Some (strong) opinions on HPC and the use of GPUs

Mike Giles

mike.giles@maths.ox.ac.uk

Oxford-Man Institute of Quantitative Finance

Oxford University Mathematical Institute

Oxford eResearch Centre

July 2-3, 2009

Apologies

- I’m a mathematician/engineer, not a computer scientist, and I’m well aware how much I don’t know
- my intention is to provoke discussion, not cause offence
- where I am wrong / ignorant, even that’s relevant because I’m a relatively well informed user

FPGAs

I do not believe in FPGAs:

- too hard to program
- benefits are not sufficient to justify the effort
- not convinced the future roadmap will change this
- might be good in important niche areas (e.g. bioinformatics?)

GPUs

I do believe in GPUs:

- factor 10$\times$ improvement in energy efficiency and price / performance compared to two multicore CPUs
- produced in vast quantities so gain economies of scale
- relatively easily programmed in CUDA or OpenCL – C with extensions and some C++ features
- already used in several top supercomputers
- great excitement in academic supercomputing, lots of PhDs and post-docs working on major applications
- I think 20% of academic supercomputing spend in next 5 years will be on GPU clusters

GPUs

- NVIDIA, IBM, AMD and Intel all producing GPUs
- NVIDIA has good headstart on software side with CUDA environment
- new OpenCL software standard (based on CUDA and pushed by Apple) will probably run on all platforms
- driving applications are:
  - computer games “physics”
  - video (e.g. HD video decoding)
  - computational science
  - computational finance
  - oil and gas
  - medical imaging

Why GPUs will stay ahead

Technical reasons:

- SIMD cores (instead of MIMD cores) means larger proportion of chip devoted to floating point computation
- tightly-coupled fast graphics memory means much higher bandwidth

Commercial reasons:

- CPUs driven by price-sensitive office/home computing; not clear these need vastly more speed
- CPU direction may be towards low cost, low power chips for mobile and embedded applications
- GPUs driven by high-end applications – prepared to pay a premium for high performance

CPU development

CPUs are also evolving:

- “Westmere” will have 10 cores by end of 2011
- SSE vectors (4 `float` or 2 `double`) will double in length with AVX in 2010, and then may double again in 2012?
- CPUs may also become increasingly heterogeneous – e.g. 3 “normal” cores plus a graphics core?
- applications will need to be substantially re-written to obtain the benefits

Low-level software view

GPUs:

- multiple units each with a collection of SIMD cores operating like a vector unit
- programmed from a thread point of view

CPUs:

- multiple cores each with a vector unit
- software is a mix of scalar and vector instructions

Quite similar in terms of limitations of vector execution (e.g. branching)

Software opinions

I don’t believe in auto-parallelising C / C++ compilers solving all the problems:

- after 25 years I have low expectations
- general purpose compilers (without domain-specific knowledge) seem to have trouble identifying inherent parallelism

Software opinions

I don’t believe in new general purpose languages:

- unimpressed by Fortress
- haven’t investigated Chapel
- very hard for a new general purpose language to compete against C / C++ / Java
- programming doesn’t just need a language, it needs the development environment – this takes time to develop

Software opinions

I do believe in extensions, parallel libraries and program transformations for codes written in C / C++:

- start from a familiar language with a mature development environment
- add the parallel elements needed to get good performance
- can’t do it all with libraries – need tools for program transformation to generate code for target hardware

Software opinions

In considering HPC applications / parallelisation challenges, helpful to look at 7 dwarfs (Phil Colella / A View from Berkeley):

- dense linear algebra
- sparse linear algebra
- spectral methods (FFT's)
- N-body problems (molecular dynamics)
- structured grids
- unstructured grids
- Monte Carlo / Google MapReduce

Software opinions

I do believe in lazy execution on GPUs for high-level object oriented languages:

- current developments for C++ and MATLAB – also wanted for R
- OO features used to build up an execution DAG for vector objects resident on GPU
- DAG transferred (with JIT compilation?) and executed when needed
- I'd like to see work on packages which support multiple front-ends (high-level languages) and back-ends (low-level languages on target hardware)

Software opinions

I also believe in program generators:

- application-specific high-level specification (e.g. SciComp in computational finance)
- generation of low-level target-specific code (e.g. for multicore CPUs, or GPUs, or AVX for CPUs)
- I'd like to see work on packages which makes it easy to define a new application-specific high-level languages and transform it into code for multiple back-ends

Software opinions

In some application areas, it would be helpful if low-level language supported the execution of task DAGs

- particularly important in linear algebra where big matrix operations get expressed as a sequence of small ones with various dependencies
- want to declare the tasks and the dependencies, and leave to run-time system to manage it

Other than this, I’m quite comfortable with low-level languages like CUDA and OpenCL

Education and training

We need a lot more of both for both computer science and application science students and post-docs:

- parallel computing as part of undergraduate science curriculum
  – I'd settle for OpenMP as an addition to existing C/C++ teaching in science areas
- all computational science PhDs and post-docs should learn OpenMP, and have some exposure to MPI and CUDA/OpenCL
- courses also need to be put on for industry

Final thoughts

Without help from computer science, the HPC community will get on and make the most of GPUs through

- parallel libraries
- community self-help
- hard work

With help from computer science, it could make our jobs a lot easier (so more productive) and motivate a lot of interesting computer science research.