Performing with CUDA

W. B. Langdon
CREST lab,
Department of Computer Science

Pages 423-430

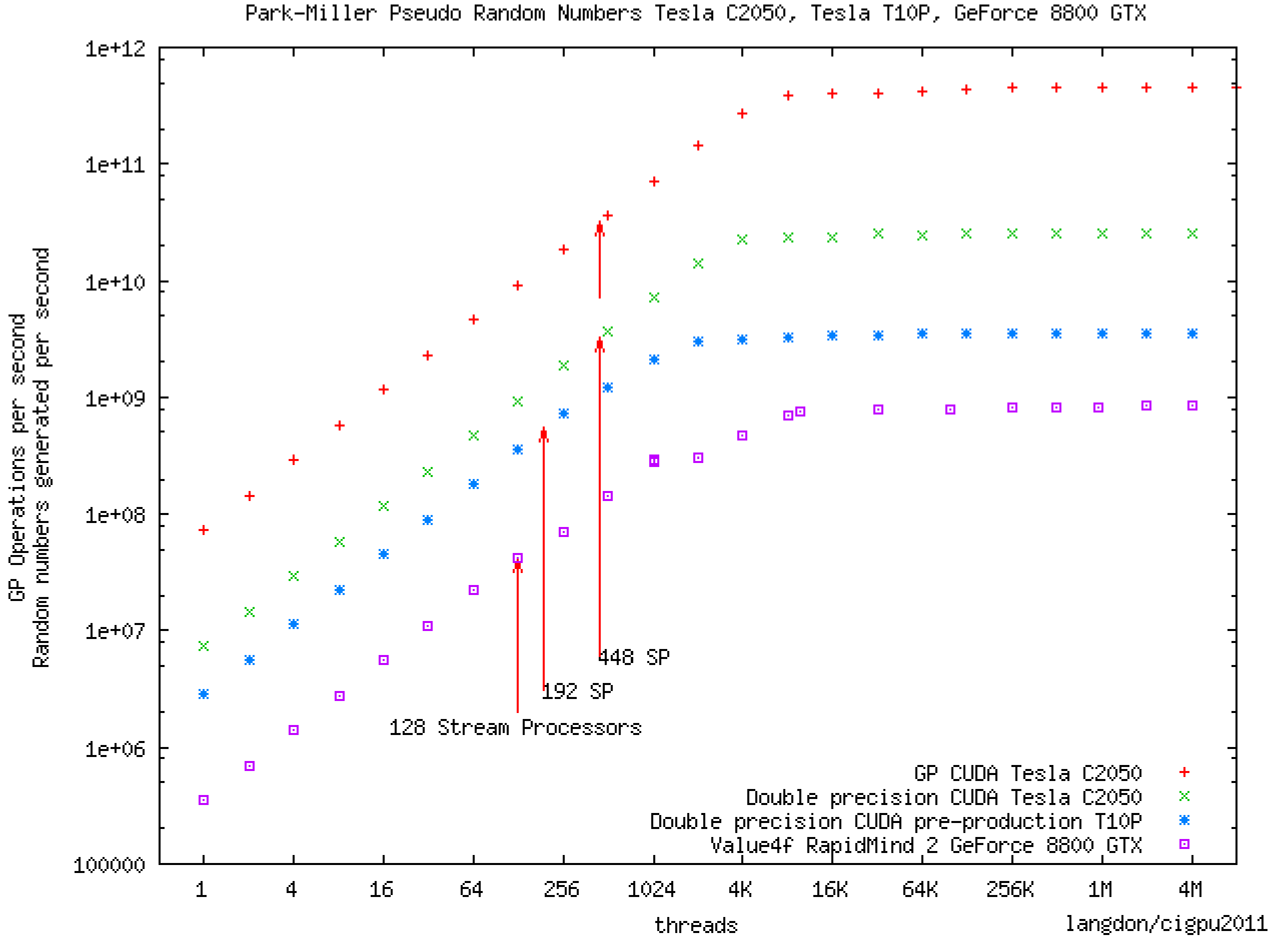
Introduction

- Initial steps
- Concentrate upon what is different about high performance with GPU:
  - Many threads
  - Finding and avoiding bottlenecks
- Conclusions

Before you code

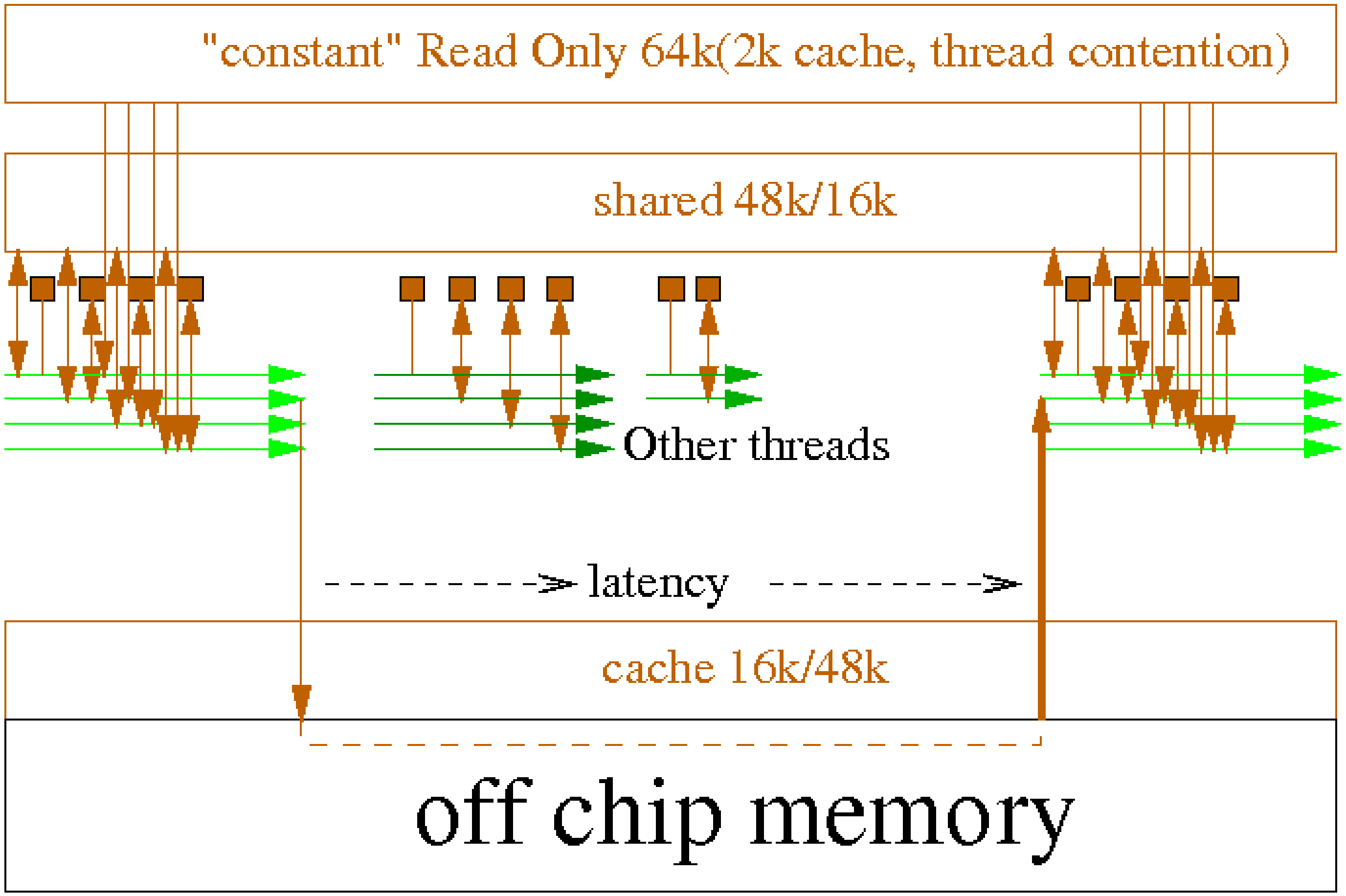
- How much of your new application will be run in parallel? If <90% **stop**.
- EA called “embarrassingly parallel”
- If big population: one thread per member
- May be hard to parallelise fitness function
- How much of GPU’s speed, memory do you need? (Advertised performance is best possible)

GPU computing needs many threads



Best speed ≥ 20× number of stream processors

GPU many threads hide latency



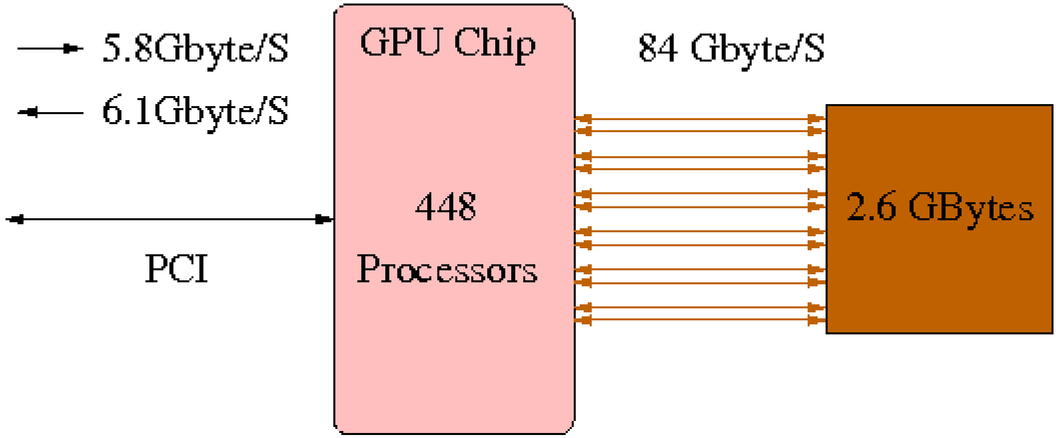
Bottlenecks

CREST

Slowest step dominates

- In a car you know if
  - Doing well, road is wide and smooth
  - In heavy traffic or road is narrow and bendy
- With a GPU it is difficult to tell what is holding you back

Fermi C2050



PCI host↔GPU link always narrower bottleneck than GPU↔on board memory.

Both can be important.

Locate Bottleneck in Design: Host PC↔GPU PCI Bus

- PCI can be estimated in advance
- Number bytes into and back from GPU per kernel call.
- How long to transfer data (byte/bandwidth)
- How long between kernel launches?
  - If <1millisec consider fewer bigger launches
- bandwidthTest (see switches) gives PCI speed.

Other Bottlenecks

- In theory can do the same for GPU-global memory transfers but.
  - Hard to do.
  - PCI can run at 100% usage (pinned memory)
  - Hard to predict fraction of usage inside GPU
  - What effect will caches have?
  - Enough threads to keep both processors and memory buses busy.
  - Atomic and non-coalesced operations may have unexpectedly large impact

Performance by Hacking

- Measuring performance
- Is performance good enough? **Stop**
- Can it be made better? No: **stop**.
- Identify and remove current bottleneck.
- Measure new performance. What is new bootleneck?

Timing whole kernels on host

```
//Time transfer of d_1D_in from PC to GPU
cutilSafeCall( cudaThreadSynchronize() );
cutilCheckError( cutResetTimer(hTimer) );
cutilCheckError( cutStartTimer(hTimer) );

cutilSafeCall(
  cudaMemcpy(d_1D_in,In,In_size*sizeof(int),
                   cudaMemcpyHostToDevice));

cutilSafeCall( cudaThreadSynchronize() );
cutilCheckError(cutStopTimer(hTimer));
const double gpuTimeUp = cutGetTimerValue(hTimer);
gpuTotal += gpuTimeUp;
```

Remember to use cudaThreadSynchronize.

See examples in CUDA SDK sources.

Timing Kernel Code

- Perhaps use GPU’s own clock
- Alter kernel to do operation N+1 times instead of just once.
  - Time per operation ≈ extra kernel time/N
- Ensure new code behaves same as old
- Ensure nvcc compiler does not optimise away your modification

```
//prevent compiler optimising away junk_timing_info
if(in_length<0) d_out=junk_timing_info;
```

- Results can be disappointing: less compute time may mean more time waiting for memory.

CUDA Profiler

- Two parts
  - Counters on GPU, write data to host files
  - User interface to control which counters are active and display results
- Linux Visual profiler not stable
  - Use spreadsheet, gnuplot etc instead
- CUDA Profiler good for measuring:
  - Divergence
  - Cache misses (non-coalesced IO)
  - Serialised access to constant memory

Multiple GPUs

- CUDA requires you to use conventional threads on host (eg pthreads).
- Large overhead on creating GPU data structures on host. So:
  - Create CUDA data once at start of run
  - Create pthreads once at start of run

Other Approaches

- Can you compress data.
  - eg send bytes across PCI rather than int
- Can you keep data on GPU to avoid re-reading it?
- Would it be better to re-calculate rather than re-read?

Conclusions

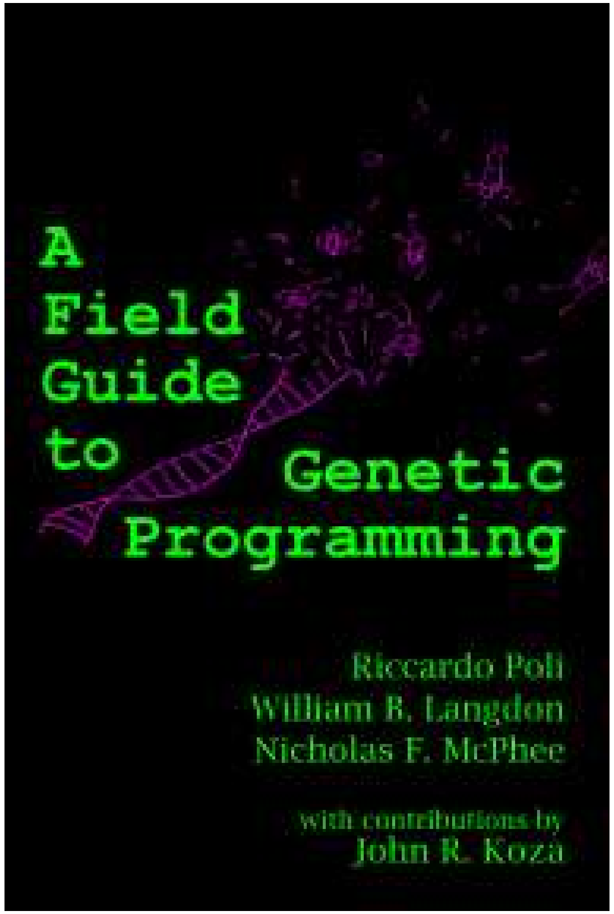
- Design before you start.
  - Will non-parallel part prevent useful speedup?
  - Use lots of threads
- Locate slowest step. Concentrate on it.
- Slowest step usually moving data
- Don't be afraid to waste computation
- Computation is cheap. Data is expensive

END

http://www.epsrc.ac.uk/ EPSRC

A Field Guide To Genetic Programming

http://www.gp-field-guide.org.uk/



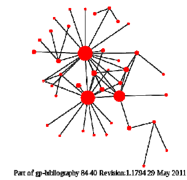
Free PDF

The Genetic Programming Bibliography

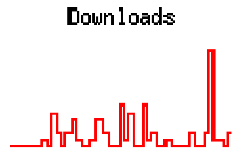
The largest, most complete, collection of GP papers.
http://www.cs.bham.ac.uk/~wbl/biblio/

With 7554 references, and 5,895 online publications, the GP Bibliography is a vital resource to the computer science, artificial intelligence, machine learning, and evolutionary computing communities.

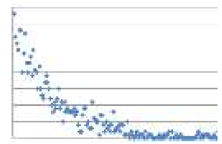
RSS Support available through the Collection of CS Bibliographies.



A web form for adding your entries. Wiki to update homepages. Co-authorship community. Downloads



A personalised list of every author's GP publications.



Search the GP Bibliography at
http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html