Opportunities for Parallelism

Dr. Michael K. Bane

HIGH END COMPUTE

Questions

1. What do you understand by "parallelism"

2. How/where is parallelism in computers?

Parallel / parallelism

- Concurrent / concurrency
- Many things ("tasks", "operations", "calculations",…) at once
- Run forever with fixed separate (parallel lines)
- Co-existing (parallel universe)
- Equivalent (the parallel circles of constant latitude)
- Electrical circuits

Parallel Programming

- Running one or more codes concurrently in order to
  - reduce the time to solution (divide work by more cores)
  - model harder cases (scale up problem with increasing core count))
  - model larger domains (more memory)
  - use models at higher resolutions (more memory)
  - reduce the energy to solution

- For most of these we will need to
  - divide the work between cores
  - divide the data between cores

Approaches to parallelism

- Hardware
  - Multiple-core processors – clusters – clusters of clusters
  - Many core accelerators & co-processors
  - Vectorisation & ILP (intra core)
- Software
  - Use of libraries (eg MKL)
    - Math Kernel Lib (Intel) is threaded ie parallel (see Exercise001)
  - Compiler
  - Programming Languages: C++, Java, Haskell, occam
  - Extensions to languages
    - Directives based: OpenMP, OpenACC
    - Libraries based: MPI, OpenCL

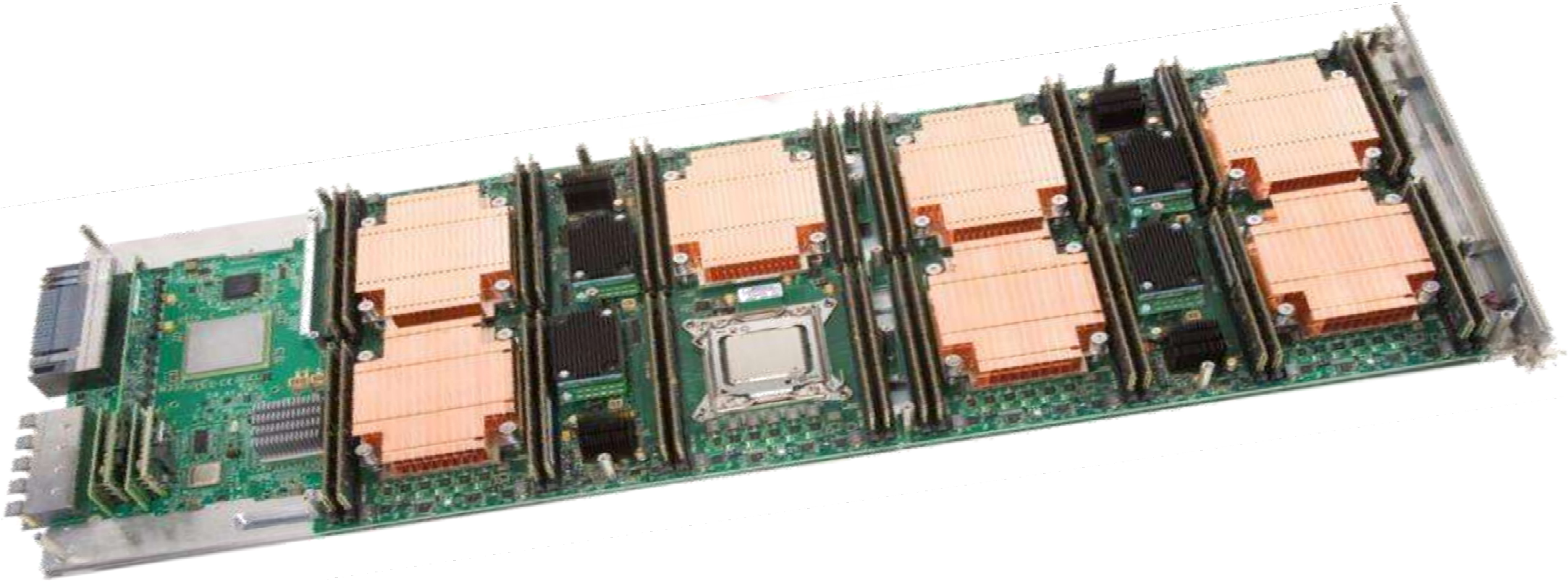
Questions

1. Where do you see parallelism in the natural world?

2. What prevents us having parallel simulations of the parallelism observed in the natural world?

Possible Solutions

1. **Light Rays**
   - Stationary pumpkin: Rays are independent so can model each in parallel
   - Moving pumpkin: image per position is independent, so can also parallelise over time
2. **Paint by numbers**
   1. task parallelism (each doing one colour)
   2. Limits & load imbalance depending on number of colours/pens/people and on number of areas to be coloured in
3. **Jigsaw**
   1. Divide by type (eg sea/beach/dunes) -> task parallelism; could also do edges .v. internal (but load imbalance since former is O(N) and latter is O(N^2)
   2. Iterating over take a piece and try every place it fits -> monte carlo
   3. More pieces -> more work (and more comms)
4. **Coloured balls**
   1. Could scale but there may be overhead of working out who to get which colour
   2. Alternative sorting: everybody sorts a local pile and then merge local piles to give global sort
5. **Find next prime number**
   1. Checking primeness can be done in parallel; checking a region for a prime could be done in parallel
   2. Given there are screen savers to find next prime, there must be reasonable parallelism
6. **Fibonnaci**
   1. Ideally know the analytical solution -> many great advances in computational ability are due to ALGORITHMIC IMPROVEMENT rather than faster/parallel computers
7. SETI@home, Folding@home



ARCHICTECTURE

What are the 2 main memory models?

- Recap: questions from SL2
- Diagram on whiteboard

SHARED MEMORY

- Memory on chip
  - Faster access
  - Limited to that memory
  - … and to those nodes

- Programming typically OpenMP (or another threaded model)
  - Directives based
  - Incremental changes
  - Portable to single core / non-OpenMP
    - Single code base ☺

DISTRIBUTED MEMORY

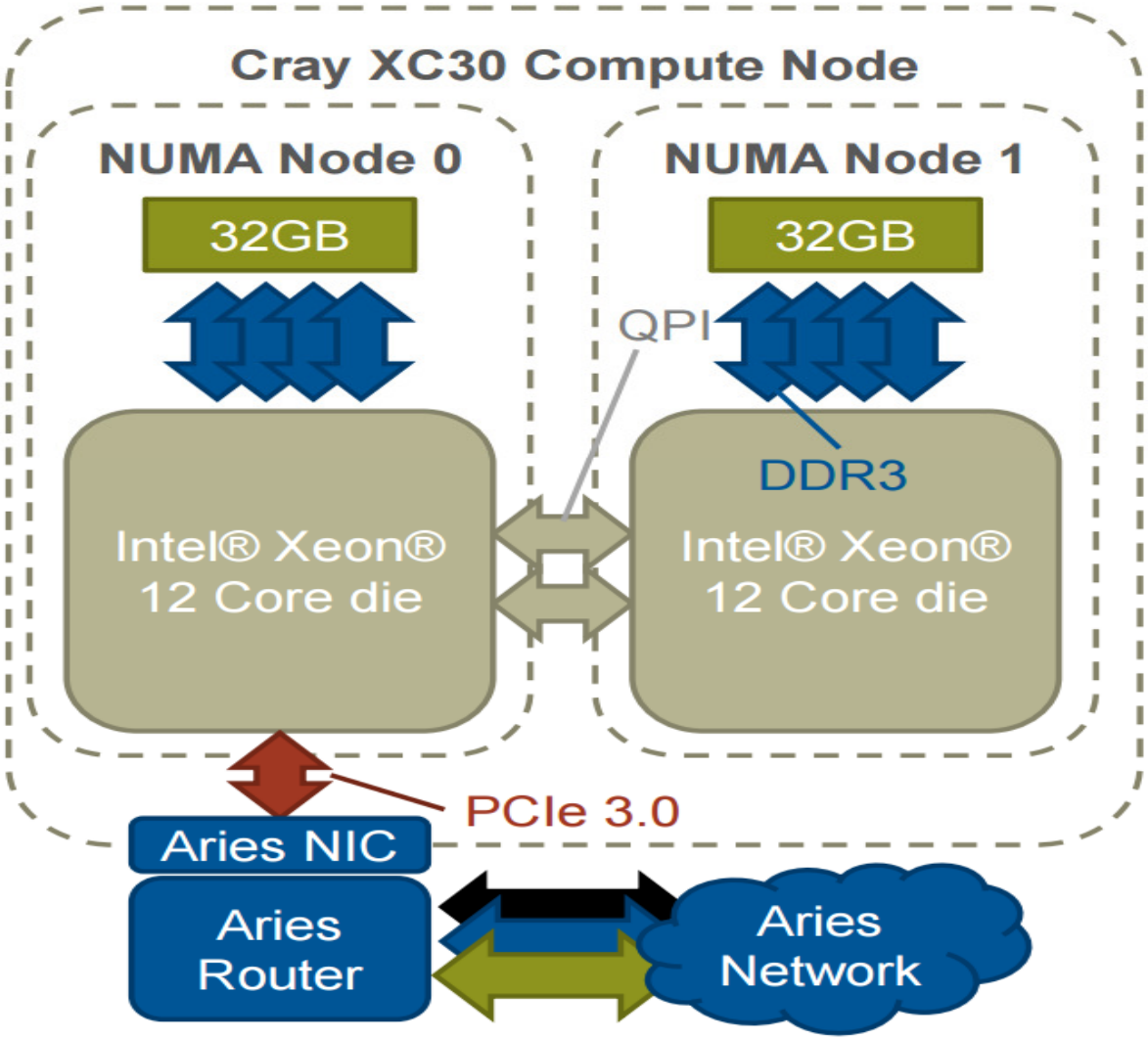
- Access memory of another node
  - Latency & bandwidth issues
  - IB .v. gigE
  - Expandable (memory & nodes)

- Programming 99% always MPI
  - **M**essage **P**assing **I**nterface
  - Library calls
  - More intrusive
  - Different MPI libs / implementations
  - Non-portable to non-MPI (without effort)

Examples for OpenMP

	Typical Number of cores addressing Shared Memory	Shared Memory size /GB	Typical Shared Mem programming paradigm	Directives supported
Desktop PC	2-4 (HT not good idea)	4-32	OpenMP	
Workstation	8-32	32-128	OpenMP	
Node of Archer	24	64 (some 128)	OpenMP	
Cavium 2x ThunderX	96 (2x 48c)		OpenMP	
Intel Xeon Phi	60-64 cores (HT works!)		OpenMP	
NVIDIA GP100 (5.3TF DP)	60 Streaming Multiprocessors (SMs) *each* of 64 "CUDA cores"	64 KB per SM	CUDA	OpenMP 4 or higher OpenACC
AMD GPU			OpenCL	
SGI UV3000	4,096 threads on 256 sockets	64 TB (yes TB!)	OpenMP	

Cray XC30 Intel® Xeon® Compute Node



The XC30 Compute node features:

- **2 x Intel® Xeon® Sockets/die**
  - 12 core Ivybridge
  - QPI interconnect
  - Forms 2 NUMA nodes
- **8 x 1833MHz DDR3**
  - 8 GB per Channel
  - 64 GB total
- **1 x Aries NIC**
  - Connects to shared Aries router and wider network
  - PCI-e 3.0

http://archer.ac.uk/about-archer/gallery/xe6-xc30-overview.pdf

- Programming usually a mix of
  - MPI between nodes (or NUMA regions)
  - OpenMP on a node (or for given NUMA region)
- Ability to use directives (OpenMP) programming to "offload" to GPUs and Xeon Phi

- Exciting times
  - New memory tech (MCDRAM/XPhi, stacked memory/GP100)
  - Mixing accelerators/GPUs and CPUs
    - and FPGAs

Next…

- Focus on the OpenMP programming
- Can summarise very succinctly

- But first, any FORTRAN codes to get on to Archer?

Next…

- Focus on the OpenMP programming
- Can summarise very succinctly

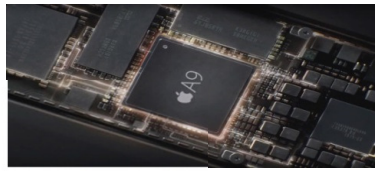
`!$ OMP` *`directive`*

- But first, any FORTRAN codes to get on to Archer?

TODAY'S HARDWARE

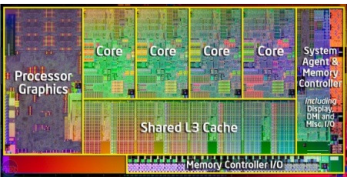
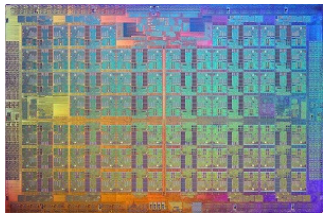
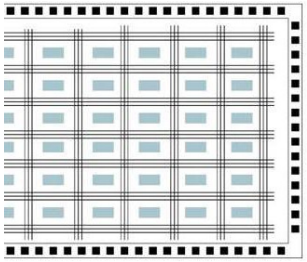
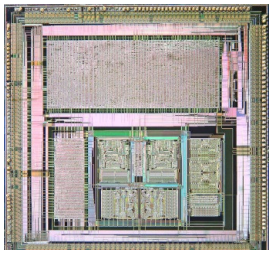
		Cost	Memory	Energy Requirements	FLOPS per second
1948	“Baby” computer, Manchester				1.1 K
1985	Cray 2	$16M			2 G
2013	ARCHER (Cray XC30). 118K cores (#41 in Top500)	£43M	64 GB/node	~2 MW 641 MFLOPS/W	1.6 P
2015	iPhone 6S. ARM / Apple A9. 2 cores	£500	2 GB		4.9 G
2015	Raspberry Pi 2B. ARMv7. 4 cores	£30	1 GB		50 M per core 200 M per RPi
2013-2015	Tianhe-2 (#1 of Top500). 3.1M cores		1 PB	17.8 MW	33.86 P
2015	Shoubu, RIKEN (#1 of Gren500). 1.2M cores		82 TB	50.32 KW 7 GFLOPs/Watt	606 T
2016	Sunway Tiahu. 10.6 M cores (new Chinese chip/interconnect etc)	$270M (inc R&D to design chips etc)	1.3 PB	15.4 MW 6 GLOPS/Watt	125 P







Images: cs.man.ac.uk, CW, appleapple.top, top500/JD, RIKEN

	**CPU**	Intel, AMD, ARM (as IP)	1 to maybe 64 cores, running at 2 to 3 GHz	Powerful cores, out of order, look ahead. Good for general purpose and generally good	1-2 sockets direct on the motherboard
	**GPU**	NVIDIA, AMD	15 to 56 "streaming multiprocessors" (SMs), each with 64-128 "CUDA Cores". Base freq about 1 GHz	SMs are good for high throughput of vector arithmetic	AMD produced "fused" CPU & GPU. Until 2016, NV cards situated at far end of PCI-e bus. In 2016, NV working with IBM for on-board solution using "NVlink"
	**Xeon Phi**	Intel	60-70 cores	Low grunt but general purpose cores	KNC was PCI-e but KNL (2016) is standalone
	**FPGA**	Altera (Intel), Xilinix	Fabric to design own layout – and reconfigurable	Can use Verilog or VHDL to map. MATLAB can also be used. Maxeler uses Java	Focus needs to be on the data flow
	**ASIC**		Anton-2 uses custom ASIC for MD calcs. Very fast but not necessarily low power		If you're designing ASIC you needn't be on this course!

HIGH THROUGHPUT COMPUTING

Many ways to get a job done fast

- So far
  - Taking one code, using parallelism to get that simulation done quicker
- But what about likes of Monte Carlo, parameter sweeps etc
  - Run one "standalone" task, a huge number of times
  - ie lots of parallelism!
- Could program as one code or look at how to run many copies

Options

- Run as one code
  - Pro: all in one place, easier for post analysis
  - Con: will be seen as one big job by scheduler

- Submit many jobs to the batch system
  - Pro: scheduler can use "back fill" to get small(er) jobs through quicker (including likes of Condor)
  - Pro: can run 50K tasks (say) without needing 50K cores
  - Pro: load imbalance irrelevant (scheduler considers others' jobs)
  - Con: need to put controlling logic at the scheduler level

How to do H*T*C

- Use "job arrays"
  eg on Archer, additional PBS flag -J 0-999

  Launches 1000 tasks, each with a $PBS_ARRAY_INDEX

  Use this env var to set up parameters eg

```
N=(1,2,3,4,6,8,9,10,12,14,15,16,18,20,21,22,24)
let elem=${PBS_ARRAY_INDEX}
./a.out ${N[$elem]}
```

- Condor – use of "spare" cycles eg on PCs
  Condor/DAGMAN: variables to control tasks and similar use of arrays and indices to select local task idents from global set

PARALLELISM IN OTHER LANGUAGES ETC

OpenMP

- Extension for FORTRAN, C, C++
- Bindings for
  - Java (or just use Java threads!)
  - Python eg Cython
  - (and many more)

Parallel Programming Languages

- UPC, CHAPEL
- Hadoop, Spark
- Julia
- CUDA, OpenCL
- Co-Array FORTRAN, Java
- Haskell – functional programming, native support for parallelism (and concurrency)
- Erlang,
- VHDL, Verilog

Parallel Programming Languages

- UPC
- CHAPEL
- Co-Array FORTRAN

- Haskell – functional programming, native support for parallelism (and concurrency)
  - Parallelism: "speeding up a pure computation (by) using multiple processors"
  - Concurrency: "multiple threads of control that execute 'at the same time'"

MATLAB

- Use of PCT
  - to parallel for loops: parfor (beware granularity)
  - To push to GPUs: GPUArray
  - Clusters: Distributed Computing Server (infra)
- OPTIMISATIONS
  - Compile it (mcc) and run the compiled exec in a job array (etc)
  - Start using C
  - Compile down to VHDL for FPGA