Delivering fairness on asymmetric multicore systems via contention-aware scheduling

A. García-García, J.C. Sáez, M. Prieto

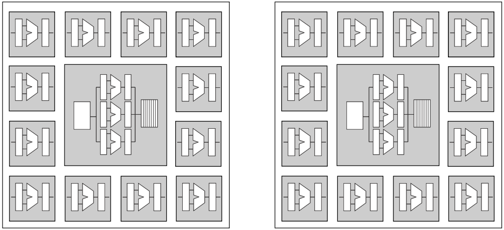
Complutense University of Madrid

The 5th Workshop on Runtime and OSes for the Many-core Era

August 28, 2017

Asymmetric Multicore Processors (AMPs)

- Performance asymmetry: big cores + small cores
- Same Instruction Set Architecture (ISA) but different features:
  - Processor frequency and power consumption
  - Microarchitecture
    - In-order vs. out-of-order pipeline
    - Retirement/issue width
  - Cache(s) size and hierarchy



ArTeCS

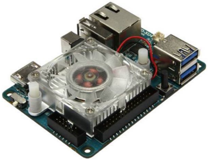
Asymmetric Multicore Processors (AMPs)

AMPs drawed interest from major HW players

- ARM big.LITTLE
- Intel QuickIA prototype (Dual-socket system)

ARM Juno board

- big: 2xARM Cortex A57
- LITTLE: 4xARM Cortex A53
- 8GB DRAM DDR3



Hardkernel ODROID XU4

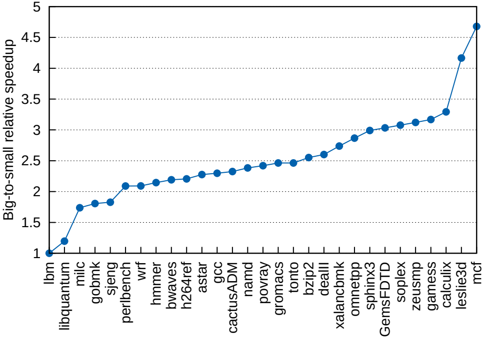
- big: 4xARM Cortex A15
- LITTLE: 4xARM Cortex A7
- 2GB DRAM DDR3

Intel QuickIA

- 4 big cores (Xeon E5450)
- 2 small cores (Atom N330)
- 16GB DRAM DDR3

ArTeCS

OS Scheduling on AMPs: challenges



- Applications may derive different benefit (speedup) from the big cores relative to small ones
- The speedup may vary over time
- Linux default scheduler (CFS) does not factor in this issue when making scheduling decisions

- Most existing approaches strive to **optimize the system throughput**
  - Map High SPeedup (HSP) applications to faster cores
- Our proposal aims to **deliver fairness** on AMPs

Fairness on AMPs

Notion of fairness

- A *completely fair scheduler* ensures **equal slowdowns among same-priority applications**
    - Slowdown: perf. in the workload vs. perf. solo execution

Fairness on AMPs

Notion of fairness

- A *completely fair scheduler* ensures **equal slowdowns among same-priority applications**
  - Slowdown: perf. in the workload vs. perf. solo execution

Conclusions from previous work

1. Fair-sharing big cores (RR) is a suboptimal fairness solution
   - It does not guarantee equal slowdown
   - Subject to throughput degradation
2. Recent extensions in the Linux kernel tailored to ARM big.LITTLE systems are inherently unfair
   - Support for interactive workloads
   - Unpredictable behavior for HPC compute-intensive workloads

Our proposal: the CAMPS scheduler

Goals

1. Optimize fairness while achieving acceptable throughput
2. OS-level solution not requiring special HW extensions or changes in the applications
3. Considers multiple slowdown-related factors when tracking progress

Outline

ArTeCS

Outline

Determining the slowdown at runtime

Definition

$$\text{Slowdown} = \frac{\text{IPS}_{\text{alone}}}{\text{IPS}_{\text{sched}}}$$

- $\text{IPS}_{\text{alone}}$: performance when running alone on a big core
- $\text{IPS}_{\text{sched}}$ can be easily measured using performance monitoring counters (PMCs)
- *Determining the $\text{IPS}_{\text{alone}}$ at runtime is challenging*

Determining the slowdown at runtime

Slowdown-related factors on AMPs

1. An application may be mapped to a small core for some time
   - Slowdown = Big-to-small speedup
2. Shared-resource contention effects
   - cache contention
   - Bus/DRAM-controller contention

Determining the slowdown at runtime

Slowdown-related factors on AMPs

1. An application may be mapped to a small core for some time
   - Slowdown = Big-to-small speedup
2. Shared-resource contention effects
   - cache contention
   - Bus/DRAM-controller contention

- Previous fairness-aware approaches for AMPs factor in the first factor only

ArTeCS

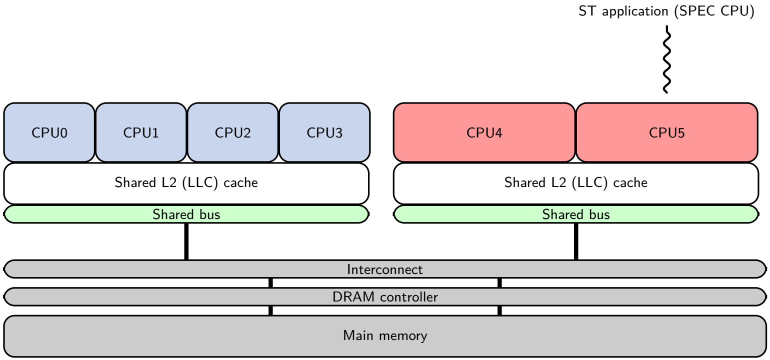
Determining the slowdown at runtime

Slowdown-related factors on AMPs

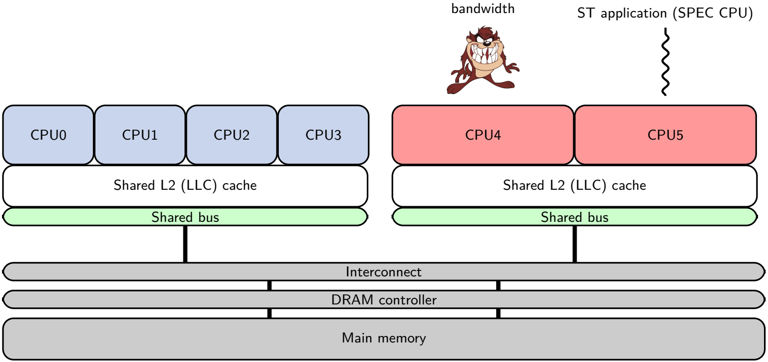
1. An application may be mapped to a small core for some time
   - Slowdown = Big-to-small speedup
2. Shared-resource contention effects
   - cache contention
   - Bus/DRAM-controller contention

- Previous fairness-aware approaches for AMPs factor in the first factor only
- Our approach does take both aspects into consideration

Impact of shared-resource contention on AMPs

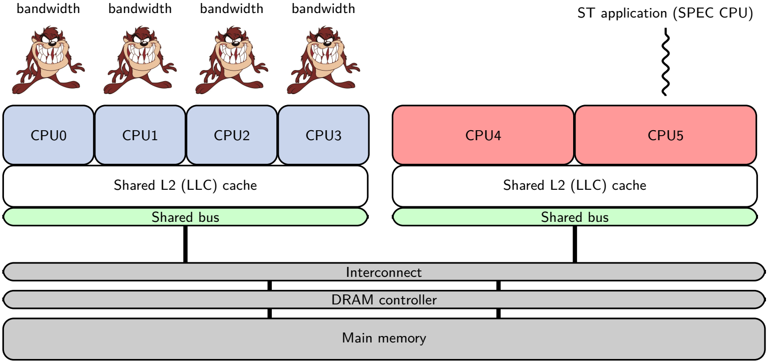


Impact of shared-resource contention on AMPs



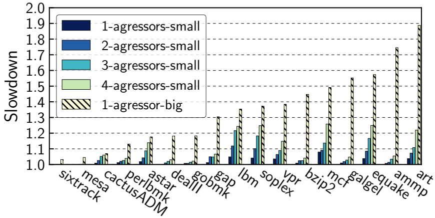
Scenario #1: Slowdown due to interference with big-core applications

Impact of shared-resource contention on AMPs

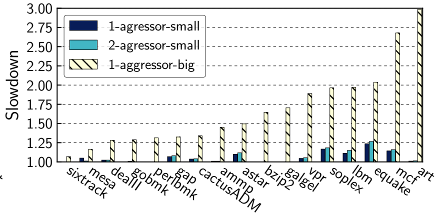


Scenario #2: Slowdown due to interference with small-core applications

Impact of shared-resource contention on AMPs



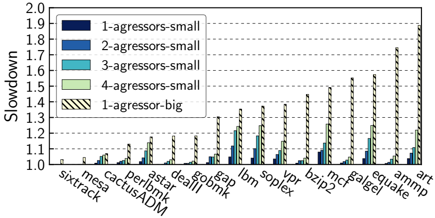
ARM Juno



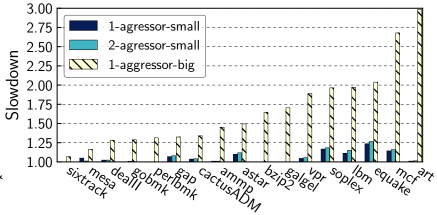
Intel QuickIA

Degradation from interference with big core threads can be substantial: up to 3X

Impact of shared-resource contention on AMPs



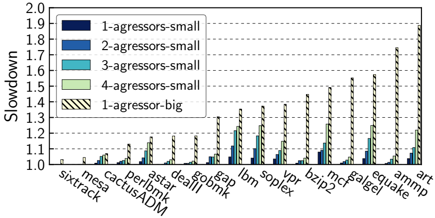
ARM Juno



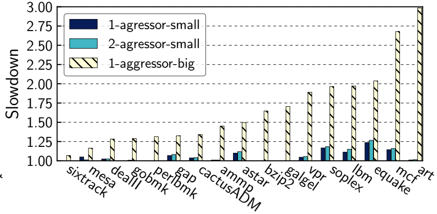
Intel QuickIA

Degradation from interference with big core threads can be substantial: up to 3X
Degradation caused by small-core aggressors is very low: up to 26%

Impact of shared-resource contention on AMPs



ARM Juno



Intel QuickIA

Degradation from interference with big core threads can be substantial: up to 3X
Degradation caused by small-core aggressors is very low: up to 26%
Some applications (low-BTR) are not subject to contention-related degradation

Determining the slowdown online

- The slowdown is aproximated online for each monitoring interval (50ms)

$$\text{Slowdown} = \frac{\text{IPS}_{\text{alone}}}{\text{IPS}_{\text{sched}}}$$

- $\text{IPS}_{\text{sched}}$ measured with PMCs
- $\text{IPS}_{\text{alone}} \approx \text{IPS}_{\text{big}}$ when the thread runs in a low-contention scenario on the big-core cluster
  - Detection of low contention scenarios: BTR (Bus-Transfer-Rate) heuristics [Xu et al., SIGMETRICS'12]
    - $BTR = \frac{bus\_read\_accesses * LLC\_cache\_line\_size * processor\_freq}{total\_cycle\_count}$
  - $\text{IPS}_{\text{alone}}$ varies over time (program phases)

Outline

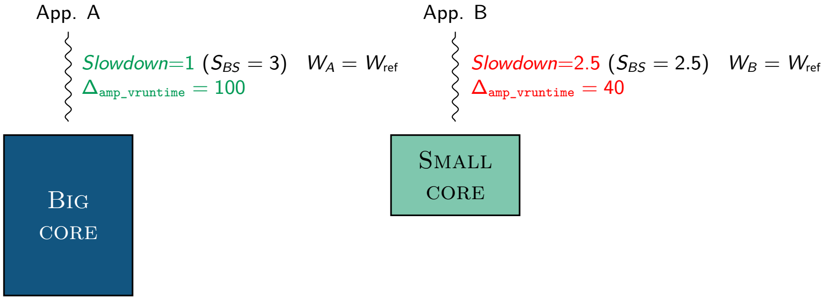
The algorithm

- CAMPS aims to even out the progress made by the various threads in the AMP
- The scheduler maintains two things for each thread
  1. **History table**: IPS_{alone} values for the various program phases
  2. **amp_progress**: Counts progress relative to running alone on a big core in isolation the whole time
     - Counter increases by $\Delta_{\texttt{amp\_progress}}$ every clock tick the thread runs on a big or a small core

$$\Delta_{\texttt{amp\_progress}} = \frac{100 \cdot W_{\mathrm{def}}}{\mathit{Slowdown} \cdot W_{app}}$$

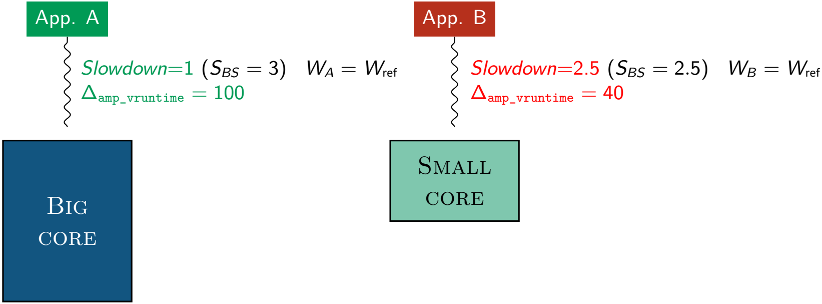
- *Slowdown* estimated online with help of the history table
- W_{app}: application's weight (application priority)
- W_{def}: Weight associated with default priority

CAMPS: Example



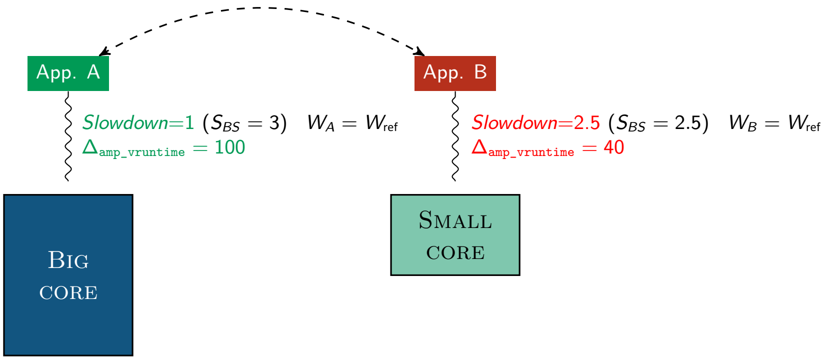
- Optimal throughput but applications do not make the same progress
- *ACFS performs thread swaps to even out progress*
  - When difference between threads's amp_progress exceeds threshold
  - Contention-aware swaps: avoid contention on the big-core cluster

CAMPS: Example



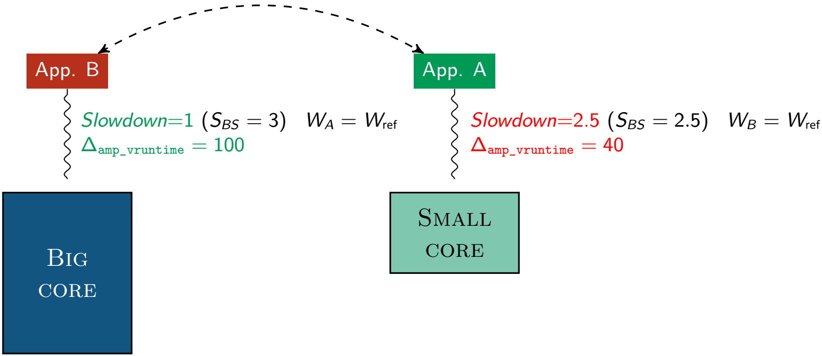
- Optimal throughput but applications do not make the same progress
- *ACFS performs thread swaps to even out progress*
  - When difference between threads's amp_progress exceeds threshold
  - Contention-aware swaps: avoid contention on the big-core cluster

CAMPS: Example



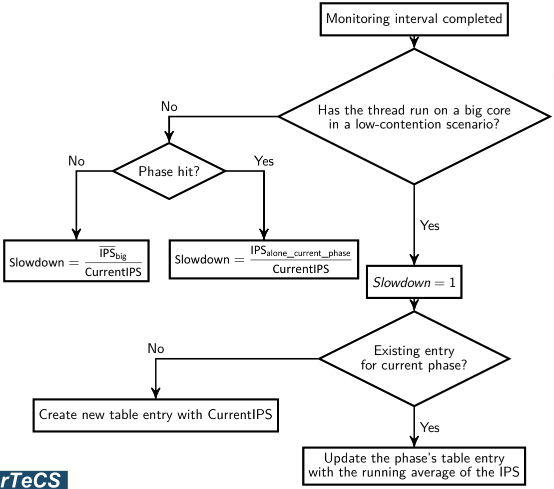
- Optimal throughput but applications do not make the same progress
- *ACFS performs thread swaps to even out progress*
  - When difference between threads's amp_progress exceeds threshold
  - Contention-aware swaps: avoid contention on the big-core cluster

CAMPS: Example



- Optimal throughput but applications do not make the same progress
- *ACFS performs thread swaps to even out progress*
  - When difference between threads's amp_progress exceeds threshold
  - Contention-aware swaps: avoid contention on the big-core cluster

Determining current slowdown: history table



- Two *control metrics* used to index the history table
  1. L1 accesses per 1K instr
  2. % branch instructions retired
- 2 samples belong to the same phase if the *Manhattan distance* of control metrics falls below a threshold

CAMPS: Non-Work-Conserving mode

- *What if low-contention scenarios on the big core cluster do not occur naturally for a given thread?*

CAMPS: Non-Work-Conserving mode

- *What if low-contention scenarios on the big core cluster do not occur naturally for a given thread?*
  - The CAMPS scheduler transitions into a **Non-Work-Conserving (NWC) mode**

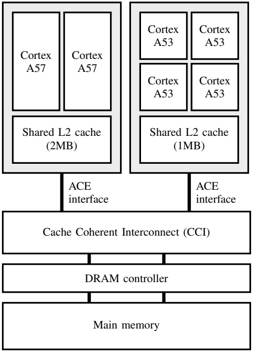
CAMPS: Non-Work-Conserving mode

- *What if low-contention scenarios on the big core cluster do not occur naturally for a given thread?*
  - The CAMPS scheduler transitions into a **Non-Work-Conserving (NWC) mode**
- **Goal NWC mode**: insert new IPS samples in history table of specific thread, as it runs on a big core
  - A low contention scenario is introduced *artificially* by:
    1. Mapping the thread on a big core
    2. Co-scheduling with low-BTR threads to avoid contention on the big core cluster
    3. Some big cores may be disabled temporarily if there is still contention
- When enough samples are collected, CAMPS transitions back into the normal mode

Outline

Experimental platform

Property	Value
Operating system	Ubuntu Server (Linux Kernel v3.10)
Other algorithms	HSP (Throughput), RR, ACFS
Benchmarks	SPEC CPU, PARSEC, NAS, Minebench, FFTW



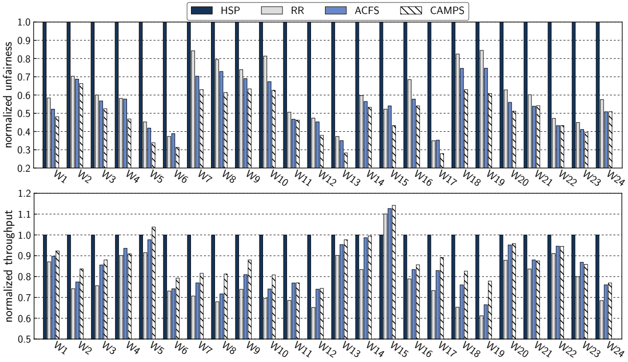
ARM Juno board

Fairness and Throughput Metrics

Metrics

- Fairness → Unfairness Metric
  - $Unfairness = \frac{MAX(Slowdown_{app1},...,Slowdown_{appn})}{MIN(Slowdown_{app1},...,Slowdown_{appn})}$
  - Ideally, equal-priority applications should suffer similar slowdowns
  - Lower-is-better metric
- Throughput → Aggregate SPeedup (ASP)
  - $Aggregate\ Speedup = \sum_{i=app\_1}^{app\_n} \left(\frac{CT_{slow,i}}{CT_{sched,i}} - 1 \right)$
  - Reflects how efficiently the workload uses the AMP
  - Higher-is-better metric

Results



Mixes of *memory-intensive* and *light-sharing* sequential applications

- Optimizing Throughput (HSP) comes at the expense of serious fairness degradation (up to 72% vs CAMPS)
- CAMPS vs. state-of-the-art (ACFS): *10.6% avg. reduction in unfairness, up to 17% increase in throughput*
- *For low-contention workloads, CAMPS matches the results of ACFS*

Outline

Conclusions

- CAMPS: an OS-level contention-aware fair scheduler for AMPs
  1. Factors in the two major slowdown-related aspects when tracking thread progress
  2. Outperforms the state-of-the-art approach (ACFS) in both fairness and thoughput
- CAMPS' Key design aspects
  1. Leverages a history table to aproximate the slowdown online
     - IPS_{big} under low-contention suitable to approximate IPS_{alone}
  2. Scheduler portable across processor models and architectures
     - Does not depend on platform-specific prediction models
     - Does not rely on special hardware extensions to function
     - Necessary performance metrics available in most PMUs

Questions