GPUs: what are they good for?

Mike Giles

mike.giles@maths.ox.ac.uk

Oxford e-Research Centre

University of Oxford

Fujitsu Research Laboratories: Feb 1, 2011

Outline

- CPUs and GPUs: comparison, trends and opinions
- what to look out for in GPU applications
- libraries for GPUs
- my experience with GPU programming
- OP2: an open-source library for unstructured grid applications

CPUs

Intel's Sandy Bridge CPUs:

- 2-8 cores, each hyperthreaded
- complex cores with out-of-order execution and branch prediction to avoid delays when waiting for data
- each core has an AVX vector unit (8 floats or 4 doubles)
- 30 DP GFlops/core (15 GFlops without AVX)
- some models also have integrated graphics units – mainly fixed function, not useful for HPC?
- 64kB L1 and 256kB L2 cache/core
- up to 8MB shared LLC (Last Level Cache)
- bandwidth to main DDR3 memory is around 30GB/s

GPUs

NVIDIA's Fermi GPUs:

- 14 "units" called Streaming Multiprocessors (SMs) which have:
  - 32 simple in-order SIMD cores which act as a vector unit $\Longrightarrow$ 37 DP GFlops/SM
  - 16-48 threads/core to hide delays
  - 32k 32-bit registers
  - 16kB L1 cache
  - 48kB shared memory
- GPU also has
  - 384kB unified L2 cache
  - 150 GB/s bandwidth to main GDDR5 memory
  - 5 GB/s bandwidth to CPU across PCIe bus

Differences

- very different if AVX vectors are not used; not so different if they are
- factor 5-10$\times$ difference in peak GFlops
- factor 5$\times$ difference in memory bandwidth
- slow CPU-GPU link a potential bottleneck
- CPU has cache coherency at L1 level; GPU avoids the need through language construct which requires no interference between different “thread blocks”
- GPU uses much more multithreading; requires a lot of registers so each thread has its own set

Future?

GPUs: more of the same

- more memory bandwidth
- more SMs?
- more cores per SM?
- more registers per core?
- more shared memory?
- more GPUs per graphics card?

Biggest headache: PCIe bottleneck

Solution: add ARM cores to run O/S and external I/O

(ties in nicely with Tegra SoC strategy at low-end for smartphones and tablets)

Future?

CPUs:

- increase cores
  - but will general purpose software use them?
  - . . . and is cache coherency scalable?
- increase AVX vector length?
  - “simple” response to GPUs for HPC
  - . . . but what is the programming model?

Biggest headache: main memory bandwidth

Solution: memory stacking?

Big question: why care about HPC?

My opinion

- NVIDIA have a clear vision, both for the hardware and the software – and it's important to have both
- Intel may have a good roadmap for the hardware, but
  - I don't think there's a clear software vision
  - they're trapped by their existing customer base with lots of sequential applications
  - HPC is a negligible revenue source
- I think NVIDIA GPUs will have a major presence in HPC for at least the next 5 years
- the real commercial fight is at the SoC level

Supercomputing

- #1 Tianhe-1A: 7168 NVIDIA Fermi GPUs
- #3 Nebulae: 4640 NVIDIA Fermi GPUs
- #4 Tsubame-2: 4224 NVIDIA Fermi GPUs

New US petaflop systems coming soon:

- NCSA/UIUC (IBM Blue Waters – 300k 8-core CPUs)
- Oak Ridge (CRAY XE6 with NVIDIA GPUs)

US exascale plans currently assume an evolution
from one of these two architecures

Is a GPU suitable for HPC?

In general, I expect to get a 5-10$\times$ speedup on a single GPU compared to two multicore Xeons.

However, when I look at a possible brand new application, there are a few things I look out for:

- where does the data live?
- is there a lot of conditional branching?
- are there existing libraries I can use?

Is a GPU suitable for HPC?

Where does the data live?

The 5GB/s bandwidth of the PCIe bus connecting the CPU and GPU can be a major bottleneck.

Need 200–1000 operations per variable transferred for the transfer cost to be negligible.

Apart from a few applications such as dense linear algebra (N^3 compute versus N^2 data) it generally means moving the whole application over onto the GPU.

Porting big applications is easier on CPUs – use OpenMP then concentrate on vectorising “hot spots”

Is a GPU suitable for HPC?

Is there conditional branching?

Old vector architectures (inc. Fujitsu's?) executed both sides of a conditional branch, and used a logical merge operation to keep the results wanted.

Similarly, GPUs (and AVX vector units) use predicated instructions where it's only performed for required threads within thread warp (32 threads).

With a lot of branching, can lose a factor 32 in performance – then no benefit compared to scalar execution on CPUs.

(Can sometimes overcome this by re-structuring the code – needs some thought / ingenuity.)

Is a GPU suitable for HPC?

Are there useful libraries?

Writing really efficient GPU code requires some expertise. Application experts shouldn't re-invent the wheel – should always look to exploit libraries written by GPU experts.

Often, the key algorithms and techniques date back to the days of CRAY and Fujitsu vector supercompuers, and Thinking Machines' massively-parallel Connection Machine.

e.g. binary tree reduction, extension to parallel scan (prefix sum) and its use in radix sort.

Phil Colella and the 7 dwarfs

- senior researcher at Lawrence Berkeley National Laboratory
- talked about “7 dwarfs” of numerical computation in 2004
- expanded to 13 by a group of UC Berkeley professors in a 2006 report: “A View from Berkeley”

  `www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf`
- key algorithmic kernels in many scientific computing applications
- very helpful to focus attention on HPC challenges and development of libraries and problem-solving environments/frameworks.

Dense linear algebra

- CUBLAS
  - library provided / maintained by NVIDIA
- MAGMA
  - a new LAPACK for GPUs
  - Jack Dongarra, Jim Demmel and others
- FLAME
  - similar, but being developed by Robert van de Geijn at UT Austin with various collaborators
- CULAtools
  - similar, but developed by a company, EM Photonics

Sparse linear algebra

- iterative solvers:
  - CUSPARSE library for efficient sparse matrix-vector multiplication developed / maintained by NVIDIA
  - Andreas Klöckner (Brown University) has “Iterative CUDA” package based on same SpMV products
  - Manfred Liebmann & colleagues (University of Graz) has implemented algebraic multigrid

    `www.austriangrid.at/fileadmin/uploads/media/talk_haase_ag3.pdf`
- commercial direct solvers:
  - Access Analytics (ex-Boeing Computer Services)
  - ANSYS/Acceleware
  - Robert Lucas (ISI/USC)
  - Grusoft

Spectral methods

- CUFFT
  - library provided / maintained by NVIDIA
  - significant input from Satoshi Matsuoka and others at Tokyo Institute of Technology `www.voltaire.com/assets/files/Case studies/titech_case_study_final_for_SC08.pdf`
- nothing else needed?

N-body methods

- NAMD / VMD (UIUC)
  - molecular dynamics codes
- OpenMM (Stanford)
  - open source package for molecular modelling
- paper by Mark Harris (NVIDIA) and others

  `http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html`
- recent work on fast multipole methods by
  - Barba and Yokota (Boston University)

    `www.maths.bris.ac.uk/~maxry/publications/2009Yokota,R2.pdf`
  - Lashuk *et al* at Georgia Tech

    `www.ma.utexas.edu/users/lexing/publications/sc09.pdf`

Structured grids

- lots of people have developed one-off applications
- Fermi has greatly simplified single-GPU applications
- Graham Pullan and Tobias Brandvik (Cambridge)
  - most impressive results I've seen, and a general-purpose multi-GPU framework
- Jonathan Cohen (NVIDIA Research)
  - developing a library called OpenCurrent:

    `kac.maths.ed.ac.uk/NSF-NAIS/Edit/Slides/Cohen.pdf`

May be other general-purpose work I'm not aware of

Unstructured grids

Several projects underway:

- OP2 (Oxford / Imperial College)
- Liszt (Stanford)
- German collaboration (DLR, T-systems, and others)
- Rainald Löhner (GMU – Washington DC)

Again, there may be other work I’m not aware of

Monte Carlo

- I’ve worked with NAG to develop a GPU library with RNG and related routines
  - mrg32k3a, Mersenne Twister and Sobol
  - uniform, exponential, Normal and gamma output distributions
  - Brownian bridge construction
  - more to come
  - `www.nag.co.uk/numeric/GPUs/`
- NVIDIA has included my `erfinv` function in their math library, and produced a RNG library CURAND
  - XOR-shift, Mersenne Twister and Sobol (based on my code)

Summary

- active work on all of the dwarfs
- in most cases, significant effort to develop general purpose libraries or frameworks, to enable users to get the benefits without being CUDA experts

My experience

- started in 2007 when NVIDIA released CUDA software environment – previously using GPUs for scentific applications was too tough
- also tried Clearspeed accelerator – no harder to program, but didn’t deliver great price / performance
- haven’t tried the IBM Cell – feedback from others suggests I was wise/lucky, and IBM have killed it anyway
- also haven’t tried AMD’s GPUs – their OpenCL compiler is still immature

My experience

- started with Monte Carlo simulations
  - very easy, up to 100$\times$ speedup in single precision compared to 1 CPU thread
- then moved to random number generation
  - more interesting
  - early hardware didn't have double precision support so I had to use non-standard implementation
  - I also had to improve the inverse error function implementation because it branched too much
  - 35$\times$ speedup for same RNG generator compared to Intel's VSL library on a Xeon

My experience

- next step was simple structured grid PDE methods in computational finance
  - explicit time-marching (like Jacobi iteration for solving elliptic PDE)
  - ADI implicit time-marching (Alternating Direction Implicit)
  - 10$\times$ speedup in single precision compared to two quad-core Xeons

My experience

Community building efforts:

- EPSRC-funded Many-core and Reconfigurable Supercomputing Network
  - FPGAs, GPUs and other accelerators
  - latest MRSC conference in Bristol in April
- 1-week CUDA Programming course
  - 40 "students" in 2009, 80 in 2010
  - roughly 40% from Oxford, 40% from other universities, 20% from industry and government
- various research groups in Oxford
  - stochastic modelling in mathematical biology
  - particle filters in Bayesian statistical analysis
  - real-time data processing in astrophysics

More opinions

Problem:

- lots of potential to be exploited from GPUs and CPUs with vector units
- programming too complex for many users
- also, hardware and underlying software still evolving quite rapidly

Solution:

- numerical libraries and domain-specific high-level languages
- simple high-level abstraction for application users
- computing experts provide optimised implementations for multiple target platforms

OP2 History

OPlus (Oxford Parallel Library for Unstructured Solvers)

- developed for Rolls-Royce 10 years ago
- MPI-based library for HYDRA CFD code on clusters with up to 200 nodes

OP2:

- open source project
- keeps OPlus abstraction, but slightly modifies API
- an “active library” approach with code transformation to generate CUDA or OpenCL code for GPUs, and OpenMP/AVX code for CPUs

OP2 Abstraction

- sets (e.g. nodes, edges, faces)
- datasets (e.g. flow variables)
- mappings (e.g. from edges to nodes)
- parallel loops
  - operate over all members of one set
  - datasets have at most one level of indirection
  - user specifies how data is used
    (e.g. read-only, write-only, increment)

OP2 Restrictions

- set elements can be processed in any order, doesn't affect result to machine precision
  - explicit time-marching, or multigrid with an explicit smoother is OK
  - Gauss-Seidel or ILU preconditioning in not
- static sets and mappings (no dynamic grid adaptation)

OP2 User build processes

Using the same source code, the user can build different executables for different target platforms:

- sequential single-thread CPU execution
  - purely for program development and debugging
  - very poor performance
- CUDA / OpenCL for single GPU
- OpenMP/AVX for multicore CPU systems
- MPI plus any of the above for clusters

GPU Parallelisation

Could have up to 10^6 threads in 3 levels of parallelism:

- MPI distributed-memory parallelism (1-100)
  - one MPI process for each GPU
  - all sets partitioned across MPI processes, so each MPI process only holds its data (and halo)
- block parallelism (50-1000)
  - on each GPU, data is broken into mini-partitions, worked on separately and in parallel by different functional units in the GPU
- thread parallelism (32-128)
  - each mini-partition is worked on by a block of threads in parallel

Airfoil test code

- 2D Euler equations, cell-centred finite volume method with scalar dissipation (miminal compute per memory reference – should consider switching to more compute-intensive “characteristic” smoothing more representative of real applications)
- roughly 1.5M edges, 0.75M cells
- 5 parallel loops:
  - `save_soln` (direct over cells)
  - `adt_calc` (indirect over cells)
  - `res_calc` (indirect over edges)
  - `bres_calc` (indirect over boundary edges)
  - `update` (direct over cells with RMS reduction)

Airfoil test code

Current performance relative to a single CPU thread:

- 35× speedup on a single GPU
- 7× speedup for 2 quad-core CPUs

OpenMP performance seems bandwidth-limited – loops use in excess of 20GB/s bandwidth from main memory.

CUDA performance also seems bandwidth-limited:

count	time	GB/s	GB/s	kernel name
1000	0.2137	107.8126		save_soln
2000	1.3248	61.0920	63.1218	adt_calc
2000	5.6105	32.5672	53.4745	res_calc
2000	0.1029	4.8996	18.4947	bres_calc
2000	0.8849	110.6465		update

Conclusions

OP2:

- a new open-source high-level framework for parallel execution of algorithms on unstructured grids
- looks encouraging for providing ease-of-use, high performance, and longevity through new back-ends
- next step is addition of MPI layer for cluster computing

GPUs:

- a major development in HPC
- likely to have coninuing impact for next 5 years
- more work needed to simplify their use by application scientists

Acknowledgements

- Gihan Mudalige (Oxford)
- Paul Kelly, Graham Markall (Imperial College)
- Nick Hills (Surrey) and Paul Crumpton
- Leigh Lapworth, Yoon Ho, David Radford (Rolls-Royce) Jamil Appa, Pierre Moinier (BAE Systems)
- Tom Bradley, Jon Cohen and others (NVIDIA)
- Jacques du Toit, Robert Tong (NAG)
- EPSRC, TSB, NVIDIA, Rolls-Royce and NAG for financial support
- Oxford Supercomputing Centre