Performance Analysis of a Cardiac Simulation Code Using IPM

Peter Strazdins*
Computer Systems Group,
Research School of Computer Science,

Markus Hegland,
Mathematical Sciences Institute,

The Australian National University

Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, 14 Nov 2011

(slides available from http://cs.anu.edu.au/~Peter.Strazdins/seminars)

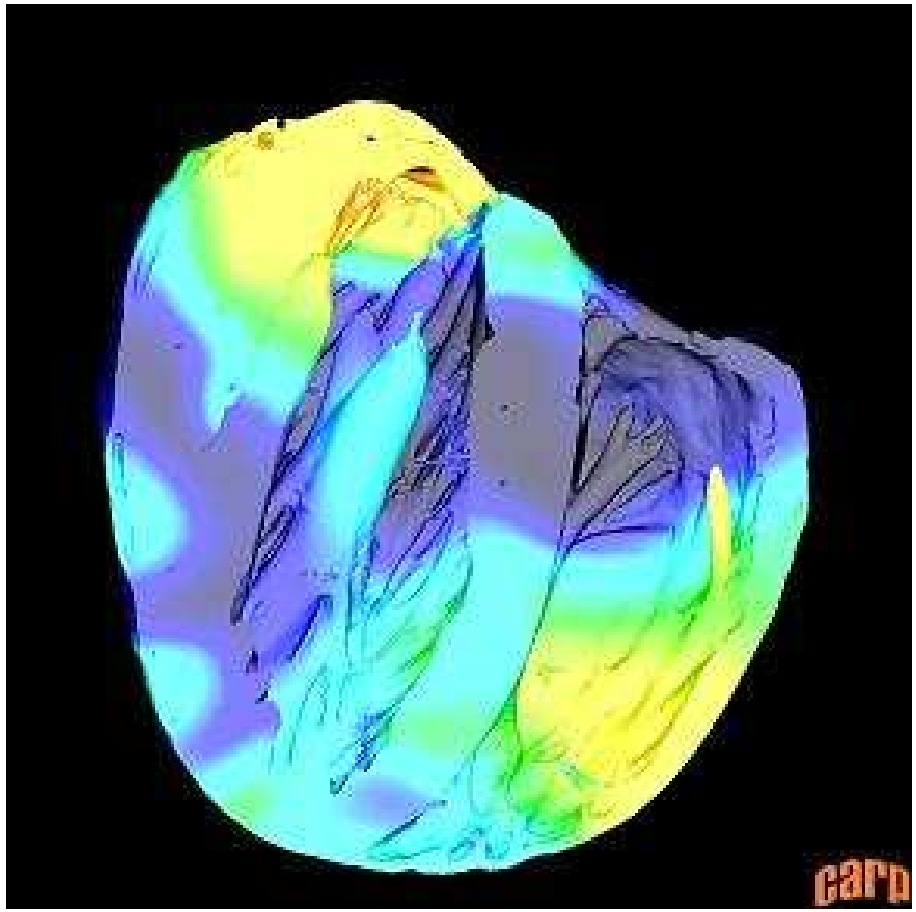
THE AUSTRALIAN NATIONAL UNIVERSITY

1 Overview

- the Chaste project
- modelling in Chaste
- the vayu cluster
- related work
  - effect of NUMA affinity (IPM)
- benchmark configuration and methodology
- results
  - scalability and hardware event count analysis; IPM insights
- conclusions and future work

2 The Chase Cardiac Simulation Project

- Chaste: software infrastructure for modelling the electro-mechanical properties of the heart
- large system of C++ code, many dependencies
- required resolution necessitates parallelization via MPI
- most computationally-intensive part is solution of a large sparse linear system once per timestep
- workload uses a high resolution rabbit heart (Oxford University) (2 $\times$ 1 GB files – 4 million nodes, 24 million elements)



3 Cardiac Modelling in Chaste

- bi-domain equations: continuity of electrical charges within & between heart cells
- chemical reaction kinetics describe ion transport at cell membranes
- after spatial discretisation by finite element method,

$$M_u \frac{d\mathbf{u}}{dt} = A_i\,\mathbf{u}_i - \mathbf{g}(\mathbf{u},\mathbf{c}) \qquad M_c \frac{d\mathbf{c}}{dt} = \mathbf{f}(\mathbf{u},\mathbf{c}) \qquad A_i\,\mathbf{u}_i + A_e\,\mathbf{u}_e = 0$$

- $\mathbf{u}_i$ and $\mathbf{u}_e$ are the electric potentials (inside & outside cells) ($\mathbf{u} = \mathbf{u}_i - \mathbf{u}_e$)
- $\mathbf{c}$: chemical composition at the cell membranes
- $g()$: current across the membranes, $f()$: kinetics of the ion channels
- results in the following linear system

$$\begin{bmatrix} M_u & 0 & hA_i \\ 0 & M_c & 0 \\ A_e & 0 & -(A_i + A_e) \end{bmatrix} \begin{bmatrix} \mathbf{u}^{k+1} \\ \mathbf{c}^{k+1} \\ \mathbf{u}_i^{k+1} \end{bmatrix} = \begin{bmatrix} M_u\,\mathbf{u}^k - hg(\mathbf{u}^k,\mathbf{c}^k) \\ M_c\,\mathbf{c}^k + hf(\mathbf{u}^k,\mathbf{c}^k) \\ 0 \end{bmatrix}$$

- separate set of equations models the mechanical properties

4 The Vayu Cluster at the NCI National Facility

- 1492 nodes: two 2.93 GHz X5570 quad-core Nehalems (commissioned Mar 2010)
- memory hierarchy: 32KB (per core) / 256KB (per core) / 8MB (per socket); 24 GB RAM
- single plane QDR Infiniband: latency of 2.0μs & 2600 MB/s (uni-) bandwidth per node

- jobs (parallel) I/O via Lustre filesystem
- jobs submitted via locally modified PBS; (by default) allocates 8 consecutively numbered MPI processes to each node
  - typical snapshot:

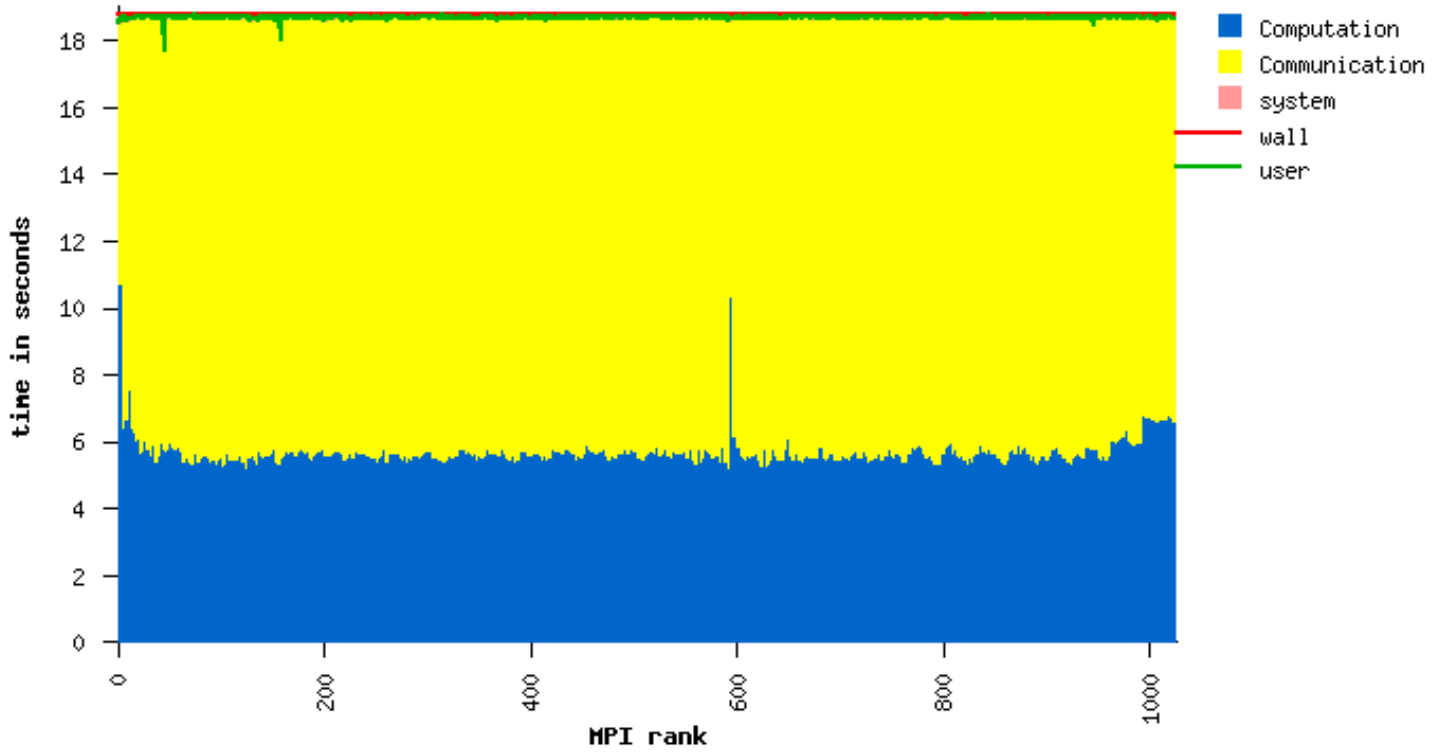
```
1216 running jobs (465 suspended), 280 queued jobs, 11776 cpus in use
```

  - estimating time and memory resources accurately is important!
- allocation for our work was a few thousand CPU hours, max. core count 2048 . . .

5 Related Work

- little so far on scalability of Chaste:
  - parallel efficiency of 53% for 64 processors (3.0 GHz Xeon cluster) – most time in PetSc linear solver (Pitt-Francis et al, 2009)
  - 100ms of bidomain activity took 9 minutes on 2048 cores (Fujitsu, 2010)
- several recent works on using IPM (beginning with Wright et al, 2009)
- efficient performance evaluation methodologies for large-scale atmosphere codes (Strazdins et, al 2011)
  - after 'warming', first few timesteps accurately predicts future performance
  - (local version of) OpenMPI with process and NUMA affinity
    - generally gives 20% better performance
    - reduction in the variability of non-IO intensive benchmark from 12% to 3% on the average error of repeated measurements

6 Effect of NUMA Affinity – Exposed by IPM Profiles



- profiling workloads already running at the limit of memory is hard!
- recent work with the Met Office UM (global atmosphere model) v7.8, no output, 32×32 process grid
- no affinity: groups of 4 processes (socket 0) – spikes in compute times

7 Benchmark Configuration & Methodology

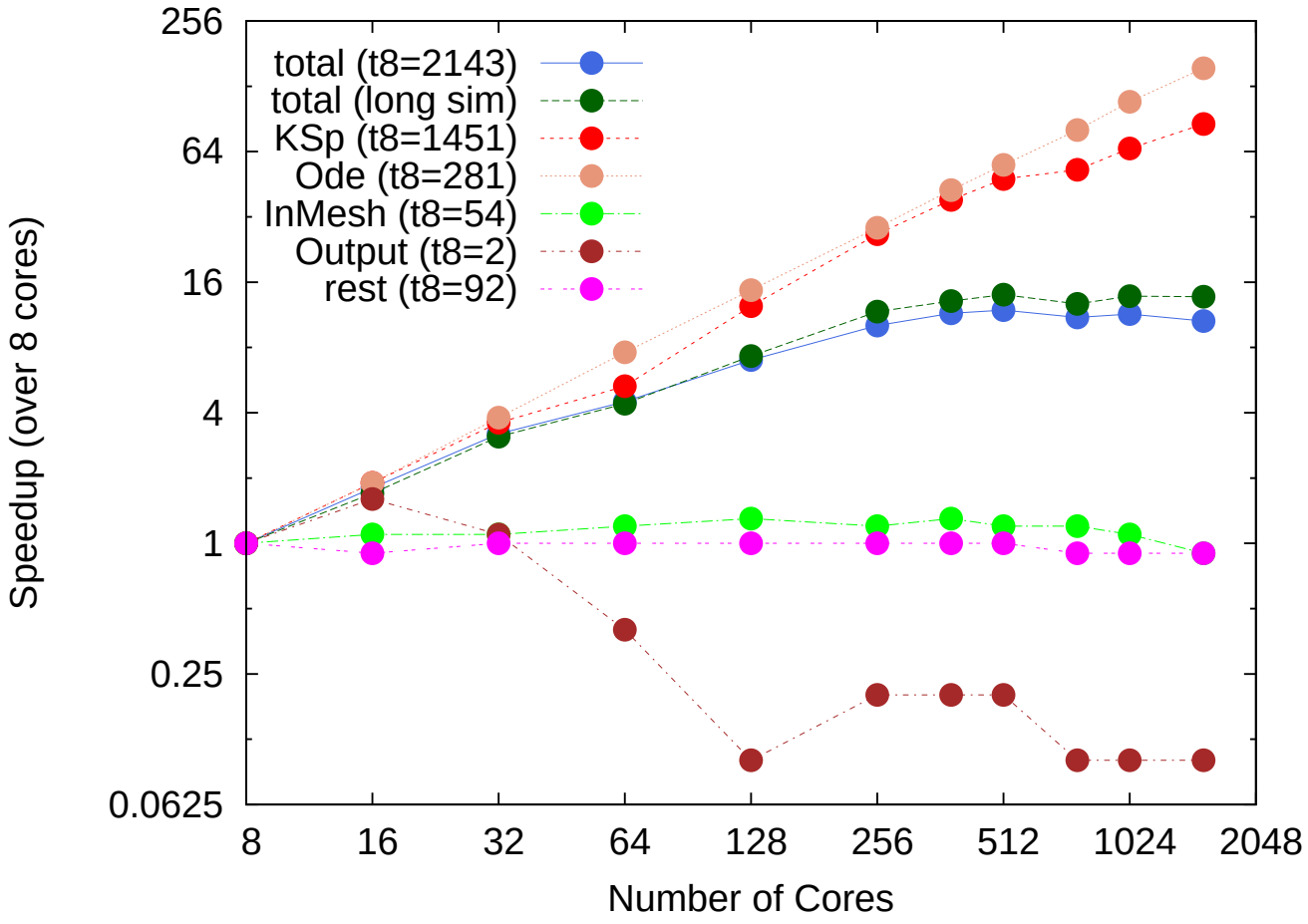
- complex build process via `scons` - hard to get diagnostic information!
- large number of dependencies (Python, HDF5, Parmetis, and PetSc)
- problems traced to conflicts with multiple versions installed on `vayu`
- used `gccOpt` build with g++ 4.1.2), Boost 1.38.0 and MKL 10.1.2

duration	5.0 ms
stimulation duration	0.25 ms (no delay)
timesteps	0.005 ms (ODE), 0.01 ms (PDE), 1ms (output)
KSP tolerance	1e-8 (absolute)
KSP solver	default: `cg` with `bjacobi`
post-processing	none

- activated an internal profiler in the `GenericEventHandler` class: enabled easy IPM profiling of the main sections
- benchmarking methodology: run each sim. 5 times, taking the minimum
  - the minimal number of timesteps for accurate benchmarking (1ms was too short - only 100 iterations)

8 Results – Scaling Behavior

- ‘t8’ is the time in seconds for 8 cores (due to memory constraints, could not run on less)



9 Results – Scalability Analysis

- scalability of total time: max. 11.9 at 512 cores (from 8)
- scalability of ODE and KSp quite high; loss due to un-parallelized 'rest' and inversely scaling 'Output' (using HDF5)
- execution time spent for 1024 cores

section	$\%t$	%comm	main MPI	comments
rest	43	30	all-gather	25% time in I/O
Output	20	30	barrier	high load imbalance
InMesh	19	41	broadcast	
KSp	14	25	all-reduce (8b)	
Ode	1	18	all-reduce (4b)	
AssSys	0.8	0		slight imbalance; some I/O
AssRHS	0.5	7	waitany	high load imbalance

- note: IPM dilated the time spent in the KSp section by 50% and overall by 10%

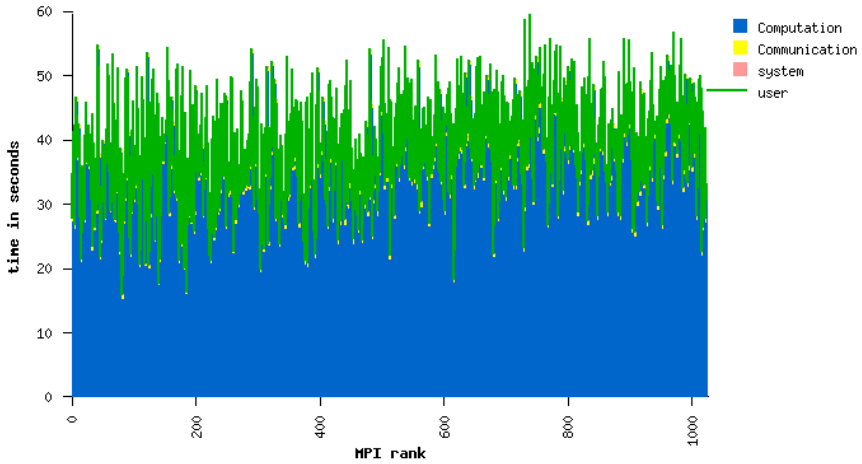
10 Results - Hardware Event Count Analysis

- comparison of 2 solvers (counts $\times 10^9$, average events per process)

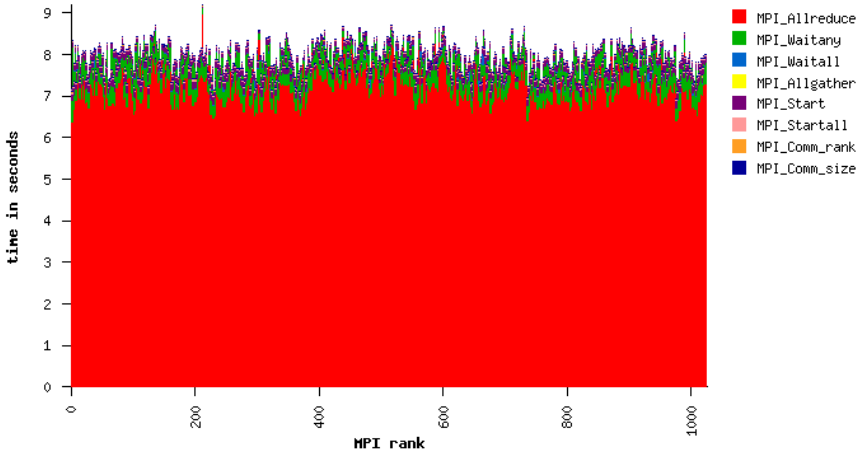
KSp	cores	t job	t total	t KSP	GFLOPS	FLOPs	L2$A	cycles	instr
cg	64	528	474	269	27.9	155	36	854	763
symmlq	64	549	495	292	24.7	144	38	924	688
cg	1024	222	187	21.9	54.6	8.7	2.2	122	201
symmlq	1024	194	176	19.1	62.3	8.3	2.3	101	161

- cg appears faster at 64 cores (despite more flops) but appears slightly slower at 1024
  - probably due to variability in runs - very high due to 'Output'
- number of L2$A (estimated average cost of 4 cycles) indicates poor f.p. speeds
- at 1024 cores, actual job times is 30–50s longer still;
  only $< 10s$ of this attributable to MPI processes startup / shutdown

11 Results: IPM Load Balance and Communication Breakdown



load imbalance for 'Output'



MPI calls for the KSp (symmlq)

- above results fpr 1024 cores

12 Conclusions and Future Work

- working with such codes and systems is hard!
  - 'bleeding edge' technology, variability effects, pushing memory limits, large code systems . . .
  - pre-installation of conflicting versions of dependencies was a pitfall!
- efficient & accurate performance analysis methodology
  - applied to major components via internal profiler and IPM sections
  - IPM was lightweight enough to be used
  - some dilation, but gave some valuable insights (load imbalance, MPI breakdowns and event count statistics)
    - KSp solver not the issue at high core counts!
- future work: investigate parallelizing the 'rest' section
  - further analysis of the 'Output' section – possibly system-wide optimization is needed
    - high variability needs also to be looked into

Acknowledgements!

- NCI NF staff: Margaret Kahn (initial setup), Judy Jenkinson, Jie Cai (IPM)
- James Southern (FLE) – advice on benchmark configuration
- Fujitsu Laboratories Europe for supporting this work

Questons???