Dynamic Thread Scheduling in Asymmetric Multicores to Maximize Performance-per-Watt

Authors: Arunachalam Annamalai, Rance Rodrigues, Israel Koren and Sandip Kundu
Paper: euler.ecs.umass.edu/research/HPPAC_2012.pdf

Thomas B. Martinsen
Parallel environments and numerical methods TDT24

November 6, 2012
What do we intend to achieve with this article?

- Look at scheduling for asymmetric multicores.
- See how performance counters can be used for adaptive algorithms.
Overview

- Energy efficiency and heterogeneity
- Operating system schedulers
- Performance counters
- What has already been achieved for asymmetric scheduling
- Proposed scheduling scheme
- Evaluation
Energy efficiency and heterogeneity

- Power and energy consumption has become an increasing problem since last decade.
- Hard to increase the frequency, but can add additional cores.
- Might be more energy efficient to have specialized cores.
- Performance / watt used as energy efficient metric in this paper
Energy efficiency and heterogeneity

- Core A good for floating point operations.
- Core B good for integer operations.
- Phases in a program may have different requirements.

![Bar chart showing IPC/Watt for different workloads with Core A and Core B comparisons.](chart.png)
Scheduler has the responsibility to distribute resources between tasks.

Execution time for tasks are often unknown, restricts some algorithms.

Round Robin and FIFO are scheduling algorithms.

Unaware of the hardware.
Performance counters

- Special-purpose registers that counts hardware-related activities.
- Performance Application Programming Interface (PAPI) interface to performance counters.
- Can measure cache misses, instructions completed, cycles stalled waiting for memory, branches etc...
What has already been achieved for asymmetric scheduling

- Previous experiments have sampled power, and swapped threads based on power efficiency.
- Started to use performance counters instead of sampling.
- Developed a hardware based scheduler, which is able to swap threads between cores for maximizing performance / watt. The scheduler was also able to morph two cores too one strong and one weak.
- This paper just consider thread swapping.
Specialized Cores: Integer and Floating point.
Proposed scheduling scheme

- Run many benchmarks, store INT/FP ratio and the achieved performance/watt.
- Use curve fitting technique for creating an expression that maps INT/FP to performance/watt.
- Create static ruleset based on observations from the expression created to determine when it is useful to swap threads.
- Online performance monitoring, swapping threads based on INT/FP ratio and the static ruleset.
Benchmarks

- INT intensive: bitcount, sha, intStress.
- FP intensive: fpStress, equake, ammp.
- INT and FP mix: apsi, ffti, pi
- Collected INT/FP and performance/watt every 20 ms.
## Performance/watt ratio matrix

<table>
<thead>
<tr>
<th>INT\FP</th>
<th>0% - 20%</th>
<th>&gt;20% - 40%</th>
<th>&gt;40% - 60%</th>
<th>&gt;60% - 80%</th>
<th>&gt;80% - 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.61</td>
<td>0.51</td>
<td>0.36</td>
<td>0.3</td>
<td>0.18</td>
</tr>
<tr>
<td>&gt;20% - 40%</td>
<td>0.78</td>
<td>0.58</td>
<td>0.41</td>
<td>0.33</td>
<td>-</td>
</tr>
<tr>
<td>&gt;40% - 60%</td>
<td>1.14</td>
<td>0.6</td>
<td>0.53</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>&gt;60% - 80%</td>
<td>1.3</td>
<td>0.63</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>&gt;80% - 100%</td>
<td>1.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Curve fitting

- Nonlinear regression.
- Algorithms: Gauss-Newton, Gradient descent, Lavenberg-Marquardt
Dynamic Thread Scheduling:
1. Threads $T_1$ and $T_2$ assigned randomly to cores
2. Do Swap if:
   i. $(\%INT_{FP} \geq 55)$ and $(\%INT_{INT} \leq 35)$
      OR
   ii. $(\%FP_{INT} \geq 20)$ and $(\%FP_{FP} \leq 7)$
3. If no_swap for 20 ms, do Swap if:
   i. $(\%INT_{FP} \geq 55)$ and $(\%INT_{INT} \geq 55)$
      OR
   ii. $(\%FP_{INT} \geq 20)$ and $(\%FP_{FP} \geq 20)$
4. End

- $\%INT_{FP}$: INT instruction percentage of thread on FP core
- $\%INT_{INT}$: INT instruction percentage of thread on INT core
- $\%FP_{INT}$: FP instruction percentage of thread on INT core
- $\%FP_{FP}$: FP instruction percentage of thread on FP core
Window size and history depth

- Window size = number of instructions before considering swap.
- History depth = how many window sizes should pass before considering swap.
- Swap is determined by the average over the history buffer.
• Improvements over round-robin.