Approaches to Performance Evaluation On Shared Memory-Supporting Architectures

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

seminar at High Performance Computing System Lab, Tsukuba University,
20 September 2005

http://cs.anu.edu.au/~Peter.Strazdins/seminars
1 Overview

- approaches to performance evaluation in the CC-NUMA Project
- UltraSPARC SMP simulator development
  - overview
  - detailed memory system modelling
  - validation methodology
- OpenMP NAS Parallel Benchmarks: a performance evaluation methodology using hardware event counters
- preliminary ideas: performance instrumentation infrastructure for clusters supporting shared memory
- conclusions and future work
2 Approaches to Performance Evaluation in the CC-NUMA Project

- Sun Microsystems donated a 12 CPU (900 MHz) UltraSPARC V1280 to the ANU
  
  - 32KB I-Cache, 64KB D-Cache, 8MB E-cache
  - relies on hardware/software prefetch for performance
  - Sun FirePlane interconnect (150 MHz)
    - 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 ×
- (old) source code available from Sparc-Sulima home page
4 Sparc-Sulima: Accurate Memory System Design

(Andrew Over’s PhD topic)

- minimum latency between effect and impact on foreign CPU in the Fire-Plane is 7 bus cycles

- bridge-based structure

- 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 problematic

- added simulation overhead us typically 1.20 – 1.50

- scope for parallelization when running on an SMP host
5 Simulator Validation Methodology

- verifying simulator accuracy is critical for useful performance analysis
- validation is an ongoing issue in field of simulation
- microbenchmarks: verify timing of single events (written in assembler)
  - e.g. D/E Cache load/store hit/miss, atomic instr’n latency, etc
  - provided valuable data; several surprising & undocumented effects
- 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)
6 Validation: NAS Benchmarks (S-class)

- $p$ threads; number of cycles target: simulator (% of Total) (new)

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

- Simulator accuracy reasonable ($p = 1$), but less accurate 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 simulation of barrier code to blame
  - Inaccuracies in D-cache probably due to random replacement policy
7 Performance Evaluation Methodology for the OMP NPB

(Nic Jeans’s Honours topic)

- **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) → (|| region) → EndOfTaskBarrier() → ...
  - **slave**: WaitForWork() → (|| region) → EndOfTaskBarrier() → ...
- 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
8 Performance Evaluation of the OMP NPB - Results

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

**LU.W:** increasing store buffer & IS.W: increasing imbalance & and E-cache stalls 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!
9 Performance Evaluation of the OMP NPB - Extending the Methodology

• use the results to determine which feature of the architecture could be modified to improve performance

• promising candidates include:
  • reducing floating point instruction latency ($BT$, $LU$, $SP$)
  • increasing D-cache size and improving replacement policy (most)
    • using Cachegrind with a fully-associative LRU cache model (LRU)
      • can very efficiently determine cache size: cache miss ‘sweet spots’
  • modifying the store buffer (prefetch on entry) ($LU$-$HP$, $FT$, $SP$, $IS$, $BP$)
  • modifying cache coherence protocols

• a (modified) UltraSPARC simulator will be the main tool
Performance Instrumentation for Clusters: Motivations

- performance evaluation methodologies have been largely successful for improving applications and architectures for SMPs
  - clusters can be more easily reconfigured than SMPs!
- to what extent can we extend such methodologies to OpenMP benchmarks when run on clusters (with distributed shared memory)?
- “there are no existing tools that monitor interprocessor traffic in large scale SMP or cluster systems” (Cvetanovic, Cluster 2004)
  - Xmesh (HP) is a tool partially redressing this
    - gathers information from hardware event counters (on CPU and possibly PCI and interconnect chipsets)
    - has GUI showing CPU, System & PCI utilization, cache/TLB miss rates, and interconnect bandwidth over time
    - implemented so far on the Alpha GS1280 / Quadrics cluster
  - not publically available (AFAIK)
11 Supporting (Hardware) Event Counters in DSM Clusters: Ideas

• idea: implement the equivalent of SMP hardware event counters in a cluster with a page-based DSM:

• what SMP E-cache related events have a useful analogy here?
  • E-cache read / write misses and associated stall cycles
  • E-cache invalidate / write-backs and associated stalls
  • counts of memory bank accesses, stall cycles due to bank contention
  • remote memory bank R/W accesses (per CPU)

• interested to add a libcpc style infrastructure to DSM’s such as SCASH
  • apply similarly methodology as used on SMPs to the OMP NPB

• extend to events generally useful for analysing cluster communication
  • i.e. CPU cycles spent related to data sent/received
  • direct (message processing) and indirect (waiting for message; how much was due to contention?)
  • issues: processing of NIC and kernel data, transferring to user space
12 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 issues remain & more work needs to be done
  - plan to parallelize simulator in order to analyse larger workloads
- 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
  - architectural variation experiments still to be done
- need to improve cluster performance instrumentation infrastructure
  - would like to extend SMP methodologies to DSM clusters
  - requires significant low-level and network-specific work