikipedia.org/wiki/File:Intel_CPU_Core_i7_2600K_Sandy_Bridge_top.jpg

http://en.wikipedia.org/wiki/File:GeFo

multicore GPU

Parallel hardware is now the rule, not the exception

but what about the software?

research on parallelism is more relevant than ever

but maybe more importantly

because there are lots of interesting problems waiting to be solved!

In an ideal world....

sequential program

efficient parallel executable

designing scalable parallel algorithms is difficult!

impossible in general to derive an efficient parallel algorithm automatically from a sequential algorithm

implementation of parallel algorithms is hard

race conditions
deadlocks
heisenbugs

We can make this task a lot easier!

Parallelism and Functional Programming are a great fit!

no shared mutable state
&
only controlled side effects

eliminates the most common source of bugs for parallel and concurrent programs

enables aggressive optimisations

evaluation order up to the compiler

```
treeSum :: Tree -> Int
treeSum Leaf
  = 0
treeSum (Node n t1 t2)
  = s1 + s2 + n
  where
    s1 = treeSum t1
    s2 = treeSum t2
```

collection oriented operations and higher-order functions can expose parallel structures

```
sum = 0;
for (i = 0; i < n; i++) {
  sum += f (a[i]);
}
return sum;
```

```
foldl 0 (+) $ map f a
```

functional languages are great host languages for embedded DSLs

Research Questions

load balancing

language design

applications

parallel algorithms

eduling

and profiling

optimisations

irregular data parallelism

different parallel architectures

domain specific languages

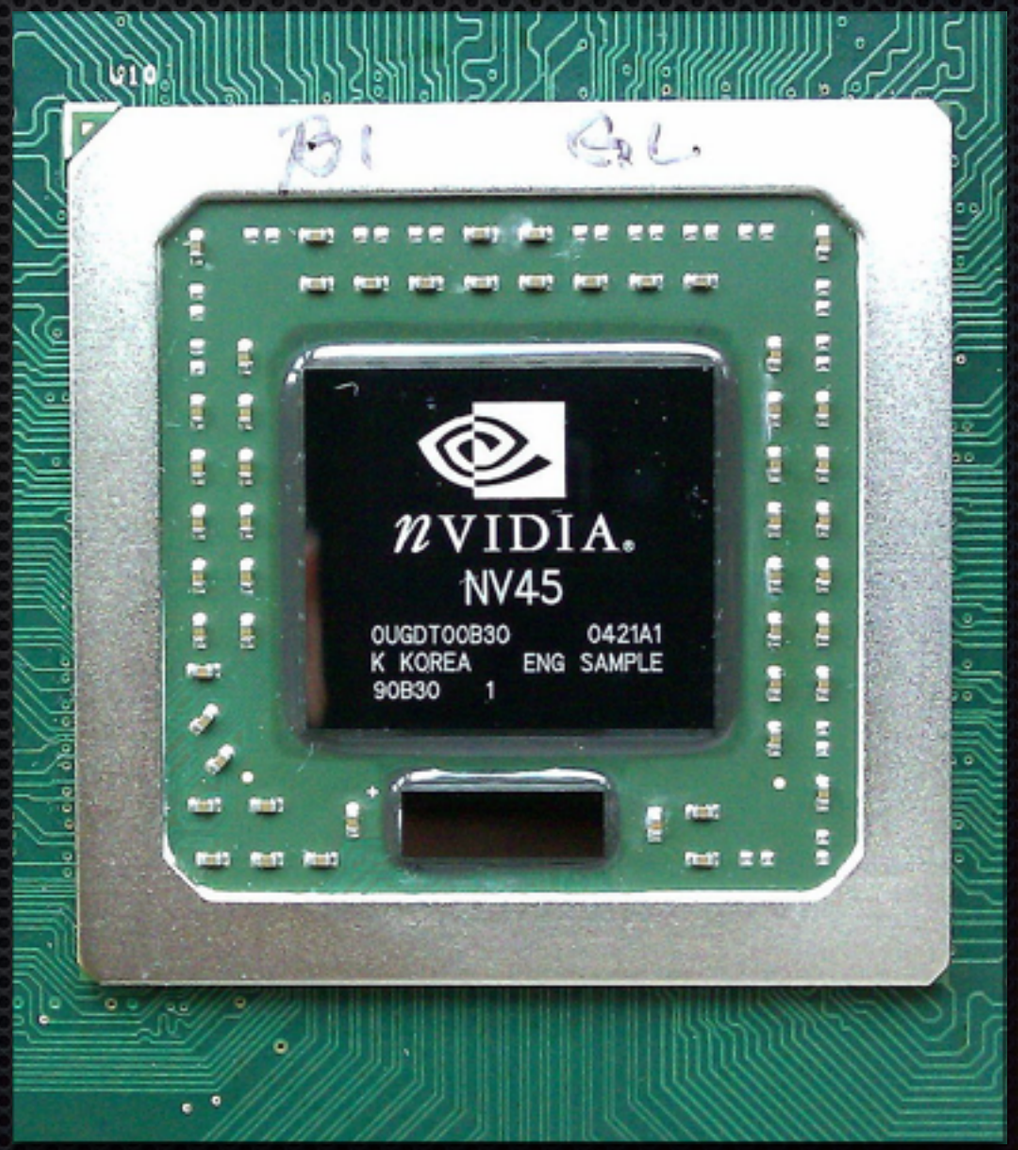
Workshop on Functional High-Performance Computing 2015 (FHPC)

“Regularizing the irregular”

Keynote by Milind Kulkarni

GPGPU

- General Purpose computing on graphics processors
- GPUs are highly cost and energy efficient for parallel computing
- notoriously difficult to program



“Generating Performance Portable Code using Rewrite Rules: From High-level Functional Expressions to High-Performance OpenCL Code” (ICFP)

Michel Steuwer, Christian Fensch, Sam Lindley, Christophe Dubach

- array primitives (fold, map, etc) express parallel computations
- low-level functional OpenCL primitives represent the OpenCL programming model
- a core dependently-typed calculus and denotational semantics;
- rewrite rules express algorithmic and optimization choices to compile to OpenCL, proofs of soundness of these rules

"Meta-Programming and Auto-Tuning in the Search for High Performance GPU Code"

Michael Vollmer Bo Joel Svensson Eric Holk Ryan Newton

- Obsidian, a Haskell EDSL to generate GPU (CUDA) code
- Framework for auto-tuning search in Haskell to optimise CUDA kernels
- Abstraction of auto-tuning searches as applicative functors

"Functional Array Streams"

Frederik M Madsen, Robert Clifton-Everest, Manuel Chakravarty, Gabriele Keller

- Accelerate is a domain specific language for fast array computations embedded in Haskell
- Large data sets are a problem for GPUs, as they only have limited amount of main memory
- Add concepts of streams to Accelerate to express chunked computations

```
dotp::
 Acc (Vector Float) ->
 Acc (Vector Float) ->
 Acc (Scalar Float)

dotP xs ys
 = fold (+) 0 $
   zipWith (*) xs ys
```

"Converting Data Parallelism to Task Parallelism by rewrites"

Bo Joel Svensson, Michael Vollmer, Eric Holk, Trevor L. McDonell, Ryan R. Newton

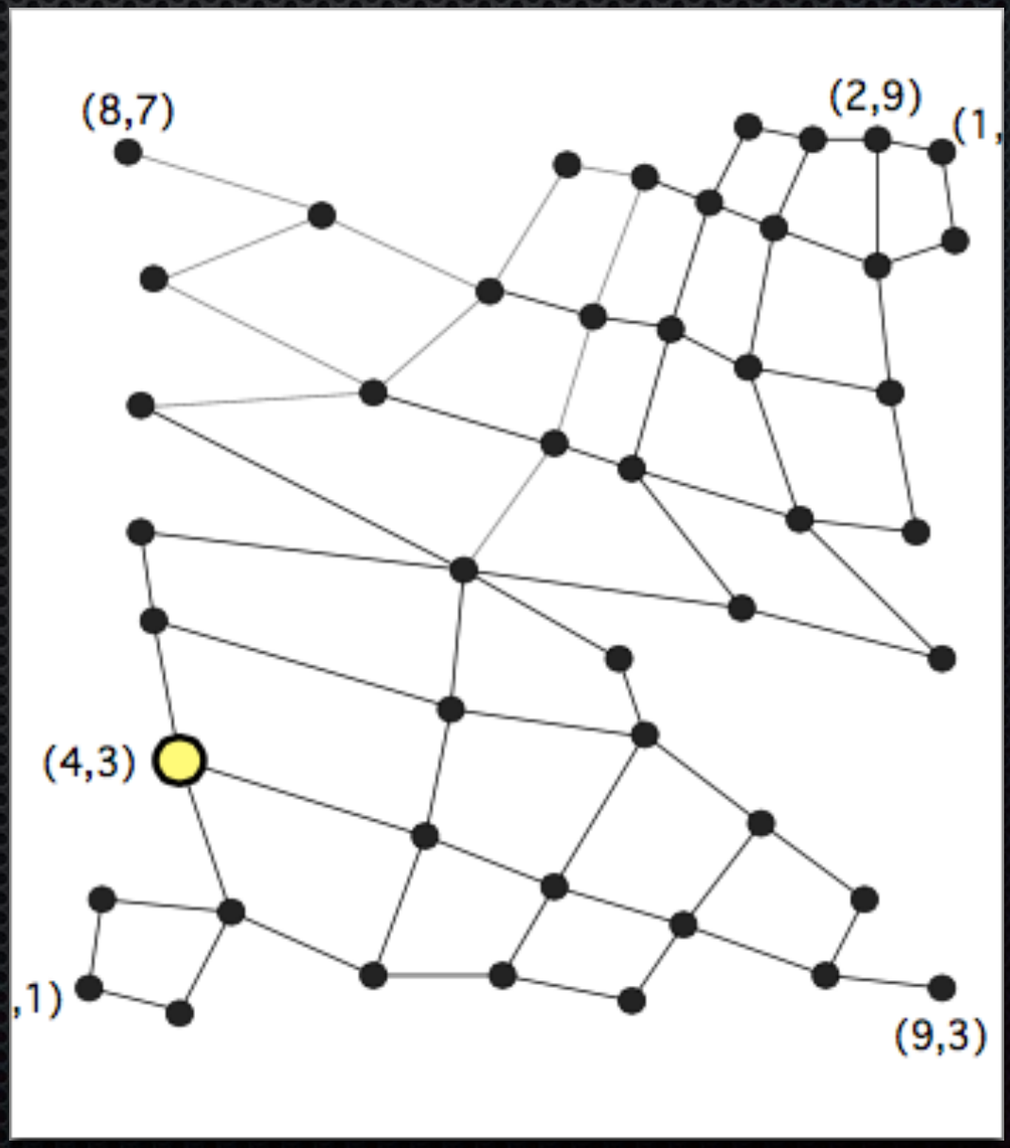
- GPU programming is hard - programming multi-GPU architectures even harder
- *Idea*: automatically fission Accelerate programs into task parallel programs which can be scheduled to multiple processing units
- transformation expressed as set of type-preserving rewriting rules
- multi-device scheduler automatically distributes operations across multiple devices

"Skeletons for distributed topological computations"

David J. Duke & Fouzhan Hossein

- Problem: visualisation of huge data sets representing meteorological, geological, physical models

- Calculate topological abstractions (minima, maxima, and saddle points)



Parallel Skeletons

- Skeletons are basically higher-order function
- Paper investigates the use of parallel skeletons in Eden to implement abstraction algorithm

```
distDC::
  Int ->
   (a-> Bool) ->
  (a -> b) ->
  (a-> [a]) ->
  ([b] -> b)->
  a ->
  b
```

“Generate and Offshore: Type-safe and Modular Code Generation for Low-Level Optimization ”

Bo Naoki Takashima Hiroki Sakamoto Yukiyoshi Kameyama

- Asuna (MetaOCaml library) to implement code-generators for a range of target languages
- Generated code guaranteed to be well typed and well scoped
- Supports parallel code generation via the use of modern CPU features, like SIMD instructions

If you're interested in parallelism

- Attend FHCP, parallelism tracks at ICFP & Haskell Symposium, Simon Marlow's talk at CUFP, Erlang Workshop (concurrency)
- Talk to us!

Thank you!

Image Sources

- https://commons.wikimedia.org/wiki/File:Dortmund_-_Zeche_Zollern24_-_Zentralplatz_07_ies.jpg
- https://commons.wikimedia.org/wiki/File:Programmer_writing_code_with_Unit_Tests.jpg
- https://commons.wikimedia.org/wiki/File:GPU_NVIDIA_NV45_ES_GPU.jpg
- clipboard art