Approaches to Performance Evaluation On Shared Memory and Cluster Architectures

Peter Strazdins
(and the CC-NUMA Team),
CC-NUMA Project,
Department of Computer Science,
The Australian National University

seminar at School of IT, Deakin, 09 November 2006

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

THE AUSTRALIAN NATIONAL UNIVERSITY

1 Overview

- approaches to performance evaluation in the CC-NUMA Project
- UltraSPARC SMP simulator development
  - overview
  - detailed memory system modelling
  - validation methodology
  - parallelization
- OpenMP NAS Parallel Benchmarks: a performance evaluation methodology using hardware event counters
- CC-NUMA Project Phase 2: extend to ‘fat-node’ Opteron clusters
- conclusions

2 Approaches to Performance Evaluation in the CC-NUMA Project

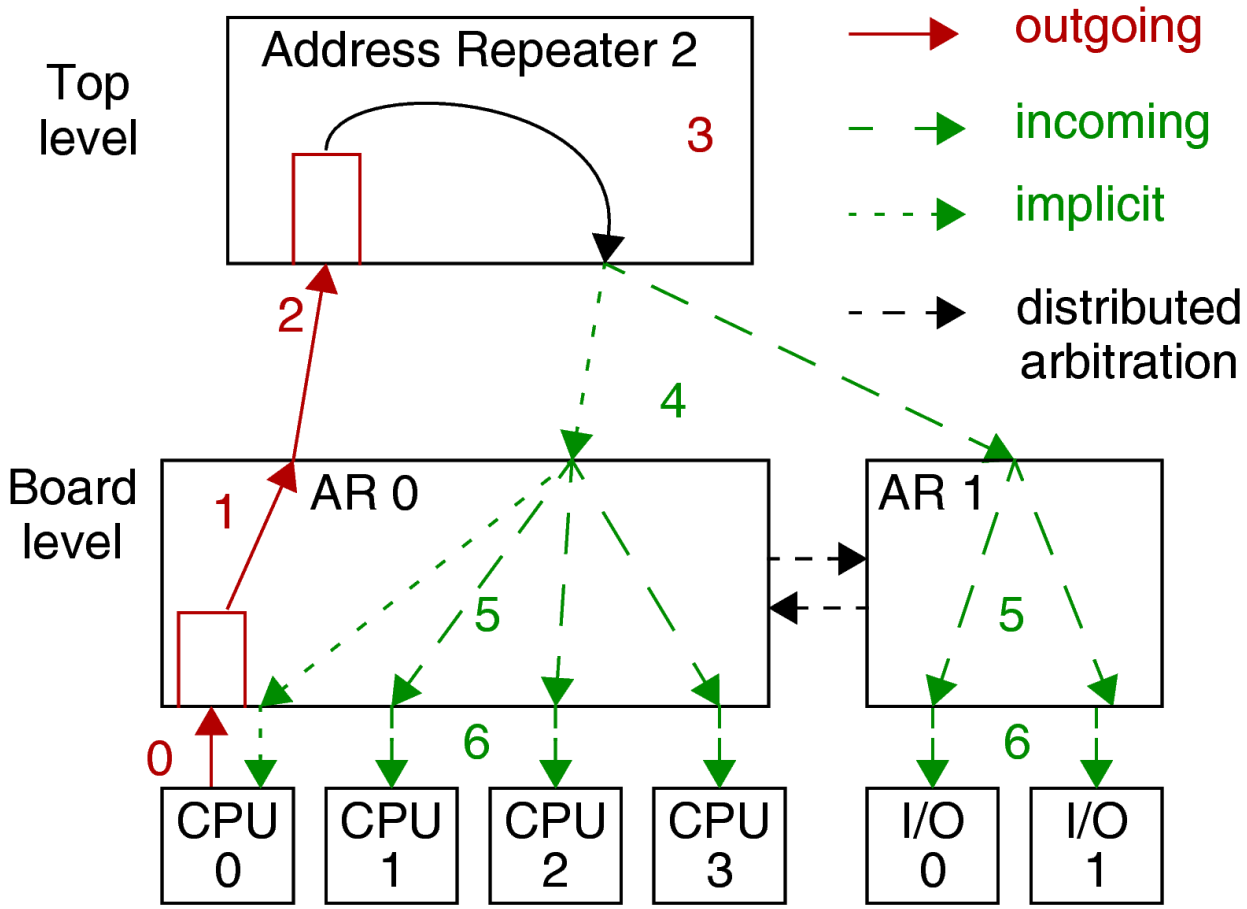
- Sun Microsystems donated a 12 CPU (900 MHz) UltraSPARC V1280 (SMP) to the ANU
  - 32KB I-Cache, 64KB D-Cache, 8MB E-cache
  - relies on hardware/software prefetch for performance
  - Sun FirePlane interconnect (150 MHz)
    - 'fat tree'-like address network, some NUMA effects
- benchmarks of interest: SCF Gaussian-like kernels in C++/OMP (by Joseph Antony)
  - primarily user-level, with memory effects of most interest
  - parallelize with special emphasis on data placement & thread affinity
  - use `libcpc` (CPC library) to obtain useful statistics
  - use simulation for more detailed information (e.g. E-cache miss hot-spots & their causes), or for analysis on larger/variant architectures
- OMP version of NAS Parallel Benchmarks also of interest

3 Sparc-Sulima: an accurate UltraSPARC SMP simulator

- execution-driven simulator with Fetch/Decode/Execute CPU simulator
  - captures both functional simulation *and* timing simulation
  - (almost) complete-machine
  - an efficient cycle-accurate CPU timing module is added
- emulate Solaris system calls at the trap level (Solemn, by Bill Clarke), including LWP traps for thread support

  permits simulation of unmodified (dynamically linked) binaries
- the CPU is connected to the memory system (caches and backplane) via a 'bridge'
  - can have a plain (fixed-latency) or fully pipelined Fireplane-style backplane
- simulator speed: slowdowns in range 500–1000 $\times$
- source code available from Sparc-Sulima home page

4 Target Architecture – UltraSPARC FirePlane Protocol



FirePlane address bus timing (from Alan Charlesworth, The Sun Fireplane System Interconnect, *ACM/IEEE Conference on Supercomputing*, Nov 2001.)

5 Target Architecture – FirePlane Protocol (cont)

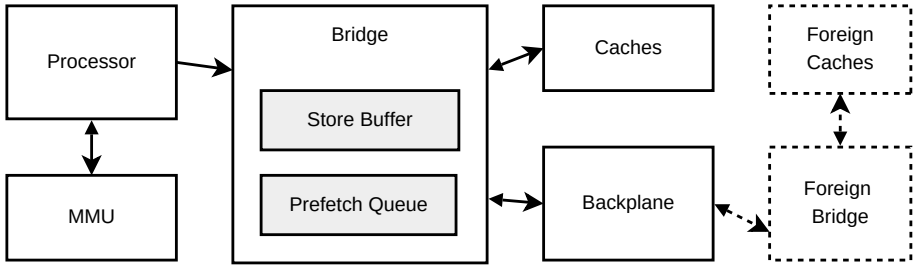
1. CPU 0 begins transaction (cycle 0)
2. system address repeater broadcasts address (cycle 3)
3. all CPUs snoop transaction's address (cycle 7)
   CPUs respond (cycle 11)
   CPUs see result (cycle 15)
4. owner initiates memory request (cycle 7)
5. data transfer begins (cycle 23)
   - completion varies depending on distance
   - 5 cycles for same CPU
   - 9 cycles for same board
   - 14 cycles for different board

note: here 'CPU' means 'the E-cache associated with the respective CPU'

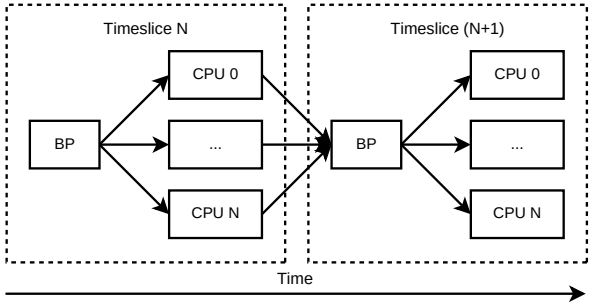
6 Sparc-Sulima: Pipelined Memory System Design

(Andrew Over's PhD topic)

- minimum latency between event & impact on foreign CPU (in the FirePlane) is 7 bus cycles – can apply parallel discrete event simulation techniques



bridge-based structure



run-loop (timeslice = 7*6 CPU cycles)

- asynchronous transactions facilitated by retry of load/store instructions, CPU event queues, and memory request data structures
- simulating the prefetch-cache and store buffer was particularly problem-atic
- added simulation overhead is typically 1.20 – 1.50
- scope for parallelization when running on an SMP host

7 Simulator Validation Methodology

- verifying simulator accuracy is critical for useful performance analysis
  - essential in any kind of performance modelling!
- validation is an ongoing issue in field of simulation
- microbenchmarks: used to verify timing of a multitude of single events
- application-level: by the OpenMP version of the NAS Parallel Benchmarks
  - use of hardware event counters (via UltraSPARC CPC library)
  - √ permits a deeper-level of validation than mere execution time
  - √ also provides breakdown of stall cycles (e.g. D/E-cache miss, store buffer)
  - × hardware counters are not 100% accurate;
    also ambiguously/incompletely specified (e.g. stall cycle attribution)

8 Validation: Microbenchmarks

- e.g. cache-to-cache transfer microbenchmark:

Processor A

```
1:  st      %g0, [A]
    call    _barrier

    call    _barrier
    ba      1b
```

Processor B

```
1:  call    _cache_flush
    call    _barrier
    rd      %tick, %l0
    ld      [A], %g0
    rd      %tick, %l1
    sub     %l1, %l0, %l0
    call    _barrier
    ba      1b
```

- also D/E Cache load/store hit/miss (various cache states/CPU pairs), atomic instr'n latency, store bandwidth, memory access (various regions), RAW, etc
- preferable to (possibly erroneous, out-of-date) data sheets
- provided valuable data, with several surprising & undocumented effects

9 Validation: NAS Benchmarks (S-class)

- p threads; number of cycles target: simulator (% of Total) (2005 results)

Metric (p)	BT		FT		IS		LU		LU-hp		MG		SP	
DC_miss	0.88	5%	0.44	12%	0.97	18%	0.44	10%	1.01	13%	1.13	31%	0.91	22%
SB_stall	1.20	27%	0.93	41%	1.15	54%	0.80	4%	0.84	14%	1.17	2%	0.72	14%
Total (1)	1.06		0.85		1.11		1.03		1.00		0.93		0.97	
Total (2)	1.05		0.78		1.10		1.00		1.00		0.89		0.93	
Total (4)	1.03		0.72		1.17		1.01		1.28		1.02		0.85	
EC_miss	0.16	3%	0.13	4%	0.33	5%	0.12	8%	0.27	19%	0.28	11%	0.20	9%
SB_stall	1.22	27%	0.67	36%	1.22	47%	0.64	9%	0.69	19%	0.45	11%	0.64	19%

- accuracy reasonable ($p = 1$), but was less so as p increases
  - E-cache miss cycles consistently underestimated (possibly target is including cycles in atomic operations and store buffer stalls)
    - but, copy-back and invalidate event counts agreed much more closely

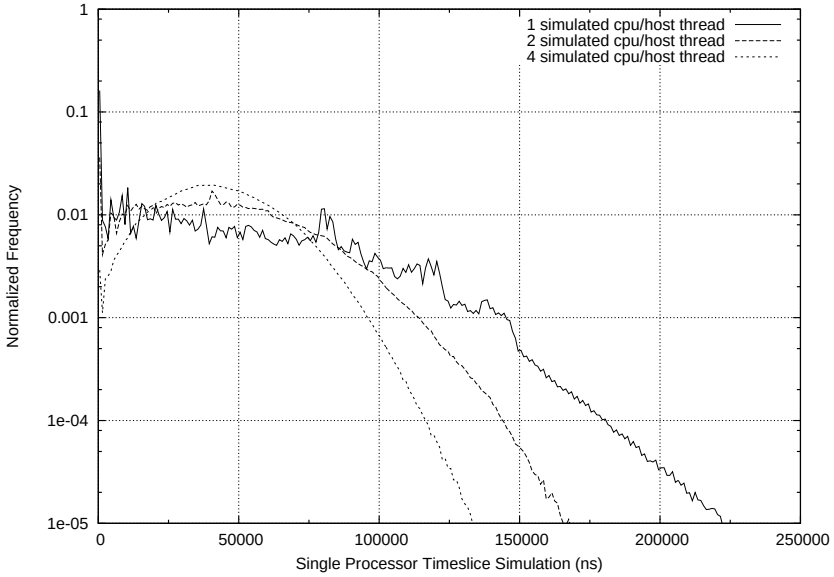
    suspect inaccurate attribution of stall cycles in barrier code to blame
  - improved by modelling the random replacement policy in the D-cache

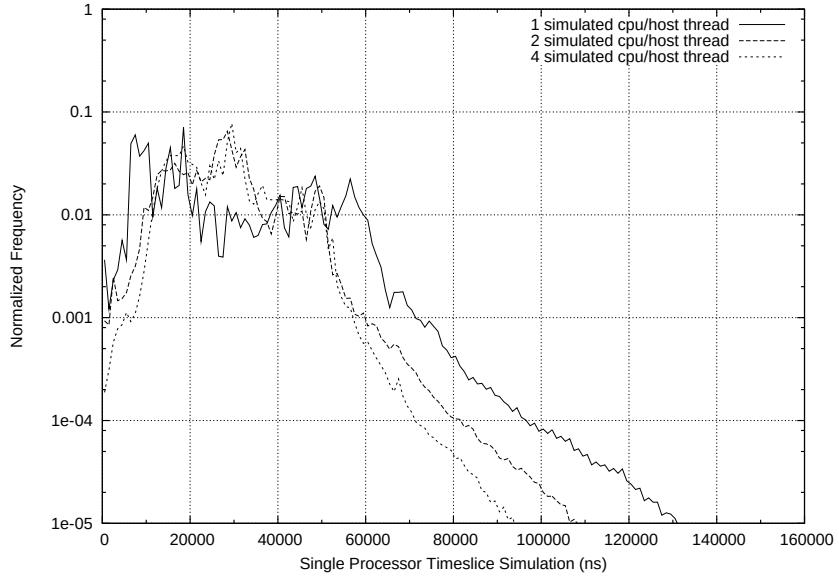
10 Parallelization of the Simulator

- on an SMP, CPUs interact with each other only through the memory system
- hence, natural to assign a separate thread to each simulated CPU
- two approaches:
  - using plain backplane model:
    - each simulated CPU runs for $s \geq 1$ simulated cycles – then synchronize (how big should s be?)
    - memory system access protected by an array of locks, based on the 'index' of the address
  - using pipelined backplane model: $s = 42$ (fixed)
    - backplane simulation phases must be serialized
    - appears to be a problem (Amdahl's Law) – but this is relatively fast
  - but there is a more serious and fundamental problem for both . . .

11 Simulator Parallelization - Load Imbalance Issues

- load imbalance in the time required to simulate s cycles
  - exacerbated by optimizations in Sparc-Sulima: 'skip' stall cycles, 'caching' of calculations (e.g. instruction decoding and address translation)
- histograms of time to simulate $s = 42$ cycles on NAS OMP `ft.S` & `is.W`:

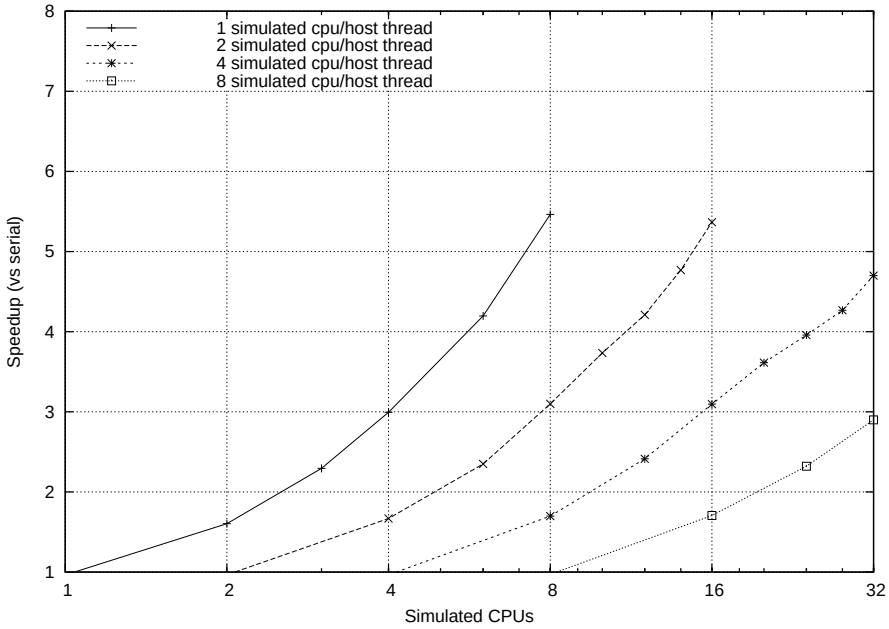


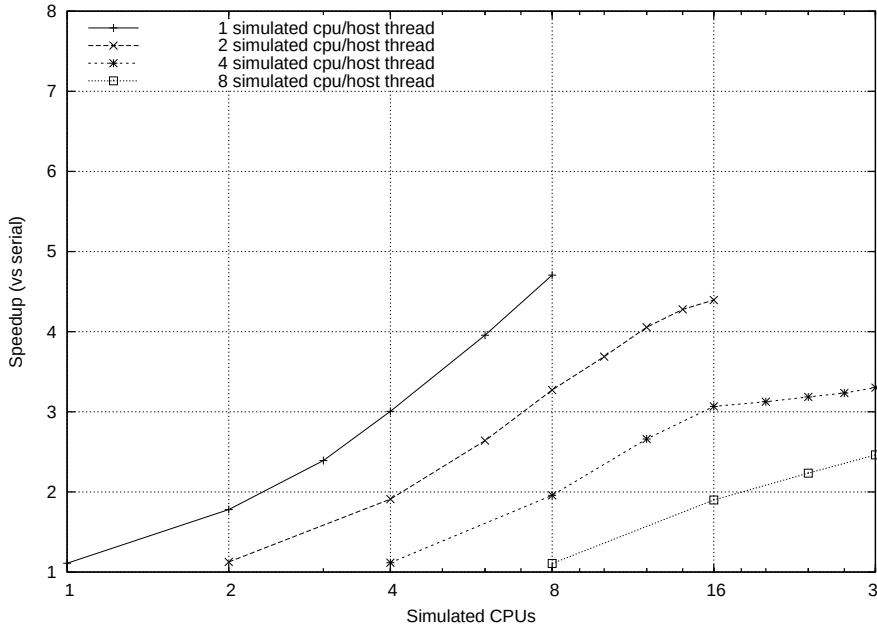


- for the plain backplane model, an accuracy analysis on the NPB indicated that $32 \leq s \leq 256$ was optimal speed–accuracy tradeoff

12 Parallelization - Results

- speedup for NAS OMP `ft.S` of plain ($s = 256$) and pipelined model:





- speedups dependent on application (instruction mix and frequency of stall events)

13 Performance Evaluation Methodology for the OMP NPB

(Honours project by Nic Jean, 2005)

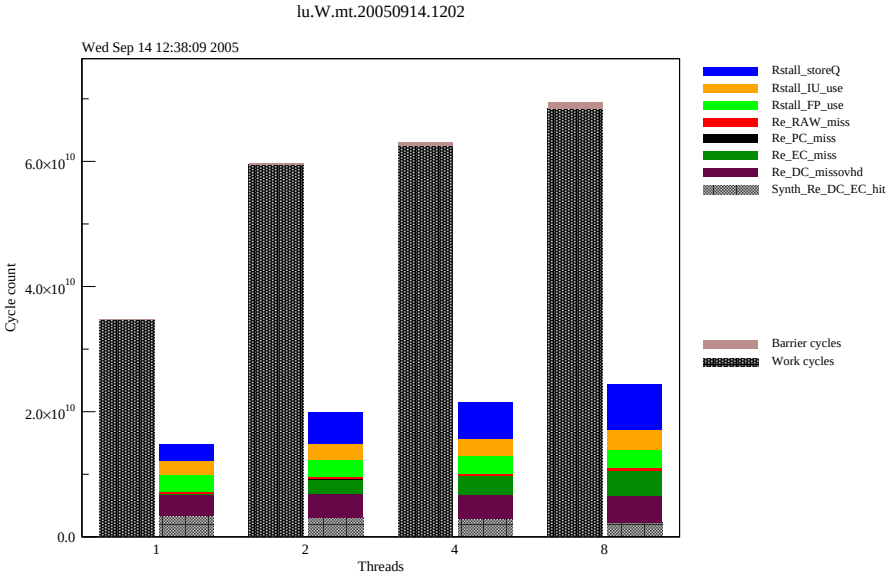
- hardware event counters are widely used in performance analysis for aiding understanding of results
- with OpenMP, obvious approach to use them on the main thread only
  - may not be representative of overall workloads
- time spent in barriers represents application load imbalance
  - not the fault of the architecture! causes pollution of event counts
- on Solaris OpenMP, barriers are implemented relatively simply:

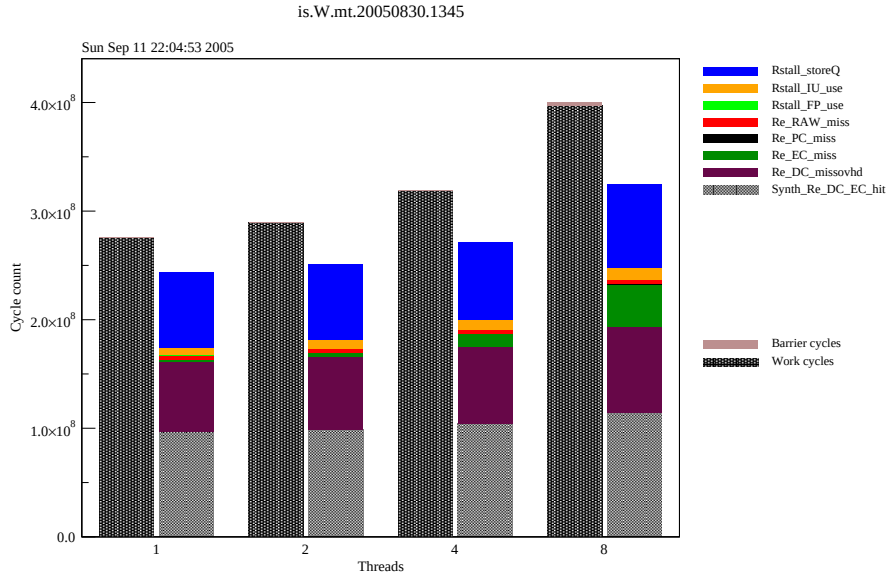
  master: (serial region) $\rightarrow$ (|| region) $\rightarrow$ `EndOfTaskBarrier()` $\rightarrow$ ...

  slave: `WaitForWork()` $\rightarrow$ (|| region) $\rightarrow$ `EndOfTaskBarrier()` $\rightarrow$ ...
- methodology: collect event counts in all threads, separating events associated with barriers
  - `$omp parallel ...  cpc_take_sample(event(iam))`
  - binary edit of `EndOfTaskBarrier()` etc to turn off/on sampling at entry/exit

14 Performance Evaluation of the OMP NPB - Results

- totalled cycles over all threads: (note: slaves have significantly different counts)





`LU.W`: increasing store buffer & and E-cache stalls

`IS.W`: increasing imbalance & CPU stalls from algorithm

- issues: results for each event must be counted on separate runs
  - must always measure CPU cycles on each run
  - slave threads sometimes have much smaller total cycles!

15 CC-NUMA Phase II: CC on NUMA Opteron Clusters

- Sun Microsystem's high-end HPC focus is has shifted on clusters of 'fat' Opteron nodes
  - i.e. 2–4 core, 16-way SMP; NUMA effects due to HyperTransport
  - e.g. the 10,480 CPU cluster at the Tokyo Institute of Technology
- accordingly, CC-NUMA project's Phase II (2007–2009) is oriented to this platform
  - algorithm development and performance evaluation of CC (electronic structure methods)
  - development and use of simulation tools for this platform
    - based on the x86-based Valgrind simulator infrastructure – fast!
    - add similar cycle-accurate CPU and memory models¿
    - also model the communication network
    - and parallelize it all!
    - seeking yet another PhD student. . .

16 Conclusions and Future Work

- accurate and reasonably efficient simulation of a modern NUMA memory system can be achieved
  - entails hard work, and limited by lack of accurate and complete documentation of the target system
  - hardware event counter based validation methodology was reasonably successful, but some issues remain
  - reasonably efficient parallelization has been achieved
  - now, we are analysing some CC applications with it!
- methodology for analysing OMP NPB has yielded some useful results
  - per-thread based analysis with separation of barrier events (caused by application load imbalance) has proved fruitful
- need to extend the performance evaluation methodology and simulation tools to clusters
  - useful to improve cluster performance instrumentation (develop network-level hardware event counter infrastructure)