Today

Recent news

Tool of the day: Profilers

Multi-thread performance
Outline

Recent news

Tool of the day: Profilers

Multi-thread performance
Recent news

Tool of the day: Profilers

Multi-thread performance
Profilers

Slow program execution:

- Poor memory access pattern
- Expensive processing (e.g., division, transcendental functions)
- Control overhead (branches, function calls)

Desired Insight:

- Where is time spent? (Source code location)
- When? (Execution History)
  - Call stack
- What is the limiting factor?

Main Types of Profilers:

- Exact, Sampling
- Hardware, Software
# Reflections on Profilers

<table>
<thead>
<tr>
<th>Sampling</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Fast</td>
<td>- Slow</td>
</tr>
<tr>
<td>- Noisy</td>
<td>+ Exact</td>
</tr>
<tr>
<td>(takes time to converge!)</td>
<td></td>
</tr>
</tbody>
</table>

No free lunch. But: No exact machine-level profiler!
Various profilers

List of profilers:
- Gprof: sampling, software, single-program
- Sysprof: sampling, software, system-wide
- Valgrind: exact, ‘hardware’, single-program
  - callgrind, cachegrind, really
- Perf: sampling, hardware, system-wide
Demo time
Making sense of Perf sample counts

What do Perf sample counts mean?

Individually: not much!

→ Ratios make sense!

What kind of ratios?

- \( \frac{\text{Events in Routine 1}}{\text{Events in Routine 2}} \)
- \( \frac{\text{Events in Line 1}}{\text{Events in Line 2}} \)
- \( \frac{\text{Count of Event 1 in X}}{\text{Count of Event 2 in X}} \)

Always ask: Sample count sufficiently converged?
Perf: Examples

- **instructions / cycles**
  
  Instructions per clock, target > 1 (seen)

- **L1-dcache-load-misses / instructions**
  
  L1 miss rate, target: small, location understood (demo)

- **LLC-load-misses / instructions**
  
  L2 miss rate, target: small

- **stalled-cycles-frontend / cycles**
  
  Instruction fetch stalls. Should never happen—means CPU could not predict where code is going. (→ pipeline stall)

- **stalled-cycles-backend / cycles**

  Execution units (ALU/FPU/Load-store) is waiting for data/computation/...
Perf: Examples

- **instructions / cycles**
  Instructions per clock, target $> 1$ (seen)

- **L1-dcache-load-misses / instructions**
  L1 miss rate, target: small, location understood (demo)

- **LLC-load-misses / instructions**
  L2 miss rate, target: small

- **stalled-cycles-frontend / cycles**
  Instruction fetch stalls. Should never happen—means CPU could not predict where code is going. (→ pipeline stall)

- **stalled-cycles-backend / cycles**
  Execution units (ALU/FPU/load/store) is waiting for data/computation. Front end? Back end?
Learning about PMU events

- Intel Optimization Manual (no.)
  - Intel® 64 and IA-32 Architectures Developer’s Manual: Vol. 3B (yes!)
- AMD Optimization Manual (no.)
  - AMD BIOS and Kernel Developers’ guide for Family 15h processors (yes!)

Latter contain event descriptions.

Former contain advice on what ratios to use.
Perf low-level hw event demo
Outline

Recent news

Tool of the day: Profilers

Multi-thread performance
   Memory-related
   Non-memory-related
Multi-thread performance

Difference to single-thread?
Multi-thread performance

Difference to single-thread?

**Memory System** is (about) the only shared resource.

All ‘interesting’ performance behavior of multiple threads has to do with that.
Recent news

Tool of the day: Profilers

Multi-thread performance
  Memory-related
  Non-memory-related
Multiple threads

Threads v. caches demo
Cache coherency

Example: “MOESI” protocol (e.g. AMD). A cache line holds...

**Modified**  most recent correct copy, memory stale. No other copies.

**Owned**  most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.

**Exclusive**  most recent, correct copy, memory fresh. No other copies.

**Shared**  most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.

**Invalid**  no valid copy of the data.

Phi Profilers  Multi-thread performance
Cache coherency

Example: “MOESI” protocol (e.g. AMD). A cache line holds...

- **Modified**: most recent correct copy, memory stale. No other copies.
- **Owned**: most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.
- **Exclusive**: most recent, correct copy, memory fresh. No other copies.
- **Shared**: most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.
- **Invalid**: no copy.

What states are safe to write? (in my and someone else’s cache)
Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds...

- **Modified**: most recent correct copy, memory stale. No other copies.
- **Owned**: most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.
- **Exclusive**: most recent, correct copy, memory fresh. No other copies.
- **Shared**: most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.
- **Invalid**: no valid copy of the data.

What states are safe to write? (in my and someone else’s cache)

(and transitions to what state?)
Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds...

- **Modified** most recent correct copy, memory stale. No other copies.
- **Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.
- **Exclusive** most recent correct copy, memory fresh. No other copies.
- **Shared** most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.
- **Invalid** no valid copy of the data.

What states are safe to write? (in my and someone else’s cache)

(and transitions to what state?)

What states did the sums array see?
Example: “MOESI” protocol (e.g. AMD). A cache line holds...

**Modified**  most recent correct copy, memory stale. No other copies.

**Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.

**Exclusive** most recent, correct copy, memory fresh. No other copies.

**Shared** most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.

**Invalid** no valid copy of the data.

What states are safe to write? (in my and someone else’s cache)

(And transitions to what state?)

What states did the sums array see?

How do memory fences fit into this picture?
**Cache coherency**

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds...

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modified</strong></td>
<td>Most recent correct copy, memory stale. No other copies.</td>
</tr>
<tr>
<td><strong>Owned</strong></td>
<td>Most recent, correct copy. Other CPUs may hold copies in S state.</td>
</tr>
<tr>
<td><strong>Exclusive</strong></td>
<td>Most recent, correct copy. No other copies.</td>
</tr>
<tr>
<td><strong>Shared</strong></td>
<td>Most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.</td>
</tr>
<tr>
<td><strong>Invalid</strong></td>
<td>No valid copy of the data.</td>
</tr>
</tbody>
</table>

What states are safe to write? (in my and someone else’s cache)

(and transitions to what state?)

What states did the `sums` array see?

How do memory fences fit into this picture?

None of this is instantaneous → queued!
Multiple sockets?
Multiple sockets?

“NUMA”
Contestion/throughput demo
‘crunchy3’ at Courant

num cpus: 32
numa available: 0
numa node 0 100010001000100000000000000000000 − 15.9904 GiB
numa node 1 000000000000000010001000100010000 − 16 GiB
numa node 2 0001000100010001000100000000000000 − 16 GiB
numa node 3 0000000000000000000100010001000000 − 16 GiB
numa node 4 0010001000100010001000000000000000 − 16 GiB
numa node 5 000000000000000000001000100010000010 − 16 GiB
numa node 6 01000100010001000100000000000000000 − 16 GiB
numa node 7 00000000000000000000000100010001000100 − 16 GiB
NUMA results

‘crunchy3’ at Courant

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential</td>
<td>core 0 –&gt; core 0</td>
<td>BW 4189.87 MB/s</td>
<td></td>
<td>sequential</td>
<td>core 1 –&gt; core 0</td>
<td>BW 2409.1 MB/s</td>
<td></td>
<td>sequential</td>
<td>core 2 –&gt; core 0</td>
</tr>
<tr>
<td>sequential</td>
<td>core 3 –&gt; core 0</td>
<td>BW 2474.62 MB/s</td>
<td></td>
<td>sequential</td>
<td>core 4 –&gt; core 0</td>
<td>BW 4244.45 MB/s</td>
<td></td>
<td>sequential</td>
<td>core 5 –&gt; core 0</td>
</tr>
<tr>
<td>sequential</td>
<td>core 29 –&gt; core 0</td>
<td>BW 2048.68 MB/s</td>
<td></td>
<td>sequential</td>
<td>core 30 –&gt; core 0</td>
<td>BW 2087.6 MB/s</td>
<td></td>
<td>sequential</td>
<td>core 31 –&gt; core 0</td>
</tr>
</tbody>
</table>
NUMA results

‘crunchy3’ at Courant

<table>
<thead>
<tr>
<th>All contention core</th>
<th>Core 0: BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.081.85 MB/s</td>
</tr>
<tr>
<td>1</td>
<td>299.177 MB/s</td>
</tr>
<tr>
<td>2</td>
<td>298.853 MB/s</td>
</tr>
<tr>
<td>3</td>
<td>263.735 MB/s</td>
</tr>
<tr>
<td>4</td>
<td>1.081.93 MB/s</td>
</tr>
<tr>
<td>5</td>
<td>299.177 MB/s</td>
</tr>
<tr>
<td>6</td>
<td>202.49 MB/s</td>
</tr>
<tr>
<td>7</td>
<td>434.295 MB/s</td>
</tr>
<tr>
<td>8</td>
<td>233.309 MB/s</td>
</tr>
<tr>
<td>9</td>
<td>233.169 MB/s</td>
</tr>
<tr>
<td>10</td>
<td>202.526 MB/s</td>
</tr>
</tbody>
</table>
NUMA results

‘crunchy3’ at Courant

two—contention core 0 —> core 0 : BW 3306.11 MB/s

two—contention core 1 —> core 0 : BW 2199.7 MB/s

two—contention core 0 —> core 0 : BW 3257.56 MB/s

two—contention core 19 —> core 0 : BW 1885.03 MB/s
NUMA? Do I need to care?

Large multi-core machines *are* NUMA.

Also: Easy, can use OpenMP $\rightarrow$ popular

What happens if you ignore NUMA?

- What happens at `malloc`?
- What happens at ‘first touch’?
- What happens if you don’t pin-to-core?
Recent news

Tool of the day: Profilers

Multi-thread performance
   Memory-related
   Non-memory-related
Recap: superscalar architecture
Recap: superscalar architecture

David Kanter / Realworldtech.com
SMT/“Hyperthreading”

Program

Processor front end

Exec. Unit 1
Exec. Unit 2
Exec. Unit 3
Exec. Unit 4
Exec. Unit 5

Potential issues?

Phi Profilers Multi-thread performance
SMT/“Hyperthreading”

Processor front end

Thread 1
Thread 2

Exec. Unit 1
Exec. Unit 2
Exec. Unit 3
Exec. Unit 4
Exec. Unit 5

Potential issues?

Phi Profilers Multi-thread performance
SMT/ “Hyperthreading”

Processor front end

Exec. Unit 1
Exec. Unit 2
Exec. Unit 3
Exec. Unit 4
Exec. Unit 5

Thread 1
Thread 2

Potential issues?
Potential issues?

- $n \times$ the cache demand!
- Power?

→ Some people just turn it off and manage their own ILP.
Locks are not slow

Lock *contention* is slow
Locks

Locks are not slow

Lock *contention* is slow
Questions?
Image Credits

- Clock: sxc.hu/cema
- Bar chart: sxc.hu/miamiamia