A Comprehensive Scheduler for Asymmetric Multicore Systems

Juan Carlos Saez*, Manuel Prieto*, Alexandra Fedorova**, and Sergey Blagodurov**

*Complutense University of Madrid
**Simon Fraser University
Contents

- Introduction
- Utility of applications
- Design and Implementation
- Evaluation
- Conclusions and Future Work
Asymmetric Multicore Processors

- Asymmetric Performance
- Common ISA

**Fast Core:**
- High Frequency
- Superscalar
- OOO execution
- Large area requirements
- High power

**Slow Cores:**
- Lower frequency
- Single-Issue
- In order pipelines
- Reduced area
- Low power
Efficiency Specialization: Exploiting ILP diversity

- Speedup on a 2.3GHz core vs. a 1.15GHz core

Sensitive to CPU performance:
- Use complex pipelines efficiently
- Few pipeline stalls

Insensitive to CPU performance:
- High LLC miss-rate
- A lot of mispredicted branches
- Frequent pipeline stalls

SPEC CPU 2006
TLP Specialization: Exploiting TLP Diversity

CMPs → cores per chip ↑ ↑

Good performance for scalable parallel applications

Not so “good” for sequential and non-scalable parallel applications

AMPs: offer the best of both worlds for multi-application workloads

Abundant “low-power” cores for running parallel code

Cores with high single-thread performance for:
• ST apps.
• Accelerate seq. sections of parallel applications

Detection by OS: Runnable thread count
Unleashing the Potential of AMP systems

- Efficiency Specialization: ST apps.
- TLP Specialization: ST and MT apps
- Previous asymmetry-aware schedulers employed one type of specialization only

→ Our goal is to design the comprehensive scheduling support to cater to TLP and ILP diversity
Contents

- Introduction
- Utility of applications
- Design and Implementation
- Evaluation
- Conclusions and Future Work
Direct SF measurement

The IPC-Driven algorithm

Monitor Instructions per second \( (IPS_{\text{slow}}) \) of the current core type

Migrate to FC to obtain \( IPS_{\text{fast}} \)

Update SF

Assign to cores

- First evaluation of **IPC-Driven** done on a simulator
- We implemented it in a real OS and evaluated on real HW
- **Two problems:**
  - Inaccurate IPC ratios
    - Phase change may happen during measurement
  - Refreshing threads create **load imbalance**
    - Contention on scarce FCs
Our scheduling policy relies on estimating SF on the current core type

- Cross-core migrations not required
  - SF Model designed specifically for the asymmetric system in question \(\Rightarrow\) more complex

We provide SF estimation model for cores differing in frequencies

- Estimate completion time for \(K\) instructions
  - \(CT = \text{Computation Time} + \text{Stall Time}\)

Stall time estimated from Last-Level-Cache miss rates (off-core requests)
Do Well-Balanced Parallel Applications benefit from using FCs?

Both fast and slow cores  →  Keeping FCs Busy

Slow cores only

\[
\text{Speedup}_{\text{app}} = f(SF_{\text{app}}, N_{\text{threads}}, NFC)
\]
Utility Factor (TLP+ILP)

\[ U_{\text{factor}_{\text{app}}} = \frac{SF_{\text{app}}}{\left( \text{MAX}(1, N_{\text{threads}} - (NFC - 1)) \right)^2} \]

\[ U_{\text{factor}_{\text{Ti}}} = \frac{SF_{\text{Ti}}}{\left( \text{MAX}(1, N_{\text{threads}} - (NFC - 1)) \right)^2} \]

- Compact metric (ILP+TLP)
- For ST apps \( \Rightarrow \) UF=SF
- Foundation for CAMP
Contents

- Introduction
- Utility of applications
- Design and Implementation
- Evaluation
- Conclusions and Future Work
Goals of CAMP

- **CAMP**: A Comprehensive scheduler for Asymmetric Multicore Processors
- Design **goals**:
  - Efficiency Specialization + TLP Specialization
  - Accelerate sequential parts of parallel applications
    - Boost `SEQUENTIAL_PART` threads without monopolizing FCs
  - Fair-Share scarce FC among threads that benefit the most in the workload (`HIGH_UTILITY` threads)
  - Low runtime overhead
    - Light-weight mechanism to filter out short program phases and reduce migrations
  - Topology-aware design
    - Avoid cross-LLC migrations when thread-to-core mapping need readjusting
Utility Factor and Classes

- **Threads’ UFs guide scheduling decisions**, so the OS needs to monitor:
  - The *runnable thread count* of the application (*process*)
  - LLC miss rate to estimate SF

- UF of a thread determines its **Utility Class**
  - LOW.Utility
  - MEDIUM.Utility
  - HIGH.Utility
  - SEQUENTIAL_PART

- The utility factor (UF) formula:
  \[
  U_{factor_{Ti}} = \frac{S_{F_{Ti}}}{(\text{MAX}(1, N_{threads}^{N_{FC}-1}))^2}
  \]
Utility Factor and Classes

A pair of thresholds (upper and lower) determines the boundaries between utility classes.

- For ST apps UF ranges from 23% to 100%.
- When MT apps are present, UFs as low as 0%

CAMP adjusts thresholds dynamically based on the workload.
Schedulers and Workload types

- CAMP vs. other schedulers:
  - Speedup Factor Driven (SFD) → Efficiency Specialization only
  - Parallelism-Aware Scheduler (PA) → TLP Specialization only
  - Asymmetry-aware Round Robin Scheduler (RR) → Fair-shares FCs

- All schedulers implemented in OpenSolaris
- We report gmean speedup over RR (per application and workload)

Workloads (SPEC CPU 2006, OMP 2001, Minebench, ...)

- ST applications → Efficiency Specialization
  - Wide variety of SFs
  - Assess Accuracy SF model (comparison with “Best Static”)
- 2 workload sets (ST and MT) → TLP specialization
  - Wide range of apps: sequential portion and SF
  - 10 Application pairs
  - More than two apps.
Experimental setup

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Platforms</td>
<td>• AMD Opteron system (NUMA) with 4 quad-core “Barcelona” chips (16 cores)</td>
</tr>
<tr>
<td></td>
<td>• Intel Xeon system (UMA) with 2 “quad-core” chips (8 cores)</td>
</tr>
<tr>
<td>DVFS Settings</td>
<td>AMD → FCs @ 2.3 GHz</td>
</tr>
<tr>
<td></td>
<td>SCs @ 1.15 GHz</td>
</tr>
<tr>
<td></td>
<td>Intel → FCs @ 3.0 GHz</td>
</tr>
<tr>
<td></td>
<td>SCs @ 2.0 GHz</td>
</tr>
</tbody>
</table>

- **ST applications**
- **MT and ST applications**
Singlethreaded applications: Efficiency Specialization

- CAMP and SFD perform similarly since UF=SF for ST apps.

- CAMP performs within 1% range of Best Static in the absence of phase changes but outperforms it when they are present.

- On the Intel platform, SFD and CAMP behave better due to the higher accuracy of the SF model.

- PA behaves like RR since it is unaware of the efficiency of individual threads.
ST and MT applications (set #1): TLP Specialization

CAMP and PA performed comparably in most cases, because they both considered TLP while SFD fails to deliver significant performance gains.

CAMP “properly” schedules memory-intensive sequential parts on SCs.

Does Information on TLP+ILP bring further improvements?
ST and MT applications (set #2): **TLP Specialization**

→ CAMP delivers greater performance gains over PA (up to 13%) for workloads that exhibit a wider diversity in memory-intensity.
PA fails to deliver *efficiency specialization* (no speedup)

SFD is unable to deliver performance comparable to CAMP for workloads that include multi-threaded applications
Contents

- Introduction
- Utility of applications
- Design and Implementation
- Evaluation
- Conclusions and Future Work
Conclusions

- CAMP accomplishes an **efficient use** of an AMP system for a **wide variety** of workloads
  - SFD does not cater to TLP diversity
  - PA does not take advantage of the ILP diversity of workloads

- **Key elements** for the success of CAMP
  - The **Utility Factor (UF)** is a **compact metric** to account for TLP+ILP of applications
  - **Light-weight technique** for discovering which threads utilize fast cores most efficiently
    - Obtaining SF for a thread **does not require running it on each core type**
  - **Short program phases are filtered out** to avoid premature migrations

- **Considering the speedup factor in addition to TLP** brings higher performance improvements (up to 13%)
  - Evident for multi-application workloads exhibiting a **wider variety of memory intensity**
Future Work

- Designing a methodology to **find performance metrics to define SF estimation models** for highly-asymmetric systems:
  - *Profound microarchitectural differences*
  - Different cache hierarchy/size
  - *Not requiring cross-core migrations for obtaining SF*

- **Cache-aware** version of CAMP
  - Light-weight policy that complements to *asymmetry-aware scheduling*
  - Assess the impact of cross-core migrations aimed to keep fast cores busy
Questions?