Some trends in HPC, and 3 questions

Mike Giles

Oxford University Mathematical Institute
Oxford e-Research Centre

IFIP WG 2.5 talk, Oxford

August 4th, 2016

Outline

Some trends, followed by 3 questions:

- fat or thin nodes?
- FPGAs: has the time finally come?
- high-level frameworks: lessons learned?

Trends

1) Performance is achieved through parallelism, and in particular vector processing:

- don't want lots of chip dedicated to "command & control"
  – instead, cores work in small groups, all doing the same instruction at the same time, but on different data

  (similar to old days of vector computing on CRAY supercomputers)

- on NVIDIA GPUs, cores work in groups of 32 (a thread *warp*)

- CPUs also have vector units (SSE, AVX) which are getting longer

  (256-bit on most, but 512-bit on Intel's Xeon Phi, coming soon to regular Xeons – "Skylake" in 2016/17)

- tricky for algorithms with lots of conditional branching, but there are various algorithmic tricks that can be used

Trends

2) Multithreading is also very important:

- CPU cores use complex, out-of-order execution for maximum single thread performance
- many-core chips use simple in-order execution cores, and rely instead on multithreading
- with 4–10 threads per core, hopefully there's one thread with data ready to do something useful
- requires more registers so that each thread has its own register space

  (latest NVIDIA P100 has about 3.5M registers in total, 1000 per core)
- this all increases the amount of parallelism an application must have to achieve good performance

  (on a GPU, I'll use 20,000 threads at the same time)

Trends

3) Data movement is often key to performance:

- 200-600 cycle delay in fetching data from main memory
- many applications are bandwidth-limited, not compute limited

  (in double precision, given 200 GFlops and 80 GB/s bandwidth, needs 20 flops/variable to balance computation and communication)
- takes much more energy / time even to move data across a chip than to perform a floating point operation
- often, true cost should be based on how much data is moved, and this is becoming more and more relevant over time
- in some cases, this needs a fundamental re-think about algorithms and their implementation

Trends

4) Increasing integration of networking onto CPUs:

- new low-end Intel Xeon D SoC server chip:
  - 8 cores
  - built-in 2×10Gb/s Ethernet
  - aimed at applications such as web servers
- Intel "Knights Landing" Xeon Phi chip has integrated OmniPath 100 Gb/sec networking
- these moves reduce costs, power consumption, network latency
- they also make all of Intel's competitors extremely nervous

  $\Longrightarrow$ rise of the OpenPOWER consortium
  (IBM, NVIDIA, Mellanox, Xilinx and others)

Trends

5) We're in a period of rapid hardware innovation ...

- Intel Xeon CPUs:
  - up to 24 cores at 2-3 GHz, each with a 256-bit AVX vector unit (and costing up to $7.2k each!)
  - 2.5 MB L3 cache per core – up to 60 MB total
  - up to 300 GB/s L3 cache bandwidth
  - up to 100 GB/s bandwidth to main memory

- Intel Xeon Phi (Knights Landing):
  - standalone or accelerator card like a GPU (about 300W) (costing from $2.5k to $6.5k)
  - 64-72 cores at 1.3-1.5 GHz, each with 0.5MB L2 cache and a 512-bit AVX vector unit, connected by a ring bus
  - 500 GB/s bandwidth to 16GB MCDRAM memory
  - 100 GB/s to main DDR4 memory

Trends

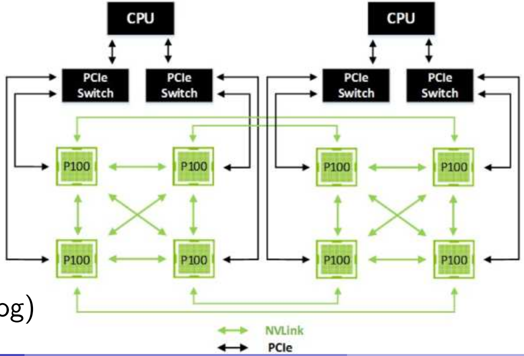
- NVIDIA GPUs:
  - new P100 has 3584 cores running at 1.1-1.5 GHz
  - organised as 56 groups of 64 cores operating (effectively) as vector groups of 32
  - half precision mode for Deep Learning
  - 170/85/42 TFlops in half/single/double precision
  - 720 GB/s bandwidth to 16GB HBM2 memory
  - similar bandwidth (?) to 6MB of L2 cache
  - 4 $\times$ 20 GB/s bi-directional NVlink interconnects to other GPUs or new IBM Power 8 CPU
  - 10 GB/s PCIe bandwidth to/from x86 host

- IBM Power 8 CPU:
  - up to 12 cores at 3 GHz, each with 4 FPUs
  - 115 GB/s bandwidth to memory
  - 2 $\times$ 20GB/s NVlink interconnect to NVIDIA GPUs

Fat or thin nodes?

Suppose there's £5M for a "novel" supercomputer – what do you buy?

1) NVIDIA DGX-1 Deep Learning server
   - 8 × P100 GPUs $\Longrightarrow$ approximately 30k cores in 3U
   - 4 NVlinks per GPU, each 20 GB/s bi-directional
   - 2 × 20-core Intel Xeon E5 CPUs
   - only 40 GB/s aggregate bandwidth to system memory, via PCIe
   - also 40 GB/s aggregate bandwidth to network via 4 × IB EDR ports



(from NVIDIA blog)

Fat or thin nodes?

2) new IBM “Minsky” server

- 4 $\times$ P100 GPUs, each with 4 NVlink connections
- 2 $\times$ IBM Power 8 CPUs, with 2 NVlink connections
- 160–230 GB/s aggregate bandwidth to system memory
- 4 $\times$ IB EDR 100Gb/s ports for networking

3) standard Intel server + GPUs

- 2-4 $\times$ P100 GPUs, each with 16x PCIe v3 connections
- 2 $\times$ 16-20 core Xeon E5 processors
- 20–140 GB/s aggregate bandwidth to system memory
- only 80 lanes of PCIe unless there are PCIe switches
- 1-4 $\times$ IB EDR 100Gb/s networking

4) Intel microserver

- single CPU Xeon-D server
- 25 GB/s bandwidth to system memory
- 2 $\times$ Ethernet 10Gb/s networking integrated into SoC

Fat or thin nodes?

My choice: 1)

- strongly motivated by machine learning applications (and relationship with NVIDIA)
- many of these need just one big GPU, with training data loaded in once and then re-used repeatedly
- some need up to 8 GPUs, so worth paying premium for NVlink
- 8-GPU fat node is also ideal for a lot of smaller molecular dynamics applications
- big caveat: very important that 8×16 GB of HBM2 memory is sufficient to hold all working data – PCIe access to larger system memory is too slow
- external networking is barely sufficient for balanced system

Fat or thin nodes?

STFC Hartree choice: 2)

- multiple motivations:
  - machine learning
  - classic HPC such as CFD
  - strong relationship with IBM – hosts an OpenPOWER centre
- should be excellent for massive datasets which need to be repeatedly streamed in from main system memory (or SSD)
- should give good distributed memory scalability for classic HPC?
- still a bit concerned if working data too big for HBM2
  – maybe Xeon Phi would be better? or need "tiling"?
- more balanced external networking with 10GB/s per GPU

Fat or thin nodes?

Cambridge choice: 3)

- multiple motivations:
  - classic HPC such as CFD, material science, molecular modelling
  - strong relationship with Dell
- should be excellent for Cambridge CFD code – high compute/memory ratio so data fits comfortably inside HBM2
- cheapest solution for applications needing only 1 GPU ?
  (but cheaper PCIe P100 is also 15% slower)
- concerned with performance if working data too big to fit into HBM2
- also concerned about balance of external networking because of insufficient PCIe lanes, unless there are PCIe switches

Fat or thin nodes?

Google, Facebook choice: 4)

http://www.anandtech.com/show/9185/intel-xeon-d-review-performance-per-watt-server-soc-champion

https://code.facebook.com/posts/1711485769063510/
facebook-s-new-front-end-server-design-delivers-on-performance-without-sucking-up-power/

http://www.storagereview.com/facebook_focuses_on_more_efficient_frontend_servers

- good compute & memory I/O performance per watt
- can pack servers very densely (20 per U?) with mini-blades each holding 4 nodes and a mini Ethernet switch
- applications don't need huge networking bandwidth
- from an HPC perspective, networking is very poor – would need integrated 100Gb/s Ethernet to be interesting

Fat or thin nodes?

Software implications?

Big emphasis on reducing data and data movement:

- reducing data movement $\Longrightarrow$ "communication-avoiding algorithms"
- reducing data storage $\Longrightarrow$ recomputation, and "tiling" to fuse multiple loops and eliminate storage of intermediate values

Also, important to achieve vectorisation: in some applications this needs some careful reorganisation of algorithms

FPGAs: their time at last?

A quote:

> *Mike, I've got to tell you about FPGAs! This new technology is going to completely change computing!*
>
> *Ian Page, summer 1992*

I have heard this regularly over the past 25 years – they've been wrong so far, but that may change.

After all, there must be a reason why Intel paid $16.7bn for Altera in 2015.

FPGAs: their time at last?

What are FPGAs?

- Field Programmable Gate Arrays – reconfigurable hardware, i.e. a bunch of logic gates and memory cells which do almost anything
- “compiling” takes up to 12 hours
- for max performance, programmed in VHDL (very difficult)
- for ease-of-use, programmed in OpenCL (similar to CUDA) but at what loss in performance?

FPGAs: their time at last?

My assessment:

- for double precision floating point arithmetic, forget it – custom hardware, as in GPUs, is more efficient
- for integer tasks, and low-precision fixed point arithmetic, FPGAs can be very efficient
- best suited to really important applications where a dedicated team of experts can hand-optimise the code, and then supply the application to others

FPGAs: their time at last?

So why do I think they could become important now?

- for IoT for power efficiency – but maybe simpler to just buy a low-power ARM processor?
- for switches, to handle complex protocols, and offload MPI processing (e.g. global reductions)
- for server chips for on-the-fly encryption and lossless data compression
- for low precision fixed point arithmetic for machine learning (Microsoft is working on this)

High-level frameworks: lessons learned?

Our own research: OP2 / OPS

- Key postdocs:
  - Gihan Mudalige – moving to Warwick as Assistant Prof
  - István Reguly – moved home to Hungary to lectureship at PPKE
- aims for future-proof efficiency on wide variety of modern architectures (GPUs, Xeon Phi, etc)
- based on FORTRAN or C++, but with additional high-level "library routines"
- pre-processing library calls leads to automated code generation (e.g. CUDA for GPUs)
- challenges:
  - "big" enough to cover many applications
  - "small" enough to make implementation / maintenance practical
  - finding secure long-term funding for maintenance

High-level frameworks: lessons learned?

Successes:

- generating code is not difficult; only need to parse library calls
- 2 codes with OP2 (unstructured grids), including 1 at RR
- 3 codes with OPS (block-structured grids), with potential for AWE
- creating framework code no harder than hand-coding of one application code
- lots of other benefits flow naturally from high-level view:
  - simple automated checkpointing
  - automated loop fusion / tiling through "lazy execution"

Biggest difficulty: secure long-term funding for maintenance (though RR paying for OP2, and AWE might pay for OPS)

High-level frameworks: lessons learned?

Alternative: separate customised high-level framework for each application

- write application code generator at a high-level in Python, Matlab, Mathematica
- generate code for low-level implementation on architecture of choice
- can use symbolic differentiation to generate linearised / adjoint code
- some UK examples:
  - FEniCS (Cambridge / Simula / Imperial College)
  - firedrake (Imperial College)
  - SBLI (Southampton)
- Met Office considering the approach for their next-gen weather code

Perhaps a more sustainable approach, but risks re-programming key underlying bits (MPI data exchange, checkpointing, tiling)
– can we put these into supporting libraries?

References

My computing talks and papers:
http://people.maths.ox.ac.uk/gilesm/cuda_slides.html
http://people.maths.ox.ac.uk/gilesm/journals.html

A "trends" talk from a year ago:
http://people.maths.ox.ac.uk/gilesm/talks/accu.pdf

A "fat versus thin" talk from a year ago:
http://people.maths.ox.ac.uk/gilesm/talks/big_little.pdf

A talk on code generation from two years ago:
http://people.maths.ox.ac.uk/gilesm/talks/codegen.pdf

A paper from 2 years ago:
M.B. Giles, I. Reguly. 'Trends in high performance computing for engineering calculations', Proc. Royal Society A, 372(2022), 2014