GPU computing for real-time de-dispersion in astrophysics

Wes Armour, Mike Giles

mike.giles@maths.ox.ac.uk

Oxford e-Research Centre

Institute for the Future of Computing, Oxford Martin School

Oxford-Man Institute of Quantitative Finance

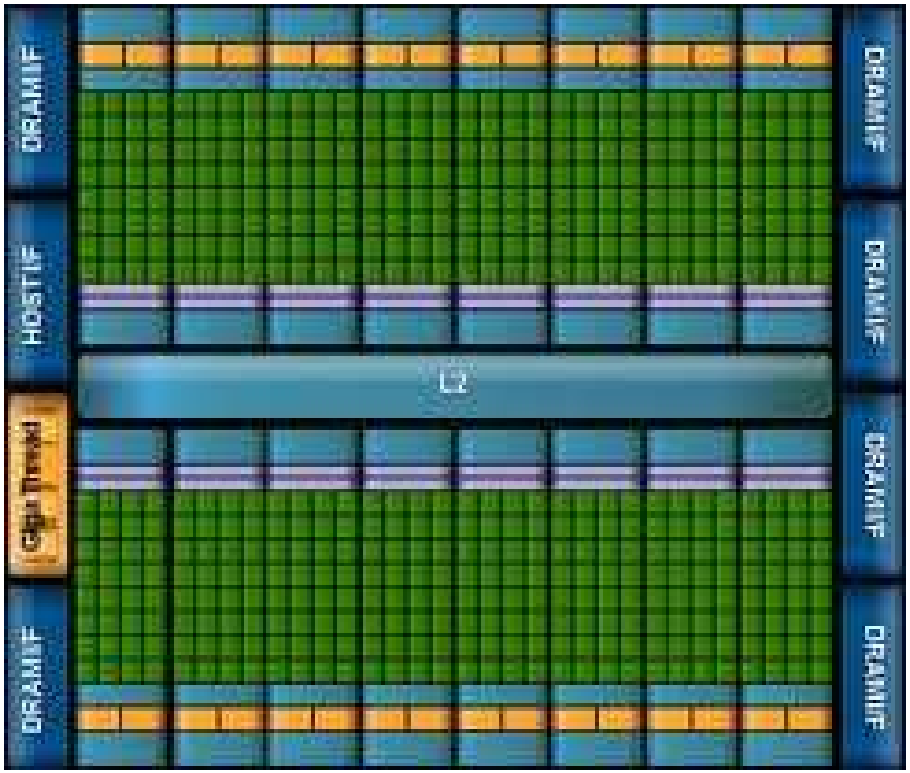
Overview

- GPUs
- radio-astronomy dataflow
- physics and maths
- software design
- performance

GPUs

The latest NVIDIA Fermi GPUs have:

- up to 16 SMs (“Streaming Multiprocessors”) each with 32 cores, 48kB shared memory, 16kB L1 cache
- Up to 6GB of attached graphics memory



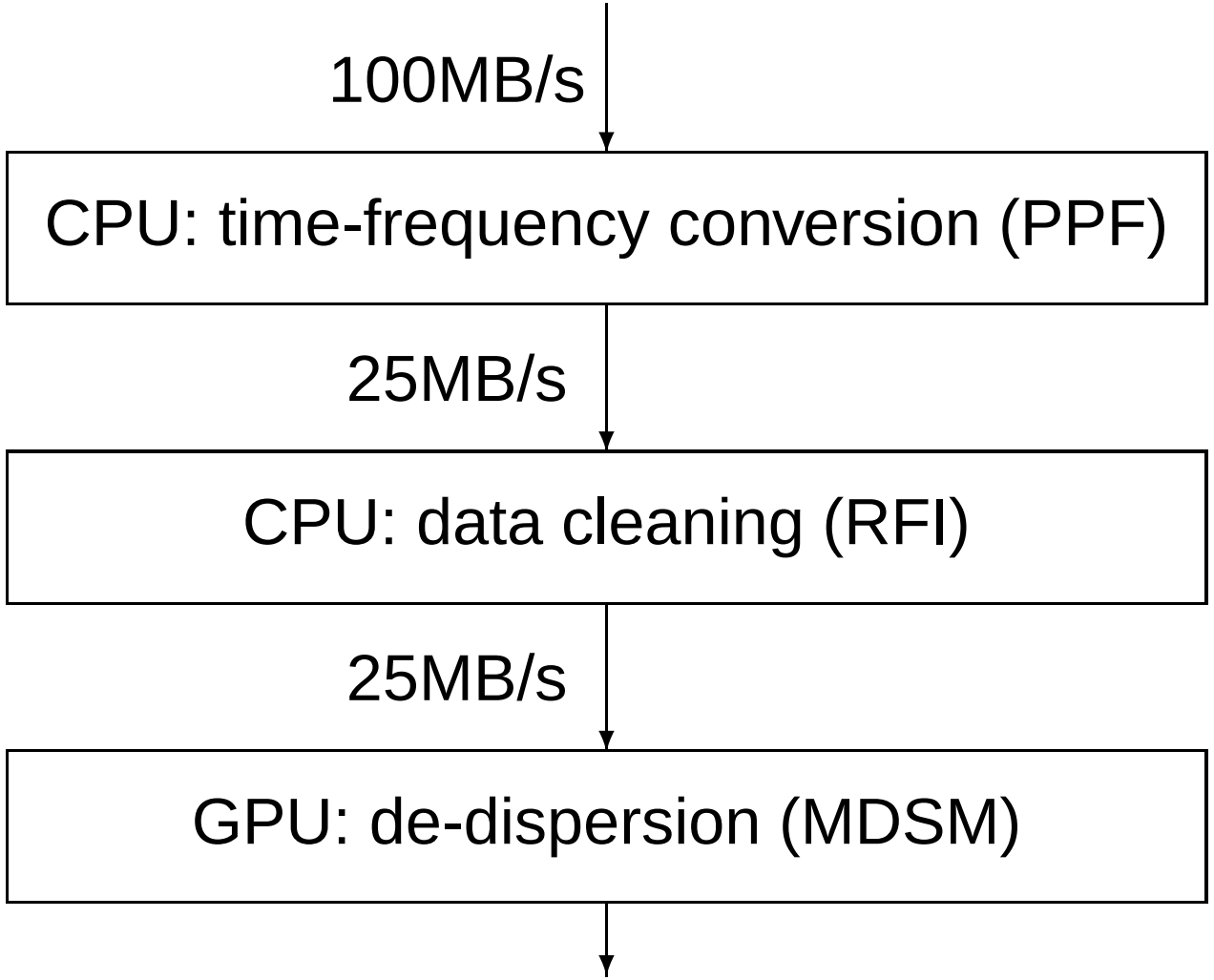
GPUs

Key performance figures for C2070:

- 1 TFlops in single precision, half that in double precision
- 120 GB/s bandwidth from graphics memory to GPU
- 5 GB/s bandwidth across PCIe bus from CPU to GPU
- typically about 10,000 threads executing in parallel

LOFAR dataflow

400MB/s input split into 4 streams of 100MB/s:



Physics

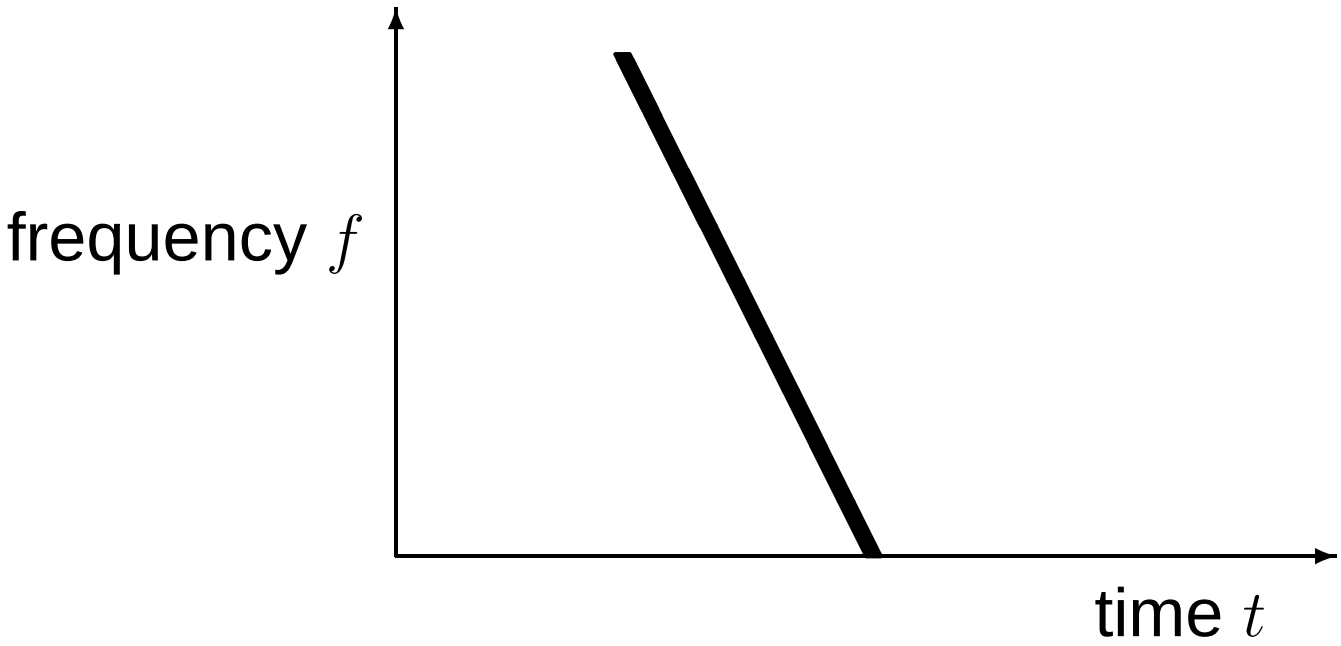
- Pulsars produce a narrow beam of electromagnetic radiation which rotates like a lighthouse beam, so a pulse is seen as it sweeps over a radiotelescope
- The signal is spread over a wide frequency range. If space was an empty vacuum, all the signals would travel at the same speed, but due to free electrons different frequencies travel at slightly different speeds (dispersion)
- The difference in travel time is proportional to distance, so the distance can be deduced from the relative time lag between different frequencies

Physics

The time delay depends on frequency f, and is proportional to the dispersion measure m which corresponds (roughly) to distance:

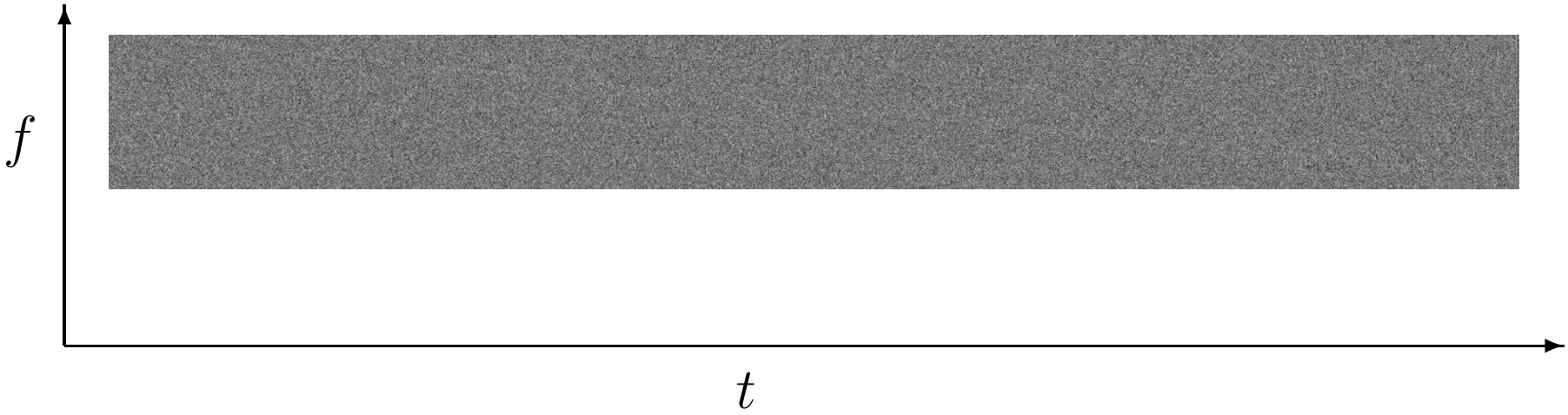
$$\tau = m\ d(f)$$

Since $d(f)$ is known, can work out m from signal data:



Physics

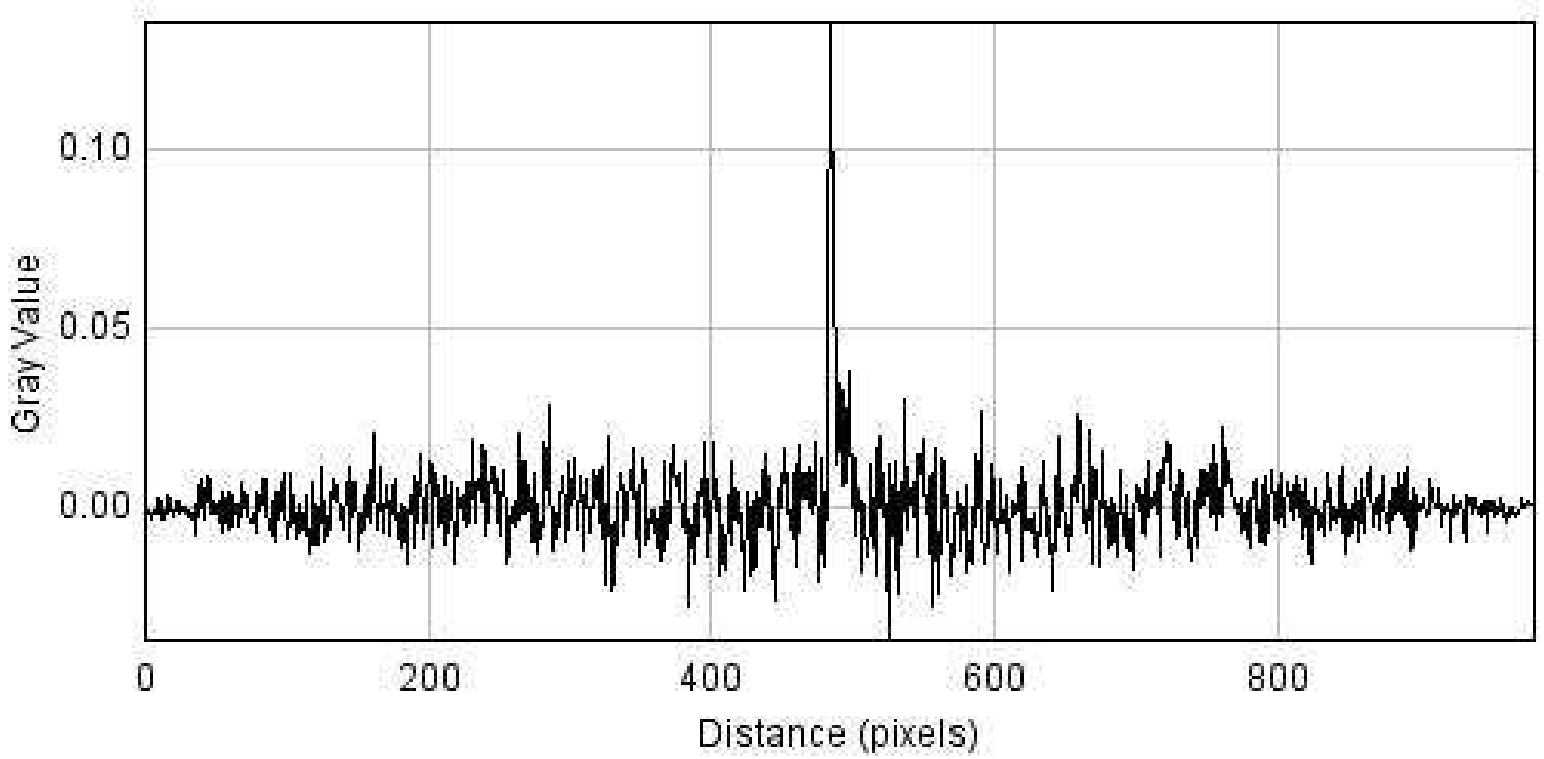
Problem: the signal is often very weak, barely distinguishable from the background noise



Physics

Solution: if we know the right value for m, then we can time-shift the data to correct for the dispersion (i.e. we can de-disperse the signal) then sum over the frequencies

This reinforces the signal relative to the background noise



Physics

New problem: we don't know the right value for m

Solution: try lots of different values for m; the right one is the one that gives a clear signal!

This needs lots of computation – that's why we are interested in using GPUs

Maths

Let

- f be integer frequency index, $0 \leq f < F$
- t be integer time index
- m be integer dispersion measure index, $0 \leq m < M$

Given input data $u(f, t)$, the objective is to compute the output

$$w(m, t) = \sum_f u\left(f,\, t - s(m, f)\right),$$

for an integer shift function $s(m, f)$ which is approximately linear in m, and varies little from m to $m+1$:

$$\max_{m,f} \left| s(m+1, f) - s(m, f) \right| \leq 5 \quad \text{(for our testcase)}$$

Back-of-envelope assessment

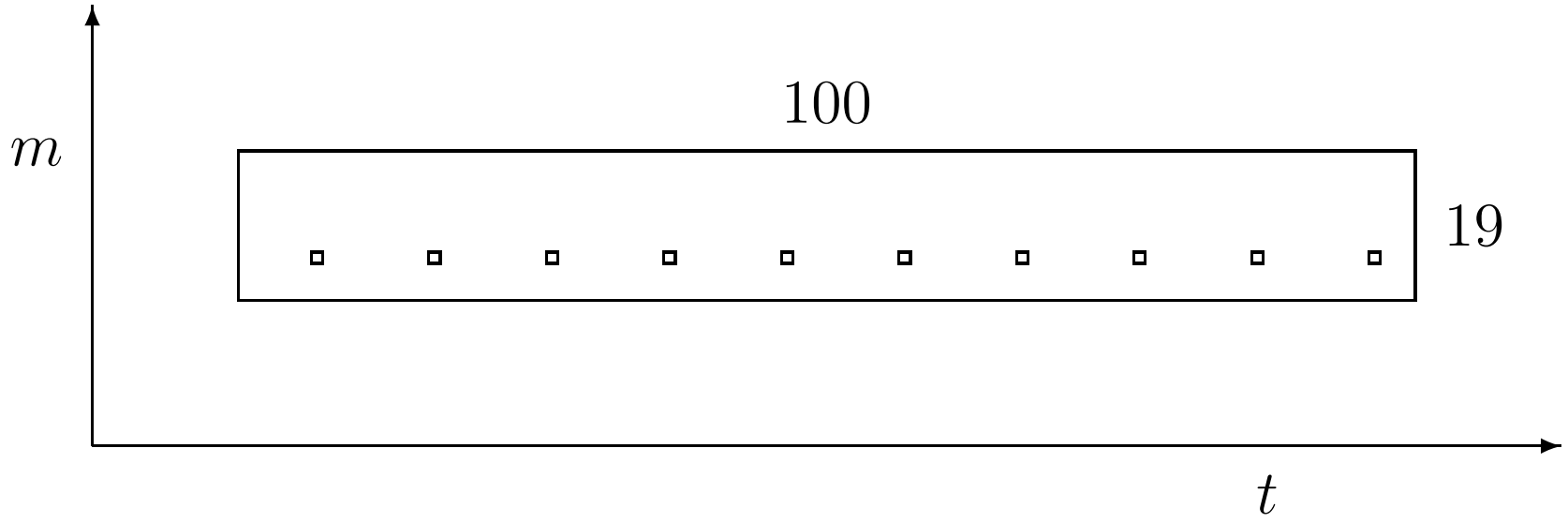
For each time slice t:

- F inputs
- M outputs
- $M\,F$ floating point operations

Typically $F,\ M\ \simeq 1000 - 2000$, so enough computation to hide communication cost of PCIe bus

GPU memory bandwidth also not a problem, provided each input data item isn't transferred too many times

Software design



Rectangular region in dispersion space worked on by a block of 10×19 threads, each handling 10 points using 24 registers

7 thread blocks run simultaneously on each SM – 1330 threads on each SM, almost 20,000 on whole GPU

Software design

Implementation:

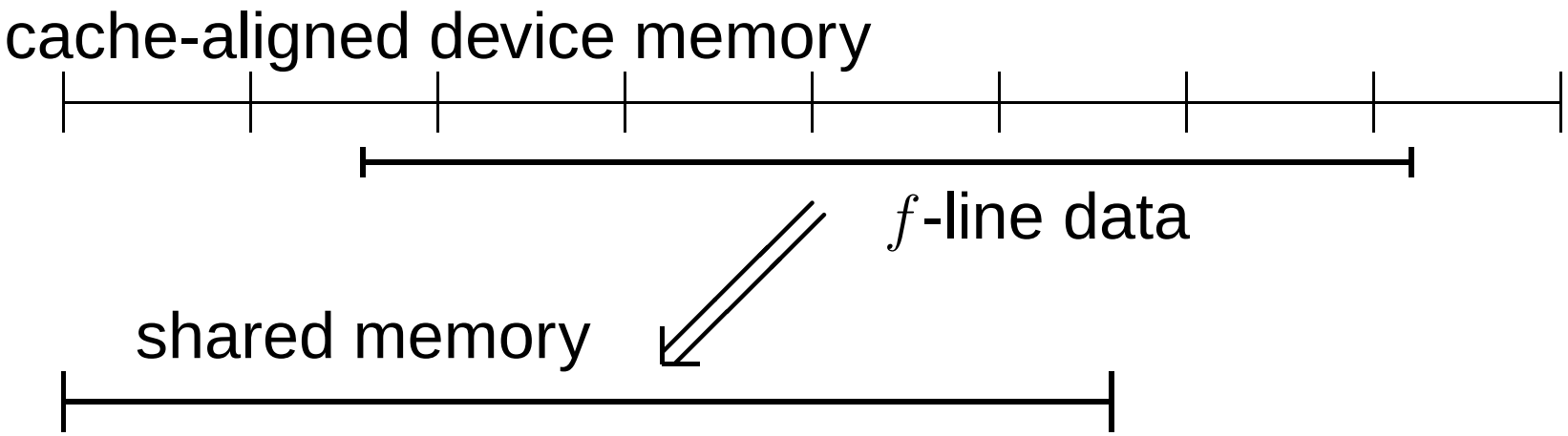
1. load f-line into shared memory
2. sync threads
3. each thread adds shifted values to 10 accumulators
4. sync threads
5. go back to step 1 and repeat for next f-line

- use of shared memory gives data reuse – most data is used 19 times, once for each m-line
- this implementation alternates communication and computation – relies on multiple blocks for overlapping

Software design

Implementation:

- each m-line needs 100 values
- at most a 5-shift between one m-line and the next
- at most a 90-shift for set of 19 m-lines, so at most 100+90 values required (1 per thread)



Performance

Static testcase:

- 2 mins of test data: 3GB
- 0.6s transfer time to GPU: 5GB/s
- 15s processing time
  - approx 500 Gops, roughly evenly split between SP floating point, integer and shared memory reads
  - approx 40 GB/s bandwidth from graphics memory
- overall: achieving 40–50% of peak compute capability and communication

I’m very satisfied with performance, not much scope for improvement – 1 GPU could handle all 4 data streams in real-time

Two lessons learned

Auto-tuning is important:

- needs a lot of fiddling around to determine optimum parameters – not obvious even to an expert
- undergraduate student has developed an open-source flexible auto-tuning package: Flamingo

Optimising data movement is key to performance:

- bandwidth struggling to match huge compute capability
- need to minimise the number of times data is moved
- applies also to CPU code – need good cache behaviour
- can cause problems in the most innocuous of circumstances, e.g. matrix transpose

Final comments

I remain keen on GPUs, but not the only game in town:

- AVX vectors in latest Intel Sandy Bridge CPUs – we will start code optimisation for this very soon since CPU code is currently limiting overall performance
- even longer vector units in new Intel MIC “GPU” (successor to Larrabee, going into new 10 petaflop supercomputer in Texas)
- I think a lot of our GPU experience will carry over to these architectures – need to minimise data movement

Acknowledgements: funded by the Oxford Martin School
`http://www.futurecomputing.ox.ac.uk/`