Runners and riders in GPU steeplechase

Mike Giles

mike.giles@maths.ox.ac.uk

Oxford University Mathematical Institute

Oxford e-Research Centre

NAG Technical Forum

24/06/2010

Overview

Hardware:

- NVIDIA (Fermi)
- AMD
- IBM
- Intel (Knights Ferry)
- CRAY and other systems vendors

Overview

Software:

- CUDA
- OpenCL
- Microsoft
- Intel (Ct & F32vec4)
- PGI
- MATLAB

NVIDIA

New Fermi GPU:

- added L1 / L2 caches to give improved programmability
- added ECC memory for reliability
- much greater double precision speed (roughly 500 GFlops)
- 448 SP cores (can function as 224 DP cores) arranged as 14 groups (SMs) of 32 cores
- each SM has 64KB to be split between L1 cache and local shared memory
- has all hardware features needed for full C++ support

NVIDIA

Assessment:

- not a huge improvement in SP speed (25%?)
- big improvement in DP (factor 2-4$\times$?)
- big improvement in programmability, especially for finite difference applications
- are they putting too much emphasis on HPC market, rather than money-making games market?
  (but they are also doing very well on the low-end)
- however, it is helping them make big strides in HPC
  - Nebulae (#2 in Top 500)
  - Tsubame 2, planned at TITech
  - Jaguar follow-on from CRAY, planned at ORNL

AMD

- still no major push on HPC
- very focussed on games market – doing well there at NVIDIA's expense
- also Fusion chip for integrated CPU/GPU – good for netbooks / laptops
- conversation with AMD person at conference suggests they won't enter the race until maybe 2012/3
- plan to use memory stacking for big increase in memory bandwidth to motherboard slot, instead of using a separate PCIe graphics card?

IBM

- an early runner with the Cell processor
- could argue it never really had a rider – hobbled by poor software development system
- now dropped out of the race – no further development of Cell for HPC
- instead building systems and blades using NVIDIA GPUs

Intel

- has been developing a GPU chip code-named Larrabee
- hit major development problems and has now formally abandoned plans to compete in discrete graphics
- however, out of the ashes has come the Knights Ferry processor which is essentially an HPC GPU without graphics
- announced as a research platform, being made available to a few select groups – no indication if / when it may become a commercial product
- not clear if the market is big enough to sustain Intel's interest

Intel

Knights Ferry:

- (town in CA where Little House on the Prairie filmed)
- 300W chip on PCIe card with 2GB GDDR5 memory
- 32 cores, each with a scalar unit and a vector unit (512-bit == 16 SP or 8 DP, compared to 128-bit SSE and 256-bit AVX vectors)
- simple cores (in-order execution, no branch prediction?) like Fermi
- 2MB coherent cache, compared to 0.8MB on Fermi
- only 4 threads per core (but this may just be minimum?) compared to 16 – 32 typically on Fermi
- unknown memory bandwidth, similar to Fermi?

Intel

Knights Ferry – my assessment:

- hardware design similar in many ways to Fermi (more similar than different)
- I’m told that performance on my LIBOR Monte Carlo application is similar to Fermi, and 10$\times$ better than quad-core Nehalem
- certainly looks interesting, my main reservations concern software and commercial intent

CRAY

- building supercomputers based on Fermi GPUs
- first contract is with ORNL for Jaguar successor
- take Fermi chips from NVIDIA but re-design everything else around it, and possibly also driver software for GPU<–>GPU transfers
- also developing their own compiler to produce PTX code (low-level code, almost at GPU machine level)

Other systems vendors

HP, Dell, IBM, Supermicro:

- building blades and systems using complete cards and software from NVIDIA

SGI:

- has NVIDIA offering but teaming up more with AMD?

SUN:

- who are they?
- James Coomer now at Dell

Software

- clearly no point in good hardware without good software
- feels like back-to-the-future – essentially vector computing at the hardware level, except that they all function as attached co-processors with explicit data exchange with host processor
- key algorithms, difficulties, solutions well established – interesting to see how each company responds

CUDA

- a “grid” of independent “blocks”, each consisting of lots of threads grouped in “warps” of length 32
- each block maps to a single SM (streaming multiprocessor); each warp is effectively a vector, with all threads in a warp executing the same instruction at the same time
- explicit use of local shared memory to communicate between threads within a block
- massive multithreading is key to good performance
- big challenge in implementation can be limiting number of registers used

CUDA

- code is written from the point of view of a single thread – the compiler generates the appropriate vector instructions for each warp.
- CUDA 3.0 a big advance on CUDA 2.3 – added support for a lot of C++ features, and new Fermi hardware features
- still a bit buggy, but 3.1 due out soon
- overall, I find CUDA easy to work with – code often looks quite similar to original C/C++ code
- at SIAM Conference on Parallel Processing in January, everyone in this area was using CUDA – lack of competitive hardware from AMD meant no incentive to move to OpenCL

OpenCL

- open standard pushed by Apple and agreed with NVIDIA, AMD, Intel, IBM
- based on early release of CUDA's lower-level device API, so not as full-featured as CUDA 3.0
- ARM and Imagination Technologies both have OpenCL compiler teams, so OpenCL can/will run on iPhones, iPads and other smart mobile devices
- OpenCL applications developed for
  - HD video codecs
  - image processing (Adobe)

OpenCL

- makes a lot of sense for low-end consumer applications where platform-independence is essential – but hasn't taken off for HPC
- could change *if* Intel adopts OpenCL for Knights Ferry, *or* AMD introduces good high-end hardware
- for now, I'm ignoring it for HPC, but keeping an eye open
- OpenCL 1.1 spec just released by Khronos (same organisation that manages OpenGL and a variety of other standards)

Microsoft

- the one big player not on the OpenCL working group
- working well with NVIDIA and AMD to ensure Windows is OK for CUDA and OpenCL?
- I still wouldn’t recommend it in a multi-user environment
- their preferred solution is DX Compute, part of DX11 – could be a serious competitor to OpenCL in the games market, but not in HPC

Intel

Software is so important they give you lots of choices:

- `icc` + OpenMP
- low-level vector instructions (F32vec4)
- TBB (thread building blocks)
- Cilk++
- Ct
- OpenCL (under development?)

  `www.khronos.org/developers/library/2009-hotchips/Intel_OpenCL-and-CPUs`

- as a developer I find this confusing – no guidance on which is best under what circumstances
- only Intel has the size to do this

F32vec4

This is the low-level way to get maximum performance out of Intel vector units:

- explicit datatype, a vector of `4 floats` to match the size of the SSE vectors
- corresponding F64vec2 datatype for `doubles`
- could use a `typedef` or templates to generalise to F32vec8 for AVX and F32vec16 for Knights Ferry
- user has to pack original data into the vectors, and unpack afterwards
- MKL now has math functions for this datatype

F32vec4

- current chips support conditional assignment

```
b[i] = (a[i]>0) ? c[i] : d[i];
```

- I think future chips may have *predicated* operations for conditional execution

```
if (a[i]>0) b[i] = c[i]*d[i];
```

- vector code ought to be produced by current Intel compiler for appropriate flags, but in practice it seldom manages to generate it – why?
- code produced by Intel for LIBOR test case looks ugly, horribly verbose and unreadable
- I don’t want to code at this low level, but Intel compiler ought to be able to do much better?

Ct

- now on beta release
- full release sometime next year?
- I think this is Intel's preferred approach, and executes almost as fast as F32vec4 code
- I think it is comparable to CUDA in terms of readability – uses its own C/C++ language extensions
- will be interesting to see how quickly it is adopted, and how good a job Intel do in "selling" it

PGI

- one version of PGI compiler uses pragmas to generate CUDA executables from plain C / FORTRAN
- another version defines a FORTRAN equivalent of CUDA – I’m going to use this for a CFD project with Rolls-Royce
- hopefully, they might also develop a CUDA compiler to generate vector code for Intel chips – ought to be quite straightforward once you have a CUDA parser

MATLAB

CUDA support coming in next release of Parallel Computing toolbox :

- special CUDA vector/matrix datatype
- dense linear algebra operations using CUDA BLAS
- alternatively, user can define a scalar function to be applied to corresponding elements of vector arguments
- will be interesting to see how well it is taken up
- team of about 20 working on it in Mathworks, half in Cambridge UK
- no plan yet to write RNGs

Conclusions

- on the hardware side, NVIDIA is the only game in town at present for HPC, and so CUDA is the software choice
- Intel's Knights Ferry is worth watching – it could be good, but right now I wouldn't bet on it coming to market
- I'm maybe too dismissive of AMD – I don't have the time to properly investigate their hardware/software
- I'm hoping PGI will be the solution to code portability
- MATLAB could also be the solution for many people?

What does it mean for NAG?

- GPU computing taking off at low-end and high-end
- software picture is a bit confusing – CUDA is best for now, but the future is unclear
- hardware picture is much clearer – return of vector computing
- look at compute-intensive algorithms / routines and think about vectorisation
- distinguish between routines where users just want the right answer, and those where they want performance
- need to be nimble – others like Mathworks jumping in
- also good opportunities in HPC training – over 60 signed up for my CUDA course in July