Trends in HPC: hardware complexity and software challenges

Mike Giles

Oxford University Mathematical Institute
Oxford e-Research Centre

ACCU talk, Oxford

June 30, 2015

Question ?!

How many cores in my Dell M3800 laptop?

Question ?!

How many cores in my Dell M3800 laptop?

- 1 – 10?
- 10 – 100?
- 100 – 1000?

Question ?!

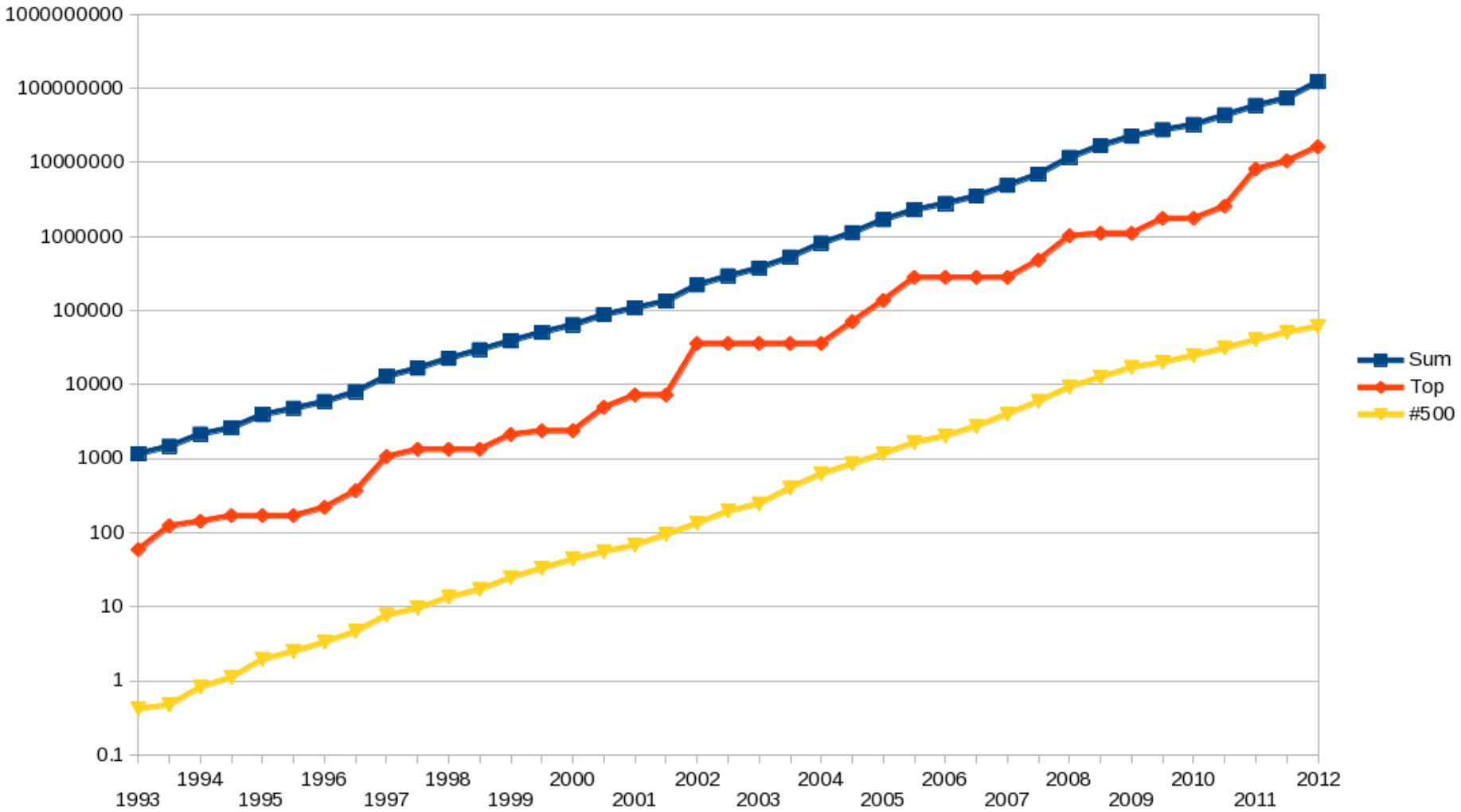
How many cores in my Dell M3800 laptop?

- 1 – 10?
- 10 – 100?
- 100 – 1000?

Answer: 4 cores in Intel Core i7 CPU
+ 384 cores in NVIDIA K1100M GPU!

Peak power consumption: 45W for CPU + 45W for GPU

Top500 supercomputers



Really impressive – 300× more capability in 10 years!

My personal experience

- 1982: CDC Cyber 205 (NASA Langley)
- 1985–89: Alliant FX/8 (MIT)
- 1989–92: Stellar (MIT)
- 1990: Thinking Machines CM5 (UTRC)
- 1987–92: Cray X-MP/Y-MP (NASA Ames, Rolls-Royce)
- 1993–97: IBM SP2 (Oxford)
- 1998–2002: SGI Origin (Oxford)
- 2002 – today: various x86 clusters (Oxford)
- 2007 – today: various GPU systems/clusters (Oxford)
- 2011–15: GPU cluster (Emerald @ RAL)
- 2008 – today: Cray XE6/XC30 (HECToR/ARCHER @ EPCC)
- 2013 – today: Cray XK7 with GPUs (Titan @ ORNL)

Top500 supercomputers

Power requirements are raising the question of whether this rate of improvement is sustainable.

1 supercomputers:

- 1977: Cray-1 used 115 kW
- 1985: Cray-2 used 150-200 kW
- 1993–96: Japanese Numerical Wind Tunnel used 500 kW
- 2005: IBM Blue Gene/L used 716 kW
- 2015: Chinese Tianhe-2 Xeon Phi system uses 17 MW

Top500 supercomputers

There are 4 main types of supercomputer in the Top500 list:

- "traditional" systems based on Intel/AMD CPUs
- systems based on NVIDIA GPUs
- systems based on Intel Xeon Phi accelerators
- IBM Blue Gene systems

plus 1 notable exception:

- K computer, based on Fujitsu Sparcs

Top500 supercomputers

Current top 5 (November 2014):

- Tianhe-2 (National University of Defense Technology, China)
  – 34 PFlops (and 18MW!)
  Custom design with 48,000 57-core Xeon Phi accelerators

- Titan (Oak Ridge National Lab, USA) – 18 PFlops
  Cray XK7 with 18,688 NVIDIA GPUs, each with 2688 cores

- Sequoia (Lawrence Livermore Lab, USA) – 16 PFlops
  IBM Blue Gene/Q with 100,000 16-core PowerPC procs

- K computer (RIKEN, Japan) – 11 PFlops
  90,000 8-core Fujitsu Sparcs

- Mira (Argonne National Lab, USA) – 8 PFlops
  IBM Blue Gene/Q with 50,000 16-core PowerPC procs

Top500 supercomputers

UK top 5 (June 2014):

- #25, ARCHER (EPSRC) – 1.6 PFlops
  Cray XC30 with 9,840 12-core Xeons

- #28, #29 (ECMWF) – 1.6 PFlops each
  Cray XC30 with 6,930 12-core Xeons

- #30, Blue Joule (STFC) – 1.2 PFlops
  IBM Blue Gene/Q with 7,000 16-core PowerPC procs

- #43, Dirac (Edinburgh) – 1 PFlops
  IBM Blue Gene/Q with 6,000 16-core PowerPC proc

- Also of note: #378–381 ("financial institution") – total of 1 PFlops
  IBM iDataPlex with NVIDIA GPUs

Trends

1) Performance is achieved through parallelism:

- Moore's Law continues, circuitry keeps getting smaller
  $\Longrightarrow$ much more circuitry on each chip
- energy efficiency very important, power $\propto$ frequency^3
  $\Longrightarrow$ can't get more performance by higher clock frequency
- hence, performance now comes through massive parallelism
- clock frequency has even come down a little to reduce energy consumption per flop
  - Intel Xeon CPU – 3.0 GHz, with 3.6 GHz single-core boost
  - IBM PowerPC – 1.6 GHz
  - NVIDIA GPU – 1.0 GHz

Trends

2) Vector processing has returned:

- 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 (4 / 8 on most, but 8 / 16 on Intel's Xeon Phi, coming soon to regular Xeons – "Skylake" in 2015/16)
- tricky for algorithms with lots of conditional branching, but there are various algorithmic tricks that can be used

Trends

3) Multithreading is also very important:

- CPU cores are complex, rely on out-of-order execution and branch prediction to maximise single thread performance and avoid stalling when waiting for data from main memory
- many-core chips use simple in-order execution cores, rely instead on multithreading to avoid stalling
- 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 GPU has about 1M registers in total)
- this all increases the amount of parallelism an application must have to achieve good performance

  (on a GPU, I'll use 10,000 threads at the same time)

Trends

4) 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

5) NUMA (Non-Uniform Memory Access) is becoming more common, with significant performance implications:

- CPU divided into sub-units, each with its own memory
- accessing "own" memory is much faster than accessing another's
- for best performance, use distributed-memory programming approach, with separate process on each sub-unit

Trends

6) Maintaining cache coherency across multiple cores is getting tough:

- traditional "snoopy bus" approach is not scalable
- distributed approaches (e.g SGI) are scalable but lack performance (adds to memory latency)
- GPUs gave up on cache coherency – good idea?
- Intel may give up on it in the future, or perhaps maintain it only for some data (or within a sub-unit)?
- at the same time, IBM/NVIDIA are moving towards shared memory between CPUs and GPUs

Trends

7) 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
- future Intel "Knights Landing" Xeon Phi chip is expected to have integrated OmniPath 100 Gb/sec networking
- these moves reduce costs, power consumption, network latency
- they also make all of Intel's competitors extremely nervous

Trends

8) After a quiet period (1995-2005?) we're now in a period of rapid hardware innovation ...

- Intel Xeon CPUs:
  - up to 18 cores at 2-3 GHz, each with an AVX vector unit of length 4/8 (and costing up to $7k each!)
  - up to 45 MB cache (half of chip?)
  - up to 85 GB/s bandwidth to main memory, up to 300 GB/s cache-core bandwidth

- Intel Xeon Phi:
  - accelerator card like a GPU (about 300W)
  - up to 64 cores at about 1 GHz, each with 0.5MB L2 cache and an AVX vector unit of length 8/16, connected by a ring bus
  - 240-320 GB/s bandwidth to graphics memory 600+ GB/s aggregate bandwidth to L2 cache?

Trends

- NVIDIA GPUs:
  - fastest have 2880 cores running at 875 MHz
  - organised as 15 groups of 192 cores operating (effectively) as vector groups of 32
  - 250-300 GB/s bandwidth to 6GB graphics memory
  - 600 GB/s (?) bandwidth to 1.5MB of L2 cache
  - 10 GB/s PCIe bandwidth from graphics card to host
- IBM Blue Gene/Q:
  - PowerPC chips have 16 compute cores running at 1.6 GHz
  - each 4-way multithreaded
  - one extra core for communication and one as a spare
- IBM Power 8 CPU:
  - up to 12 cores per chip at 3 GHz, each with 4 FPUs
  - 300 GB/s bandwidth to memory
  - short vectors (2/4)

Trends

9) . . . and accompanying software diversity

- MPI still dominant for distributed memory computing
  - generally used only for very big applications which need to be spread across multiple systems
  - multiple processes exchanging data by sending messages
  - rather "crude", but straightforward
  - each process allocates its own memory, so works well within a single system in handling NUMA issues

- OpenMP still dominant for shared-memory multithreading
  - programmer uses compiler directives to specify which loops are to be executed in parallel
  - there is also support for task parallelism
  - easy to write parallel code, but without an understanding of caches also easy to get code which performs poorly

Trends

- CUDA and OpenCL for vectorisation on GPUs
  - unlike OpenMP, code is written from the perspective of a single thread
  - both require transfer of data between CPU and GPU, but latest versions enable the programmer to make it implicit rather than explicit

- OpenACC for simpler high-level parallelism for certain applications
  - similar to OpenMP – programmer specifies which loops are to be executed in parallel on the GPU/accelerator
  - quite effective for fairly simple applications

- no good solution yet for exploiting vector capabilities in CPUs and Xeon Phi – though lots of options from Intel
  - low-level vector primitives
  - Cilk Plus
  - OpenMP 4.0
  - auto-vectorisation

- plus we still have language diversity (Fortran, C, C++, Python)

Coming soon

1) "stacked" memory

- Xeon Phi by end of 2015
- NVIDIA Pascal GPU in 2016
- 1TB/s bandwidth, with 16-32 GB?
- will it look like an extra level of cache, or be programmable?
- how soon will it come to regular Xeon CPUs?

2) photonics

- maybe 5 years away?
- integrated onto chips
- optical routers
- faster / more efficient networking

Coming soon

3) new "CORAL" supercomputers

US DoE Oak Ridge and Lawrence Livermore labs:

- due to be delivered in 2017
- "fat" nodes with multiple IBM Power9 CPUs and NVIDIA GPUs
- unified shared memory with 200GB/s NVlink interconnect
- multiple 100Gb/s Infiniband networking connections between nodes

DoE Argonne lab:

- prototype in 2016, full system in 2017/8
- based on new "Knights Landing" Xeon Phi as a primary processor, not a co-processor
- 16-32GB stacked memory
- integrated OmniPath 100 Gb/s networking

Application challenge

Short-term:

- experiment with different emerging architectures
- develop implementation techniques to extract the most performance
- determine which architectures are best for different applications

Longer-term challenge:

- application developers don't want to constantly re-write their codes for the latest hardware and software
- ideal solution is for them to specify **what** they want a higher level, leaving it to parallel computing experts to decide **how** it's done

Good news – this is now being addressed by computer science / scientific computing researchers, and funding agencies

UK buzzword: "future-proof" software development

Research directions

- DSLs (domain specific languages)
  - specialised high-level languages for certain application domains
  - a more general alternative to numerical libraries
  - if successful, the work of a few computing experts supports a large number of applications
  - difficulty is in defining a high-level language (or framework) which addresses a sufficiently large and important class of applications, and getting funding to sustain development
  - can involve automated code generation and run-time compilation

- auto-tuning
  - automated optimisation of parameterised code
  - used in many software libraries in the past (e.g. FFTW, BLAS) but now being used more widely
  - very helpful with GPUs where there are lots of parameters to be set (number of thread blocks, number of threads in each thread block) and the tradeoffs are complex, not at all obvious

Research directions

- communication-avoiding algorithms
  - evolved out of growing focus on communication costs
  - models the execution cost in terms of data transfer between main memory and cache (ignoring cost of arithmetic operations)
  - then looks for implementation strategies to minimise this
  - important research direction for the future

- tiling
  - evolved out of "blocking" technique in dense linear algebra
  - improves re-use of data in the cache by overlapping different loops / phases of computation
  - complex for application developers to implement – really needs a high-level approach with sophisticated tools (e.g. compiler techniques, lazy execution)
  - a good example of the need for partnership between application developers and computer scientists

What does it all mean for developers?

Structured grids:

- strongly consider using GPUs (1 GPU $\equiv$ 6 regular Xeons?) unless computations involve lots of data at each grid point
- consider a high-level framework for future-proof programming
- alternatively, use OpenACC (or maybe in the future OpenMP 4.x) for portable software

Unstructured grids:

- much harder to get good performance on GPUs
- consider using libraries (e.g. AMGx)
- if sticking to CPUs, think about vectorisation – very important for the long-term

What does it all mean for developers?

General:

- is application bandwidth-limited? if so, re-design to reduce data communication, for example by fusing loops, even at the expense of increased computation
- auto-tuning could be very helpful
- make sure that NUMA issues are being addressed properly on CPUs
- for big applications, make sure you use parallel file I/O (e.g. HDF5)
- monitor hardware / software developments — there's a lot going on

Conclusions

- computing power continues to increase . . .
- . . . but the complexity is increasing too
- application developers need to work with computer scientists and scientific computing experts to
  - exploit the potential of new architectures
  - do so without huge programming efforts
- the good news is that there is increasing agreement that this is needed, and there's funding to support it
- no clear path ahead – need lots of research groups trying different approaches to find out what works

ASEArch

ASEArch (Algorithms and Software for Emerging Architectures) was an EPSRC-funded project to assist other CCPs (Collaborative Computational Projects) with advice, and software help, on using GPUs and other accelerators.

- led by myself and Simon McIntosh-Smith (Bristol) with support from STFC
- `www.oerc.ox.ac.uk/projects/asearch`
  website has lots of information on hardware and software developments on various platforms
- `people.maths.ox.ac.uk/gilesm/files/phil_trans14.pdf`
  review article on hardware / software trends and what they mean for computational engineering

Big Data

How could I not mention Big Data!

Answer: not my research area, and I don't feel qualified to comment (although I am involved in planning for new Alan Turing Institute)

However, my unqualified views are:

- increasingly important, not just hype
- high capacity, high bandwidth storage essential, with tight coupling to compute capability
- GPUs being used very extensively for machine learning (Google, Baidu, Facebook, Microsoft)