Slow and Steady
Measuring and Tuning Multicore Interference

Dan Iorga, Tyler Sorensen, John Wickerson, Alastair F. Donaldson
Motivation

Real-time:

- Needs timing predictability
- Can benefit from multicore processors

Our work:

- Reproducible measurements
- Uncovering aggressive configurations

https://github.com/mc-imperial/multicore-test-harness
Motivation

10 second deadline
1 second execution time with no interference

https://github.com/mc-imperial/multicore-test-harness
Measuring interference

*Bechtel and Yun. RTAS 2019*
Motivation

- 10 second deadline
- 300 seconds delay with interference

https://github.com/mc-imperial/multicore-test-harness

Image taken from Wikipedia
Comparing enemies
Comparing enemies

Are these reliable?
Comparing enemies

Is this reliable?
Slowdowns

Reported environment

No-throttling 1.2 Ghz

Uncontrolled environment

Controlled environment

Statistical approach

High variation of maximum observed value
Effect of temperature

- Frequency throttling significantly impacts measurements
- Discard all measurements taken at more than 80 °C
Slowdowns

- Reported
- Uncontrolled environment
- No-throttling 1.2 Ghz environment

Less variation
Slowdowns

- Reported
- Uncontrolled environment
- No-throttling 1.2 Ghz environment
- No-throttling 600 Mhz environment

Less variation
Other mitigations

Operating system:
- Disallow thread migration
- Run PUT at max priority
- Ensure parallel execution
- Remove unnecessary software

Compiler:
- Disable compiler optimisation for enemy processes

Hardware:
- Flush caches between runs
Slowdowns

Slowdowns:
- Reported
- Uncontrolled environment
- No-throttling 1.2 Ghz environment
- No-throttling 600 Mhz environment
- Controlled environment

Slightly less variation
Impact of context switches

- Context switches still affect the execution time
- There is a linear correlation between execution time and the number of context switches
Statistical approach

- The maximum observed value is often unreliable
- We choose the $90^{\text{th}}$ percentile instead
Slowdowns

- Reported
- Uncontrolled environment
- No-throttling 1.2 Ghz environment
- No-throttling 600 Mhz environment
- Controlled environment
- Statistical approach

Reasonable confidence
Outline

- Reproducible Measurements
- Uncovering aggressive configurations
- Results
Interference

ACCESS_PATTERN \((st, ld, st)\)

GOAL:
Make this miss!!!

4-way set associative cache
Enemy tuning

- Template enemies for each shared resource
- Victim programs for each enemy
- We tune the enemy programs to cause maximum interference
Enemy tuning

Enemy template

Concrete Enemy 0

Concrete Enemy X

Slowdown: 1.14

Slowdown: 10.08
Selecting optimal configuration

... (+ 5 more)
Enemy templates

1. #define ACCESS_PATTERN(*scratch_addr) ...
2. volatile int8_t *scratch = (int8_t*) malloc(SCRATCH_SIZE);
3. for (HEADER)
4. for (int i = 0; i += STRIDE; i < SCRATCH_SIZE)
5. ACCESS_PATTERN(&(scratch[i]));

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Enemy range</th>
<th>Cache victim</th>
<th>Memory victim</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCRATCH_SIZE</td>
<td>1-5120KB</td>
<td>LLC</td>
<td>10 x LLC</td>
</tr>
<tr>
<td>STRIDE</td>
<td>1-20</td>
<td>cache line size</td>
<td>Cache line size</td>
</tr>
<tr>
<td>ACCESS_PATTERN</td>
<td>1-5 read/writes</td>
<td>read, write</td>
<td>read, write</td>
</tr>
</tbody>
</table>
Outline

Reproducible Measurements

Uncovering aggressive configurations

Results
## Experimental setup

<table>
<thead>
<tr>
<th>Name</th>
<th>SoC</th>
<th>Arch</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi 3 B</td>
<td>BCM2837</td>
<td>ARM A53</td>
<td>4</td>
</tr>
<tr>
<td>DragonBoard 410c</td>
<td>Adreno306</td>
<td>ARM A53</td>
<td>4</td>
</tr>
<tr>
<td>Intel Joule 570x</td>
<td>570x</td>
<td>Atom x86</td>
<td>4</td>
</tr>
<tr>
<td>Nano-PC T3</td>
<td>S5P6818</td>
<td>Arm A53</td>
<td>8</td>
</tr>
<tr>
<td>BananaPi M3</td>
<td>A837</td>
<td>ARM A7</td>
<td>8</td>
</tr>
</tbody>
</table>

- We experiment on both ARM and Intel architectures
- We use as benchmarks coremark and autobench
Hostile environments

<table>
<thead>
<tr>
<th>Board</th>
<th>Most aggressive hostile environment, per resource</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cache</td>
</tr>
<tr>
<td>Raspberry Pi 3 B</td>
<td>VCCC</td>
</tr>
<tr>
<td>DragonBoard 410c</td>
<td>VMCM</td>
</tr>
<tr>
<td>Intel Joule 570x</td>
<td>VMMC</td>
</tr>
<tr>
<td>Nano-PC T3</td>
<td>VMCM MCMM</td>
</tr>
<tr>
<td>BananaPi M3</td>
<td>VCCC CMCC</td>
</tr>
</tbody>
</table>
Results Visualization

above $y = x$: We achieve a higher slowdown

 touching $y = x$: No clear winner

under $y = x$: Prior work has higher slowdowns
Results
Graveyard of enemies

<table>
<thead>
<tr>
<th>Shared resource targeted</th>
<th>Operation performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>Transfers between the CPU and RAM</td>
</tr>
<tr>
<td>Memory thrashing</td>
<td>Random writes to RAM</td>
</tr>
<tr>
<td>Pipeline</td>
<td>Arithmetic operations</td>
</tr>
<tr>
<td>System</td>
<td>I/O operations</td>
</tr>
</tbody>
</table>
Conclusions

- Reproducible measurements can be obtained using a two-pronged approach: system interference mitigation and a percentile-based metric

- Enemy programs can be precisely compared, and thus tuned using our reproducible metrics

- Tuning can uncover higher slowdowns - achieves a statistically larger slowdown compared to prior work in 35 out of 105 benchmark/chip combinations

Try it out!
https://github.com/mc-imperial/multicore-test-harness

Dan Iorga,
Imperial College London