Fairness-Aware Scheduling on Single-ISA Heterogeneous Multi-Cores

Kenzo Van Craeynest+
Shoaib Akram+
Wim Heirman +
Aamer Jaleel*
Lieven Eeckhout+

+ Ghent University
* VSSAD, Intel Corporation
Multiple core types
- representing different power/performance trade-offs

Well-established power benefits
- [Kumar et al. MICRO’03, ISCA’04]

Comercial examples
- Big.LITTLE, Kal-El
Prior Work: Put the Thread That Will Benefit the Most on the Big Core

Many different scheduling techniques

- Static scheduling
  *Chen and John, DAC’08*

- Sampling-based scheduling
  *Kumar et al., ISCA’04; Patsilaras et al., TACO’12*

- Proxies for performance
  *Memory-domance (Becchi et al., JILP’08; Koufaty et al., EuroSys’10; Shelepov et al., OS Review’09)*

  *Age-based Scheduling (Lakshminararayana et al., SC’09)*

- Model-based scheduling
  *Van Craeynest et al., ISCA’12; Lukefahr et al., MICRO’12*
Traditional Scheduling can be Suboptimal

execution time
Threads pinned on Small Cores Determine Performance

- 4x small
- 4x big
- 1x big, 3x small

(normalized run-time)

<table>
<thead>
<tr>
<th>Program</th>
<th>4x small</th>
<th>4x big</th>
<th>1x big, 3x small</th>
</tr>
</thead>
<tbody>
<tr>
<td>hist</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>wc</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>lr</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>pca</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>km</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>sm</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>blackscholes</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>canneal</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>swaptions</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>streamcluster</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>dedup</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>ferret (small)</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Fairness-Aware Scheduling on Single-ISA Heterogeneous Multi-Cores

Scheduling methodologies that aim to improve fairness

- Equal-time scheduling
- Equal-progress scheduling

Will show that Fairness-Aware Scheduling

- Significantly improves fairness
  - Allowing QoS, accounting,...
- Significantly reduced run-time for many multi-threaded applications over state-of-the-art throughput-optimizing scheduling
Schedule is fair if slowdown of all running threads is the same

\[
\text{fairness} = 1 - \frac{c_{\downarrow}S}{\mu_{\downarrow}S} = 1 - \frac{\sigma_{\downarrow}S}{\mu_{\downarrow}S} = 1 - \frac{\text{std}_{\downarrow}S}{\text{avg}_{\downarrow}S}
\]

Coefficient of variation, a measure of unfairness

\[
\text{slowdown} = S_{\downarrow}i = \frac{T_{\downarrow}\text{het},i}{T_{\downarrow}\text{big},i}
\]

Number of cycles to execute a thread on a heterogeneous multi-core

Number of cycles to execute a thread in isolation on big core
Experimental Setup

Simulated hardware

<table>
<thead>
<tr>
<th></th>
<th>small</th>
<th>big</th>
</tr>
</thead>
<tbody>
<tr>
<td>issue width</td>
<td>4-wide</td>
<td></td>
</tr>
<tr>
<td>clock frequency</td>
<td>2.6 GHz</td>
<td></td>
</tr>
<tr>
<td>cache hierarchy</td>
<td>32KB (p) / 256 KB (p)/ 16MB (s)</td>
<td></td>
</tr>
<tr>
<td>µarch</td>
<td>in-order</td>
<td>out-of-order</td>
</tr>
</tbody>
</table>

Sniper:
- parallel, hardware-validated x86-64 multi-core simulator

Multi-threaded and multi-programmed workloads
- spec2006, PARSEC and MapReduce
Achieving Fairness: Equal-time Scheduling

- Each thread runs for the same amount of time on each core type
- Can be implemented with minor changes to a Round-robin scheduler
Optimizing for Fairness Reduces Run-time for Homogeneous Multi-Threaded Workloads

1B3S system

normalized run-time

<table>
<thead>
<tr>
<th></th>
<th>hist</th>
<th>wc</th>
<th>lr</th>
<th>pca</th>
<th>km</th>
<th>sim</th>
<th>blackscholes</th>
<th>canneal</th>
<th>swaptions</th>
<th>streamcluster</th>
<th>fluidanimate</th>
<th>dedup</th>
<th>ferret</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Homogeneous

Heterogeneous
Equal-Time Doesn’t Guarantee Equal-Progress

Some threads experience a larger slowdown than others
- Equal time on different core types ≠ equal progress
- Therefore fairness is not guaranteed

---

Running on small core

Running on big core

---

execution time
Achieving Fairness: Equal-progress Fairness-Aware Scheduling

- Guarantee that all threads make the same progress compared to their big-core performance
- Continuously monitor fairness and adjust schedule to achieve fairness

\[ S_i = \frac{T_{\text{het},i}}{T_{\text{big},i}} = \frac{T_{\text{big},i} + T_{\text{small},i}}{R_i} \]

Scale execution time on small core

Overall slowdown of the thread

Performance ratio between big and small core
Estimating the Performance Ratio

- Proposed 3 methods
  - sampling-based
  - history-based
  - model-based
Performance Impact Estimation (PIE)

1. Determine where application spends its execution time
2. Use change in MLP exposed to predict change in \( \text{CPI}_{\text{mem}} \)
3. Use change in ILP exposed to predict change in \( \text{CPI}_{\text{base}} \)

[Van Craeynest et al., ISCA’12]
Fairness-aware Scheduling Across Configurations for Multi-Programmed Workloads

normalized throughput

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Pinned</th>
<th>Throughput-Optimized</th>
<th>Equal-Time</th>
<th>Equal-Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B1S</td>
<td>0.9</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>1B3S</td>
<td>0.9</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>3B1S</td>
<td>0.9</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>1B7S</td>
<td>0.9</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>7B1S</td>
<td>0.9</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
</tr>
</tbody>
</table>

fairness

QoS, cycle-accounting, abstraction of heterogeneity,...
Optimizing Fairness Reduces Run-time for Homogeneous Multi-Threaded Workloads
Optimizing for Fairness Reduces Run-time for Heterogeneous Multi-Threaded Workloads

- Heterogeneous applications
  - Threads can have different performance ratio
- Equal-progress scheduling greatly reduces run-time over throughput-optimized AND equal-time scheduling for heterogeneous multi-threaded applications
Fairness-aware Scheduling Across Configurations for Homogeneous Multi-Threaded Workloads

- pinned
- throughput-optimized
- equal-time
- equal-progress

<table>
<thead>
<tr>
<th></th>
<th>normalized run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B1S</td>
<td>1.0</td>
</tr>
<tr>
<td>1B3S</td>
<td>0.9</td>
</tr>
<tr>
<td>3B1S</td>
<td>0.8</td>
</tr>
<tr>
<td>1B7S</td>
<td>0.9</td>
</tr>
<tr>
<td>7B1S</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Conclusions and Contributions

Proposed Fairness-optimizing scheduling
- Two methods: equal-time and equal-progress

Multi-program workloads
- Achieves average fairness of 86% for a 1B3S system while within 3.6% performance of throughput-optimizing scheduling
- Allows for QoS, cycle-accounting, etc. in heterogeneous systems

Multi-threaded workloads
- Unfair performance results in no performance benefits from heterogeneity
  - Threads running on a big core wait at barriers for threads running on small core
  - Average 14% (and up to 25%) performance improvement over pinned scheduling
Questions?