Trends in HPC
(hardware complexity and software challenges)

Mike Giles

Oxford e-Research Centre
Mathematical Institute

MIT seminar

March 13th, 2013

Outline

- more power 
- more complexity 
- more help 

Pop Quiz

How many cores are in my Apple MacBook Pro?

Pop Quiz

How many cores are in my Apple MacBook Pro?

- 1 – 10?
- 10 – 100?
- 100 – 1000?

Pop Quiz

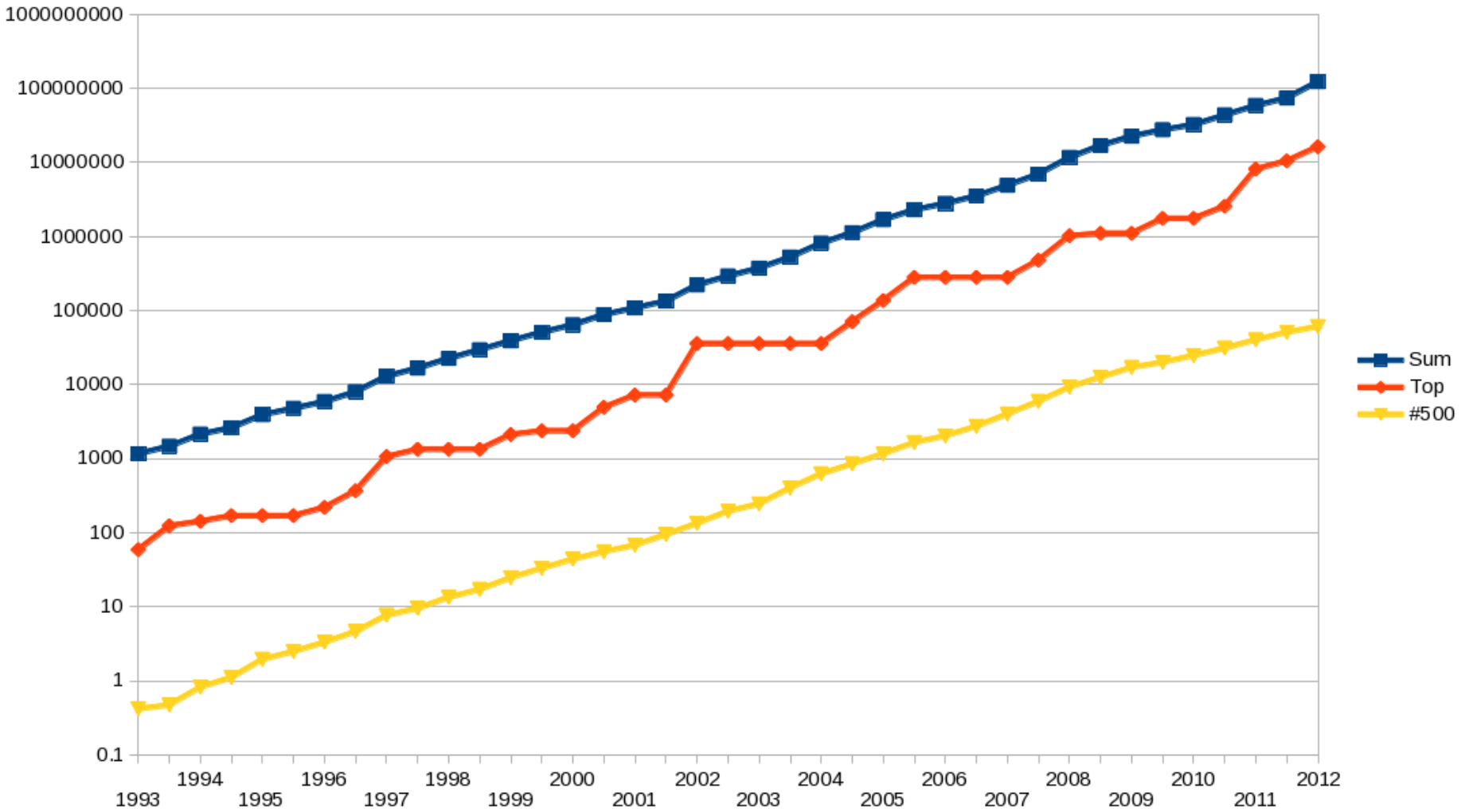
How many cores are in my Apple MacBook Pro?

- 1 – 10?
- 10 – 100?
- 100 – 1000?

Answer: 4 cores + 16 graphics "units" in Intel Core i7 CPU
+ 384 cores in NVIDIA GT650M GPU!

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

Top500 supercomputers



Really impressive – 300× more capability in 10 years!

Top500 supercomputers

There are 3 main types of supercomputer in the Top500 list:

- "traditional" systems based on Intel/AMD CPUs
- GPU-based systems based on NVIDIA GPUs
- IBM Blue Gene systems

plus a couple of notable exceptions:

- K computer, based on Fujitsu Sparcs
- Stampede, based on Intel Xeon Phi as well as NVIDIA GPUs

Top500 supercomputers

Current top 5:

- 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
- JUQUEEN (Juelich, Germany) – 4 PFlops
  IBM Blue Gene/Q with 25,000 16-core PowerPC procs

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

- Intel Xeon CPUs:
  - up to 10 cores at 2-3 GHz, each with an AVX vector unit of length 4/8
  - up to 30 MB cache (half of chip?)
  - up to 75 GB/s bandwidth to main memory,
    up to 300 GB/s cache-core bandwidth

- Intel Xeon Phi:
  - accelerator card like a GPU (about 250W?)
  - 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 2688 cores running at 700-750 MHz
  - organised as 14 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
- ARM:
  - not currently involved in HPC
  - moving into servers based on energy efficiency design strength
  - could become significant over next 10 years

Trends

6) . . . and accompanying software diversity

- MPI still dominant for distributed memory computing
- OpenMP still useful for shared-memory multithreading
- CUDA and OpenCL for vectorisation on GPUs
- OpenACC for simpler high-level parallelism (but limited applicability?)
- 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
  - OpenCL
  - auto-vectorization
- plus we still have language diversity (Fortran, C, C++, Python)

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

Key buzzwords

- High Productivity Computing (DARPA)

  http://www.darpa.mil/Our_Work/MTO/Programs/High_Productivity_Computing_Systems_(HPCS).aspx

  – motivated by concerns over effort required for HPC application development, looking for something better

- Co-design (DoE)

  http://science.energy.gov/ascr/research/scidac/co-design/

  – application developers and computer scientists working together to develop hardware / software / applications which work together

- "Future-proof" application development

  – trying to avoid being tied to any one architecture

Key technical buzzwords

- 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

Key technical buzzwords

- communication-avoiding algorithms (Jim Demmel)
  - 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

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