National HPC Facilities at EPCC

Exploiting Massively Parallel Architectures for Scientific Simulation

Dr Andy Turner, EPCC
a.turner@epcc.ed.ac.uk

Outline

- HPC architecture state of play and trends
- National Services in Edinburgh:
  - HECToR
  - ARCHER
  - DiRAC Bluegene/Q
- HPC Software Challenges
- Final Thoughts

What is EPCC?

- A leading European centre of novel and high-performance computing expertise based at the University of Edinburgh
- Formed in 1990 and involved in:
  - Research
  - Collaboration
  - Training
  - Service provision
  - Technology transfer
- Around 70 staff
- Provide national services on behalf of RCUK

epcc | THE UNIVERSITY OF EDINBURGH

Concurrent Programming and HPC

epcc

THE UNIVERSITY OF EDINBURGH

HPC Architectures

State of play and trends

HPC == Parallel Computing

- Scientific simulation and modelling drive the need for greater computing power.
- Single systems can not be made that had enough resource for the simulations needed.
  - Making faster single chip is difficult due to both physical limitations and cost.
  - Adding more memory to single chip is expensive and leads to complexity.
- Solution: parallel computing – divide up the work among numerous linked systems.

Processors

- Not many HPC processors any more
  - Use components designed for server and games industries
  - Exceptions: IBM Power, IBM BlueGene
- Trends:
  - More concurrency – higher core counts per socket
  - Longer SIMD – vector-like instructions
  - Gating – to reduce power usage
  - Stabilisation of clock speeds – no increase but the downwards trend has slowed (at least for multicore processors)
- Splitting into a number of classes
  - Complex multicore (2-3 GHz, Intel Xeon, IBM Power, AMD Opeteron)
  - Simpler manycore (1-2 GHz, Intel Xeon Phi IBM BG)
  - Heterogeneous processing (AMD Fusion, NVIDIA Denver)

Accelerators

- NVIDIA GPGPU and Intel Xeon Phi
  - Even more FP SIMD capability than CPUs
  - Simplified memory architectures (no NUMA, limited cache)
  - Simplified logic – limited support for branching, etc.
- Usually linked to CPU via PCI express
  - Separate memory spaces – makes it difficult to get high performance
- Some systems support socket-mounting of accelerators
  - Move from multi-core to many-core
- Trend for convergence of CPU and accelerator technologies

Memory

- Amount of memory per processing element is generally reducing
  - Memory is expensive both in terms of cost and power
  - Often in a NUMA setup which can cause difficulties in extracting best performance
- Trends:
  - Memory performance is increasing: reduction in latency, increase in bandwidth…
  - …but not as quickly as increases in concurrency
  - Accelerators are leading to a simplification of memory architecture but adding more constraints on realising performance

IO

- Local disk is being abandoned in favour of global, parallel filesystems
- Often designed for high performance writing of a small number of large files – other modes do not give best performance
- Trend is to larger parallel filesystems with more aggregate bandwidth
- Moving data is now one of the most expensive operations
  - Lot of interest in mobile compute – bring the compute to the data
  - HPC systems must be collocated with long-term data storage

Interconnects

- Various interconnect technologies are converging on common hardware performance
  - Not much difference between commodity (Infiniband) and proprietary (Cray, IBM) hardware
  - Differences now come in the topologies, software stack, and support for alternative parallel models
- Trends:
  - Moving network interfaces directly on to silicon
  - Using spare cores, hardware threads to support/control communications (core specialisation)

National Services in Edinburgh

HECToR

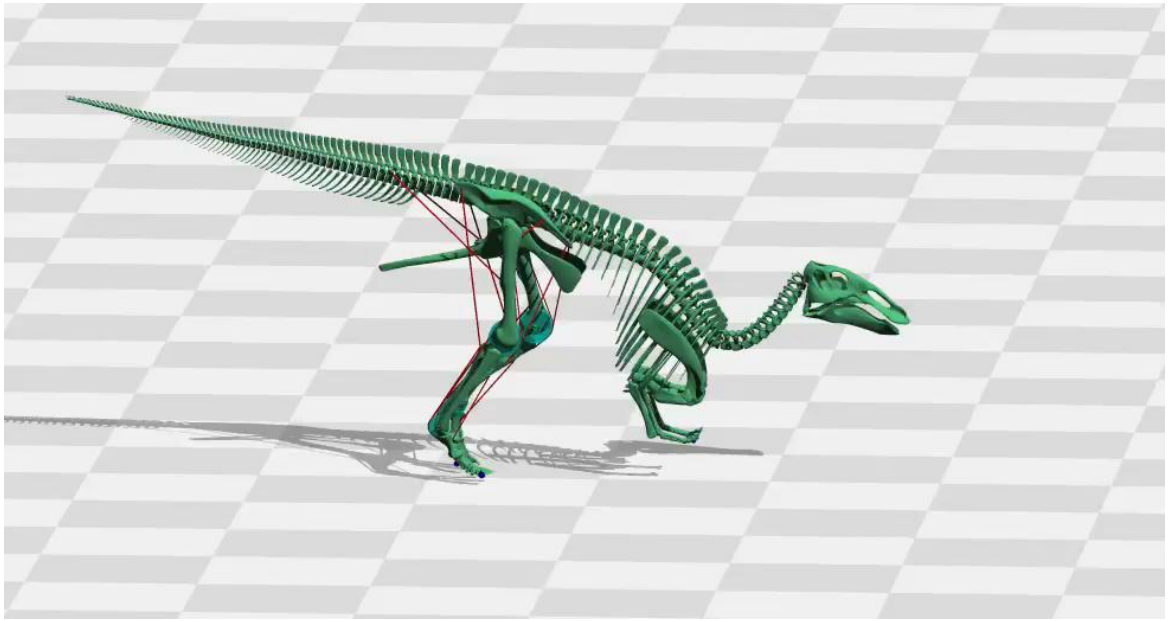
EPSRC

NATURAL ENVIRONMENT RESEARCH COUNCIL

BBSRC
bioscience for the future

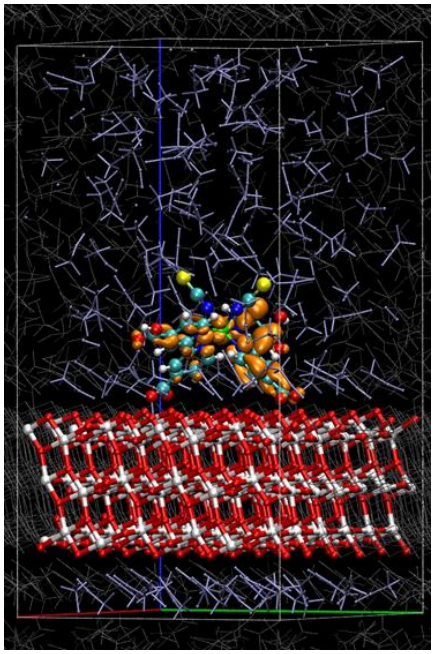
epcc

THE UNIVERSITY OF EDINBURGH

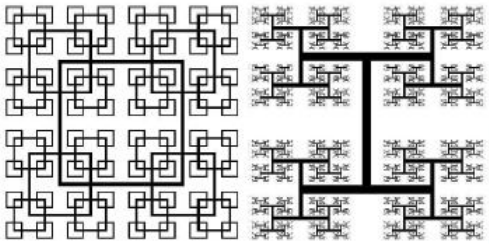


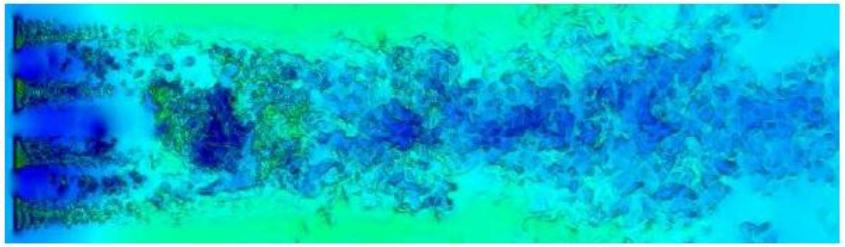
Modelling dinosaur gaits
Dr Bill Sellers, University of Manchester

Dye-sensitised solar cells
F. Schiffmann and J. VandeVondele
University of Zurich



Fractal-based models of turbulent flows
Christos Vassilicos & Sylvain Laizet,
Imperial College

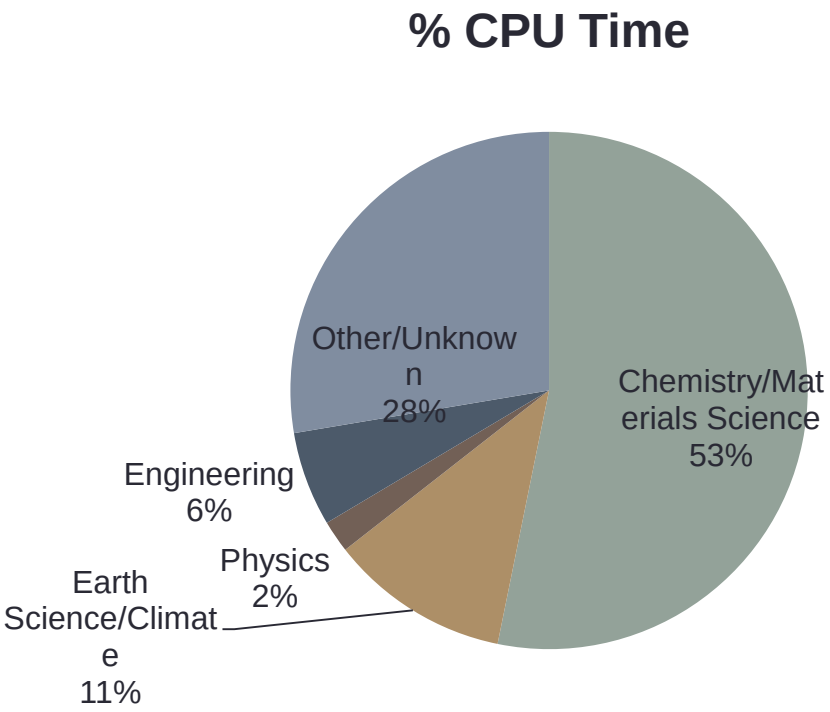


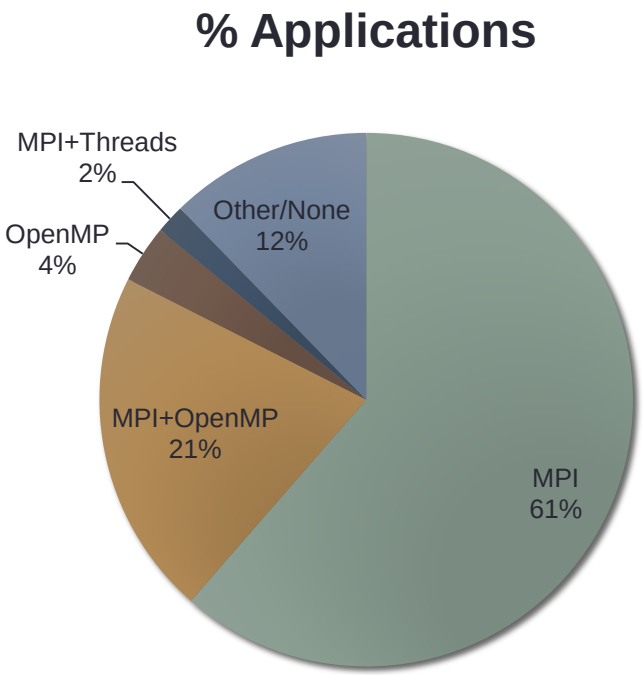


epcc

THE UNIVERSITY OF EDINBURGH

HECToR Applications





epcc

THE UNIVERSITY OF EDINBURGH

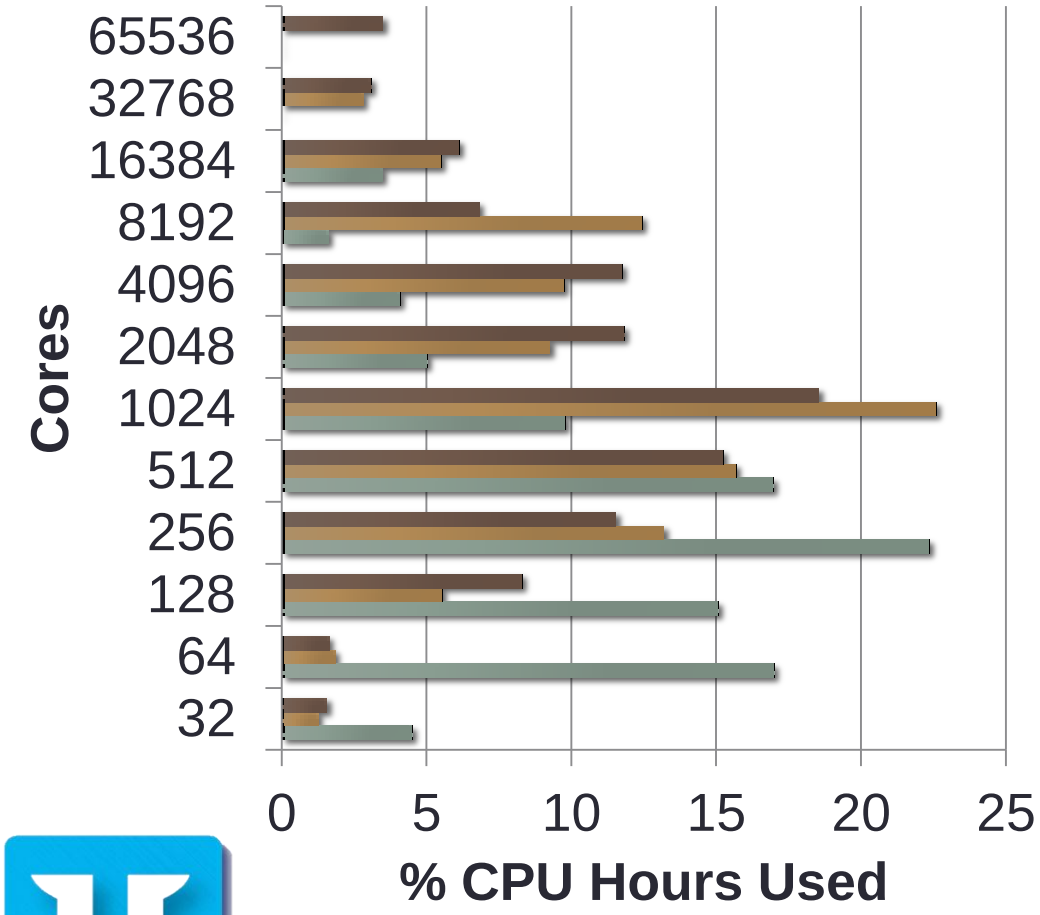
HECToR Changes

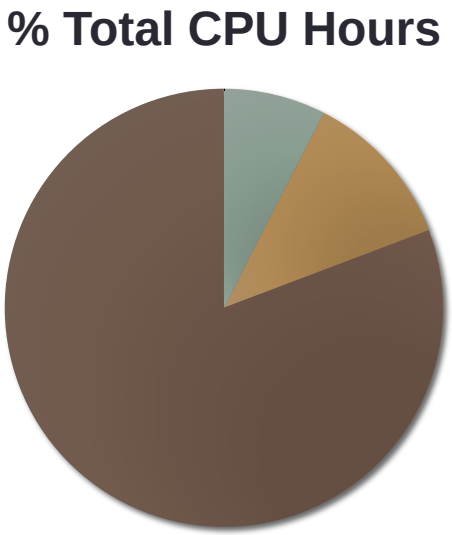
	Phase 1 ('07-'09)	Phase 2a ('09-'10)	Phase 2b ('10-'11)	Phase 3 ('11-now)
Cabinets	60	60	20	30
Cores	11,328	22,656	44,544	90,112
Clock Speed	2.8 GHz	2.3 GHz	2.1 GHz	2.3 GHz
Cores/Node	2	4	24	32
Memory/Node	6 GB (3 GB/core)	8 GB (2 GB/core)	32 GB (1.3 GB/core)	32 GB (1 GB/core)
Interconnect	6 µs 2 GB/s	6 µs 2 GB/s	1 µs 5 GB/s	1 µs 5 GB/s

epcc

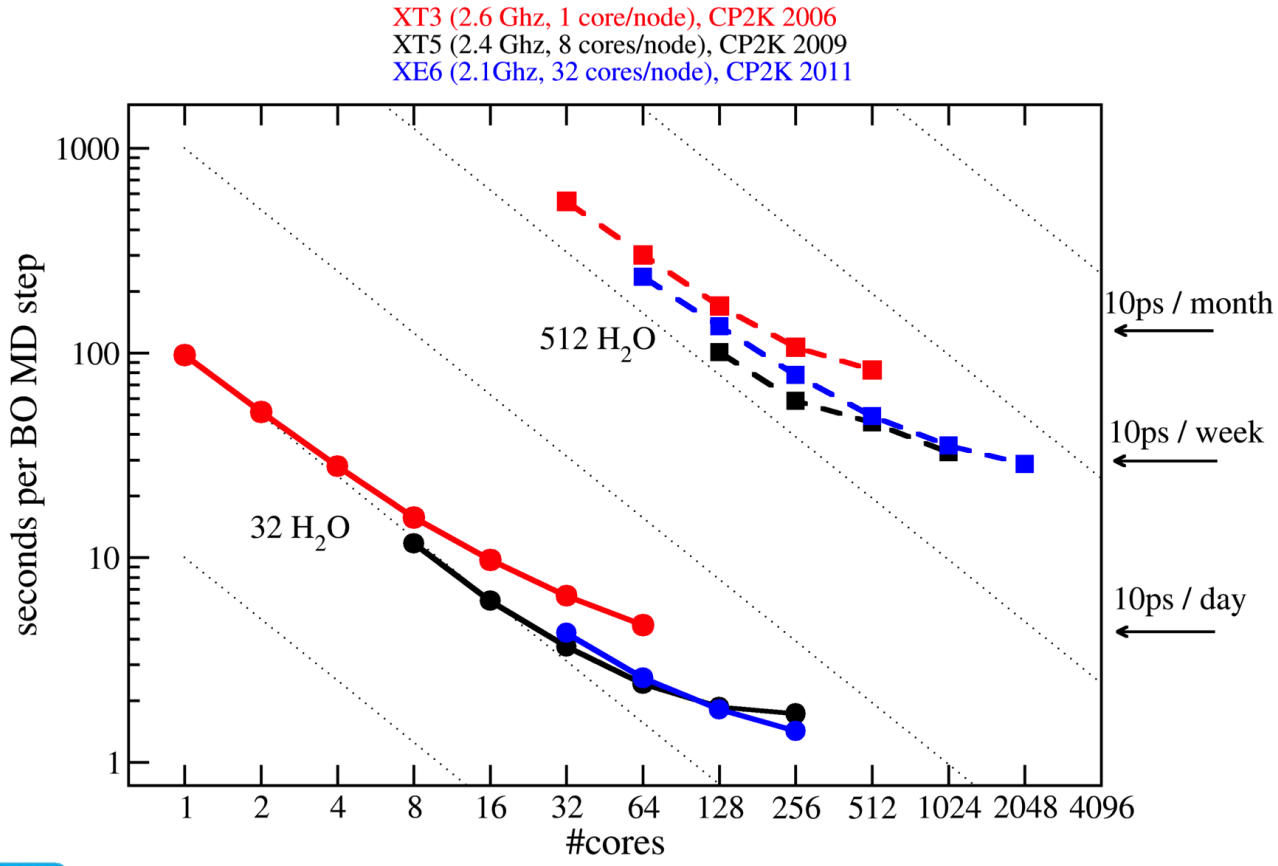
THE UNIVERSITY OF EDINBURGH

HECToR Jobs


65536
32768
16384
8192
4096
2048
1024
512
256
128
64
32
Cores
0
5
10
15
20
25
% CPU Hours Used


% Total CPU Hours
Phase 3
Phase 2b
Phase 2a

Example: CP2K Development



J. VandeVondele, ETHZ

EPSRC

NATURAL ENVIRONMENT RESEARCH COUNCIL

epcc

THE UNIVERSITY OF EDINBURGH

DiRAC BlueGene/Q

Science & Technology Facilities Council

epcc

The University of Edinburgh

BlueGene/Q: Co-design

- 18 core, 1.6 GHz BGQ Chip, quad DP SIMD instructions, 4 hardware threads per core
- Low-latency, high-bandwidth interconnect: 5D torus
- Designed in collaboration with Quantum Chromodynamics researchers
  - Runs QCD applications extremely well…
  - …but it can be difficult to get good performance for other applications
- Non-commodity processors actually cause a problem here:
  - Compilers are not as well developed and key to getting performance is being able to generate SIMD instructions

HPC Software Development

Challenges for now and the future

Exposing Parallelism

- To be able to exploit modern HPC systems you need to be able to expose all levels of parallelism in your code:
  - SIMD/vector Instructions
  - Multicore (shared-memory)
  - Distributed memory
- Data decomposition over distributed memory is the really hard part
  - Compilers do a good job of exploiting SIMD instructions and shared memory
  - Very hard for compilers to do the high-level analysis required so this is done by hand

Parallel Programming Models

- MPI is still dominant model
  - Performance is not ideal but it is very flexible – almost any combination of task and/or data parallelism can be implemented
  - Very portable – it is well supported on all HPC machines
- Hybrid MPI+OpenMP has proven to be a useful model to get performance but introduces a lot of complexity
  - Which thread passes messages?
  - Process/thread placement becomes very important
- Trends:
  - Domain-specific languages
  - Autotuning
  - Single-sided communications

Legacy Code

- Some HPC codes are older than me - there is a lot of time and expertise invested.
  - Should these be rewritten from scratch?
  - Can we improve the fundamental dependencies (*e.g.* MPI, PETSc, ScaLAPACK) to allow them to scale on modern/future architectures?
  - How can you encourage communities to migrate to new codes?
- The parallel programming model and decomposition is often implicitly assumed throughout the code
  - Difficult to refactor or add additional levels of parallelism
- Much effort spent in new parallel models but single biggest gain would be MPI improvement

Other Issues

- Memory Efficiency:
  - Amount of memory per core is decreasing but often want to run more complex simulations
  - Need to use multithreading to increase memory available without wasting compute resources
- Accelerators:
  - Still need hand-crafted code to exploit them efficiently
  - How can we make these resources generally useful
- Parallel IO:
  - 10,000 processes reading/writing at once?
  - How can you checkpoint PB of data?

Final Thoughts

What will future systems look like?

	2013	2017	2020
System Perf.	34 PFlops	100-200 PFlops	1 EFlops
Memory	1 PB	5 PB	10 PB
Node Perf.	200 GFlops	400 GFlops	1-10 TFlops
Concurrency	64	O(300)	O(1000)
Interconnect BW	40 GB/s	100 GB/s	200-400 GB/s
Nodes	100,000	500,000	O(Million)
I/O	2 TB/s	10 TB/s	20 TB/s
MTTI	Days	Days	O(1 Day)
Power	20 MW	20 MW	20 MW

Summary

- Advances in hardware are outstripping ability of software to keep up
  - Hardware currently talking about exascale…
  - …struggling to get most codes to tera-/peta-scale
- All about parallelism
  - High level parallelism is still constructed by hand. Efforts to expose this to the compiler underway.
- Need to be memory efficient
- Think carefully about data distribution
- Is legacy code working or do you need to start over?

Any questions?

a.turner@epcc.ed.ac.uk