Memory Subsystem Profiling with the Sun Studio Performance Analyzer

CScADS, July 20, 2009

Marty Itzkowitz, Analyzer Project Lead
Sun Microsystems Inc.
marty.itzkowitz@sun.com
Outline

• Memory performance of applications
  > The Sun Studio Performance Analyzer

• Measuring memory subsystem performance
  > Four techniques, each building on the previous ones
    – First, clock-profiling
    – Next, HW counter profiling of instructions
    – Dive deeper into dataspace profiling
    – Dive still deeper into machine profiling
      – What the machine (as opposed to the application) sees
  > Later techniques needed if earlier ones don't fix the problems

• Possible future directions
No Comment
The Message

• Memory performance is crucial to application performance
  > And getting more so with time

• Memory performance is hard to understand
  > Memory subsystems are very complex
    – All components matter
  > HW techniques to hide latency can hide causes

• Memory performance tuning is an art
  > We're trying to make it a science

• The Performance Analyzer is a powerful tool:
  > To capture memory performance data
  > To explore its causes
Memory Performance of Applications

- Operations take place in registers
  > All data must be loaded and stored; latency matters

- A load is a load is a load, but
  > Hit in L1 cache takes 1 clock
  > Miss in L1, hit in L2 cache takes ~10-20 clocks
  > Miss in L1, L2, hit in L3 cache takes ~50 clocks
  > Fetch from memory takes ~200-500 clocks (or more)
  > Page-in from disk takes milliseconds
    - Costs are typical; each system is different

- What matters is total stalls in the pipeline
  > If latency is covered, there's no performance cost
Why Memory Performance is Hard

• SW developers know code, algorithms, data structures
  > What the HW does with them is magic
    – Many, if not most, SW developers can't even read assembler

• HW engineers know instruction, address streams
  > How the SW generates them is magic

• HW performance optimizations further confuse the issue

• Difficulty lies in bridging the gap
  > Get data to show HW perspective to SW developers

• The rest of this talk will show how we do so
Memory Performance Problems

- Some causes of memory performance problems:
  - Initial cache miss, capacity misses
    - Layout and padding; lack of prefetch
  - Conflict cache misses within a thread
    - Striding through arrays
  - Coherence misses across thread
    - Sharing: unavoidable misses
    - False sharing: avoidable miss, not a real conflict
      - Threads refer to different fields in same cache line
      - Different processes use same VA for different PA's
  - Cache and Page coloring
    - Mappings from addresses to cache lines
The Sun Studio Performance Analyzer

- Integrated set of tools for performance measurement
  > Data collection
  > Data examination

- Many types of data:
  > Clock-profiling, HW counter profiling, ...
  > Special support for OpenMP, MPI, Java

- Common command-line and GUI interface for all

- Available on SPARC and X86, Solaris and Linux
  > It's FREE!

- You've seen it before....
Clock Profiling

- Periodic statistical sampling of callstacks
  - `collect -p <interval> target`
    - Note: many other tools do clock-profiling, too
- Shows expensive functions, instructions
  - Is it the heart of the computation, or is it stalled?
  - If it's stalled,
    - Is it stalled waiting for a previous operation?
    - Is it stalled waiting for a load?
    - Is it stalled trying to do a store?
  - Can only guess with clock profiling
Measuring Memory Costs

• Need better data to understand more
  > See: Zagha, et.al., SC `96

• Use HW counters to trigger sampling
  > `collect -h <cntr1>,<val1>,<cntr2>,<val2>,...`
    – As many counters as chip supports
    – `collect` with no arguments prints list for that machine

• Collect counter name, overflow value, callstack
  > Cache misses/references, TLB misses, instructions, ...
  > Cycles, L1-Cache stalls, L2-Cache stalls, ...
    – Measured in cycles; convertible to time

• Shows memory costs based on the counters
Memory Performance Example

- Test code: 8 copies of vector-matrix multiply
  - 8 functions named: `dgemv_<opt-flag><order>`
    - Same computation, different performance
  - Two loop orders
    - Row, column and column,row
    - `<order> = 1, 2`
  - Four optimization levels
    - Compile with `-g, -O, -fast, and -fast -autopar`
    - `<opt-flag> = _g, _opt, _hi, and _p`
Detailed Memory Performance

Separate out costs of the various caches
Two experiments, combined in Analyzer
Memory Performance Problems

• Data shows where in program problems occur
  > High cache misses, TLB misses
    – Does not show why

• Cause is striding through memory
  > Clue from differences between loop order versions
  > In this example, the compiler can diagnose
    – Studio compilers generate commentary to say what they did
      – See next slide

• In general, diagnosing these problems is hard
  > This one is easy – other cases are more difficult
Annotated Source of \texttt{dgemv_hi1}

Loop interchange – compiler knows best order of loops
Compiler commentary from \texttt{-fast} compilation
Dive Deeper

• We understand program instructions, not data

• Want better performance data
  > The data addresses that trigger the problems
  > The data objects that trigger the problems
    – i.e., Source references

• Hard to get data reference address:
  > HW counters skid past triggering instruction
    – Interrupt PC != Trigger PC
    – Current registers may not reflect state at time of event

• Solution:Dataspace profiling
  > Built on top of HW counter profiling
Dataspace Profiling Technology

• Extend HW counter to capture more data
  > `collect -h +<cntr1>,<val1>,+<cntr2>,<val2>,...`
    – + sign in front of counter name

• Causes backtracking at HW profile event delivery
  > Capture trigger PC (might fail)
  > Capture virtual and physical data addresses (might fail)
    – Track register changes that might affect address
  > Post-process to see if branch-target crossed
    – Typically, 95% of backtracking succeeds

• SPARC-only functionality, alas
  > Backtracking not possible on x86/x64
    – But instruction sampling can extend it to x86/x64
Dataspace Profiling Example

• **mcf** from SPEC cpu2000 benchmark suite
  > Single depot vehicle scheduler; network simplex
    - Single-threaded application

• Collect two experiments
  > `-p on -h +ecstall,lo,+ecrm, on`
  > `-p off -h +ecref, on,+dtlbm, on`

• Combine in Analyzer

• See Itzkowitz, *et. al.*, SC|03 for details
Dataspace Profiling: Function List

<table>
<thead>
<tr>
<th>Functions</th>
<th>Callers</th>
<th>Callees</th>
<th>Source</th>
<th>Lines</th>
<th>Disassembly</th>
<th>PCs</th>
<th>DataLayout</th>
<th>DataObjects</th>
<th>Timeline</th>
<th>Leak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total LWP</td>
<td>User LWP</td>
<td>CPU LWP</td>
<td>Stall Cycles</td>
<td>E$ Read Misses</td>
<td>E$ Refs</td>
<td>DTLB Misses</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>552.677</td>
<td>245.404</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>&lt;Total2&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>202.408</td>
<td>80.706</td>
<td>51.1</td>
<td>61.9</td>
<td>62.3</td>
<td>86.4</td>
<td>86.0</td>
<td>refresh_potential</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which functions have memory performance issues

Performance Analyzer [mcf.er, ...]

Data for Selected Object:
- Name: refresh_potential
- PC Address: 280000003000
- Size: 412
- Source File: mcf.er/archives/mcfUtil.c
- Object File: src/mcf
- Load Object: mcf
- Mangled Name: refresh_potential
- Aliases:

Process Times (sec.)/Counts

<table>
<thead>
<tr>
<th></th>
<th>Exclusive</th>
<th>%</th>
<th>Inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>User CPU</td>
<td>280.706 (51.1%)</td>
<td>280.706</td>
<td></td>
</tr>
<tr>
<td>Wait</td>
<td>282.458 (51.1%)</td>
<td>282.458</td>
<td></td>
</tr>
<tr>
<td>Total LWP</td>
<td>282.458 (51.1%)</td>
<td>282.458</td>
<td></td>
</tr>
<tr>
<td>System CPU</td>
<td>1.731 (56.2%)</td>
<td>1.731</td>
<td></td>
</tr>
<tr>
<td>Wait CPU</td>
<td>0.050 (27.8%)</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>User Lock</td>
<td>0. (0%)</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>Text Page Fault</td>
<td>0. (0%)</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>Data Page Fault</td>
<td>0. (0%)</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>Other Wait</td>
<td>0. (0%)</td>
<td>0.</td>
<td></td>
</tr>
</tbody>
</table>
Dataspace Profiling: Data Layout

Show costs against Data Structure Layout, not code
Dataspace Example Conclusions

• Hot references all are to node and arc fields

• Structures not well-aligned for cache
  > Need to pad to cache-line boundary
  > Reorganize structures to put hot fields on same line
    – Note: reorganizing might move hot fields, but not improve perf.

• High TLB misses imply need for large heap pages

• These changes led to ~21% improvement
  > But not following SPEC cpu2000 rules
    – That does not matter for real codes, of course
Dive Still Deeper

• We understand instructions and data, but not machine
  > So far, problems have been in a single thread

• Use same data to explore interactions among threads
  > Sample questions to answer:
    – Which cache lines are hot?
      – Is usage uniform across lines, or is there one very hot line?
    – Which threads refer to those lines?
    – Which addresses are being referred to by which threads?
Advanced Diagnostic Strategy

• Iterative analysis of the data:
  > Slice and dice data into sets of “objects”
    – Cache lines, pages, TLB entries, CPUs, ...
    – Threads, Processes, Time intervals
  > Find the hot objects of one set
  > Filter to include data only for those hot objects
  > Look at other types of objects to see why
    – It is non-trivial to know which ones to look at
  > Repeat as needed
Advanced Diagnostic Techniques

• Collect Dataspace profiling data
  > `collect -h +<cntr1>,<val1>,+<cntr2>,<val2>,...`

• Collect over all threads and processes

• Slice into “Index Object” or “Memory Object” sets
  > Each set has formula for computing an index from records
  > Analyzer has a Tab for each object set
  > Each Tab shows metrics for the objects in each set
    - e.g., Threads, L2-cache lines, ...
Memory and Index Objects

• Index Objects: formula does not use VADDR or PADDR
  > Formula fields present in all records
  > Can be used for all data
  > Some are predefined

• Memory Objects: formula uses VADDR or PADDR
  > Address fields present only dataspace records
  > Definitions depend on the specific physical machine
    – Cache structure, page size, TLB organization
    – Not yet captured automatically, but could be

• Define in `.er.rc` file, based on specific machine
Sample Object Definitions

• Each definition uses one (or more) fields
  > Thread
    - `indxobj_define Threads THRID`
  > Virtual address
    - `mobj_define VA VADDR`
  > L2 cache line
    - `mobj_define PA_L2 (PADDR&0x7ffe0)>>6`

• Can be a lot more complicated
  > e.g., Niagara-2 level-2 data cache line set
    - `mobj_define UST2_L2DCacheSet \ 
      ((((((PADDR>>15)^PADDR)>>9)&0x1f0) | \ 
      (((((PADDR>>7)^PADDR)>>9)&0xc) | \ 
      ((PADDR>>9)&3)))`
Displaying Objects in Analyzer Tabs

- Predefined Tabs
- 2 Tabs from .er.rc file
- Buttons to add custom Tabs
Example: `mttest`

- Analyzer test code
  - Organized as series of tasks
    - Each task queues 4 blocks, spawns 4 threads
    - Threads synchronize differently for each task
    - Each thread calls one of the `compute*` functions for its block

- We will explore why `computeB` is different
  - Takes almost 3X as much time as the others

- Collect experiment:
  - `collect -p on -h +ecstall, on`
Demo
Function List

Alphabetical (name) sort – note ComputeB vs. others
Source for compute*

Lines 1298, 1306, 1314 are **identical**
But they perform **differently**
Set Filter on `computeB`

Filter to show only those events with `computeB` as leaf

Sorry syntax is so ugly
Function List Filtered on ComputeB

Function list only shows callers of ComputeB
(In this example ComputeB is a leaf function)
Threads, VA, and PA_L2 Tabs

Four threads, four virtual addresses, one cache line
Add VA Filter for One Address

Will show only events in `computeB` referring to that one address
Look at VA and Threads again

One thread per address; true of all four addresses
Diagnosis

• Most cache misses are on a single cache line
  > Four threads get the misses
  > Four addresses are referenced
  > Each thread references only one virtual address

<table>
<thead>
<tr>
<th>Line:</th>
<th>Word 0</th>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  > Write from one thread invalidates line for all others

• Classic manifestation of false sharing
  > A notoriously difficult problem to spot
  > In true sharing, multiple threads refer to each address
Potential Future Development

- Enhance data collection
  - Support x86/x64 with instruction-based sampling
    - Set up working group at this meeting?
  - Integrate configuration capture with data collection

- Improve the GUI and navigation
  - Improve filtering grammar and syntax
  - Other usability improvements

- Develop tuning strategy
  - Systematic procedures for exploring problems
For more information

External Sun Studio Website

http://developers.sun.com/sunstudio/

External Sun Studio Performance Tools Website

http://developers.sun.com/sunstudio/overview/topics/ analyzer_index.html

SC'96 paper on HW Counter Profiling

http://portal.acm.org/citation.cfm?id=369028.369059&coll=portal&dl=ACM&CFID=33541981&CFTOKEN=50518735

SC|03 paper on Dataspace Profiling


Solaris Application Programming by Darryl Gove

http://www.sun.com/books/catalog/solaris_app_programming.xml
Acknowledgments

• Nicolai Kosche, PAE
  > Driving force for dataspace profiling enhancements
  > Developed advanced techniques
  > Invented term “DProfile” to refer to those techniques

• The Sun Studio Performance Analyzer team
  > Made it all work
Thank You

Marty Itzkowitz, Analyzer Project Lead
Sun Microsystems Inc.
marty.itzkowitz@sun.com