Exploring the Throughput-Fairness Trade-off on Asymmetric Multicore Systems

J.C. Sáez, A. Pousa*, F. Castro, D. Chaver y M. Prieto

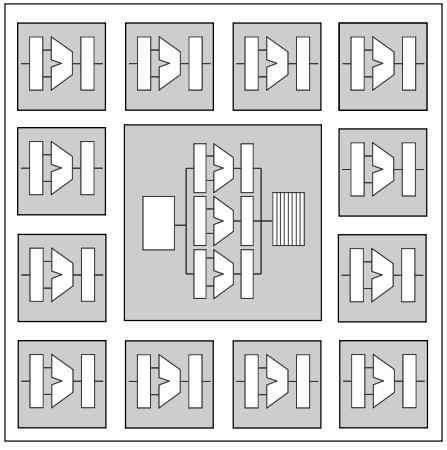
Complutense University of Madrid, *Universidad Nacional de la Plata-LIDI

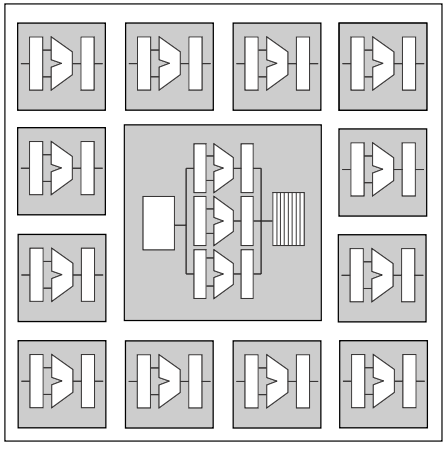
The 2nd Workshop on Runtime and Operating Systems for the Many-core Era (ROME 2014)

August 26, 2014

Asymmetric Multicore Processors (AMPs)

- Performance asymmetry: big *fast* cores + small *slow* cores
- Same instruction set architecture but different features:
  - Processor frequency and power comsumption
  - Microarchitecture
    - In-order vs. out-of-order pipeline
    - Retirement/issue width
  - Cache(s) size and hierarchy





Asymmetric Multicore Processors (AMPs)

AMPs drawed interest from major HW manufacturers

- Intel Quick-IA prototype system
- ARM big.Little

ArTeCS

Asymmetric Multicore Processors (AMPs)

AMPs drawed interest from major HW manufacturers

- Intel Quick-IA prototype system
- ARM big.Little

Challenges to OS scheduling

- Issues
  - Applications may derive different benefit (*speedup*) from the big cores relative to small ones
  - The *speedup* may vary over time
- System software must map threads to cores based on their *varying performance characteristics*

State-of-the-art AMPs schedulers

- Most approaches strive to **optimize the system throughput**
  - Map High SPeedup (HSP) applications to faster cores
  - Examples of the *HSP approach*:
    - Bias Scheduling [Eurosys'10]
    - CAMP [Eurosys'10]
    - PIE [ISCA'12]
- A few proposals aim to **deliver fairness**
  - Threads receive a fair-share of heterogeneous CPU cycles *regardless of the application speedup*
  - RR [CF'05, Eurosys'10], A-DWRR [HPCA'10]

State-of-the-art AMPs schedulers

- Most approaches strive to **optimize the system throughput**
  - Map High SPeedup (HSP) applications to faster cores
  - Examples of the *HSP approach*:
    - Bias Scheduling [Eurosys'10]
    - CAMP [Eurosys'10]
    - PIE [ISCA'12]
- A few proposals aim to **deliver fairness**
  - Threads receive a fair-share of heterogeneous CPU cycles *regardless of the application speedup*
  - RR [CF'05, Eurosys'10], A-DWRR [HPCA'10]

No previous work has comphehensively explored the interrelationship between throughput and fairness on AMPs

Motivation: Analytical study

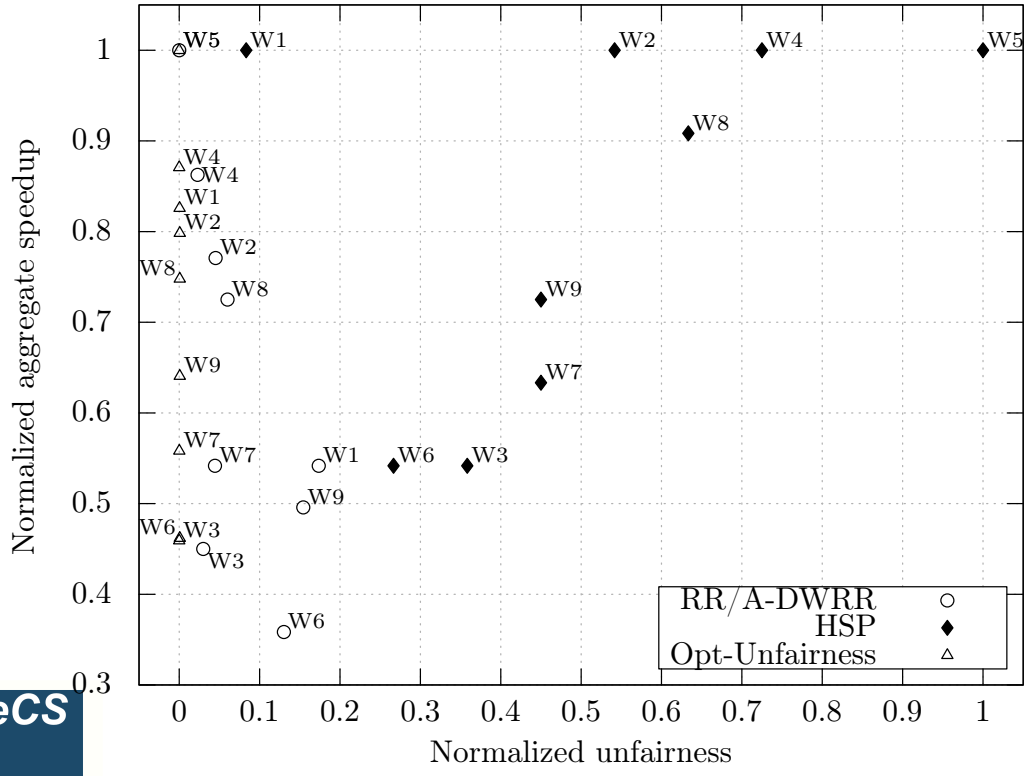
Metrics

- Throughput $\rightarrow$ Agregate 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
- Fairness $\rightarrow$ Unfairness Metric
  - $Unfairness = \frac{MAX(Slowdown_{app1}, \ldots, Slowdown_{appn})}{MIN(Slowdown_{app1}, \ldots, Slowdown_{appn})}$
  - Ideally, equal-priority applications should suffer similar slowdowns
  - Lower-is-better metric

Motivation: Analytical study

Synthetic scenario

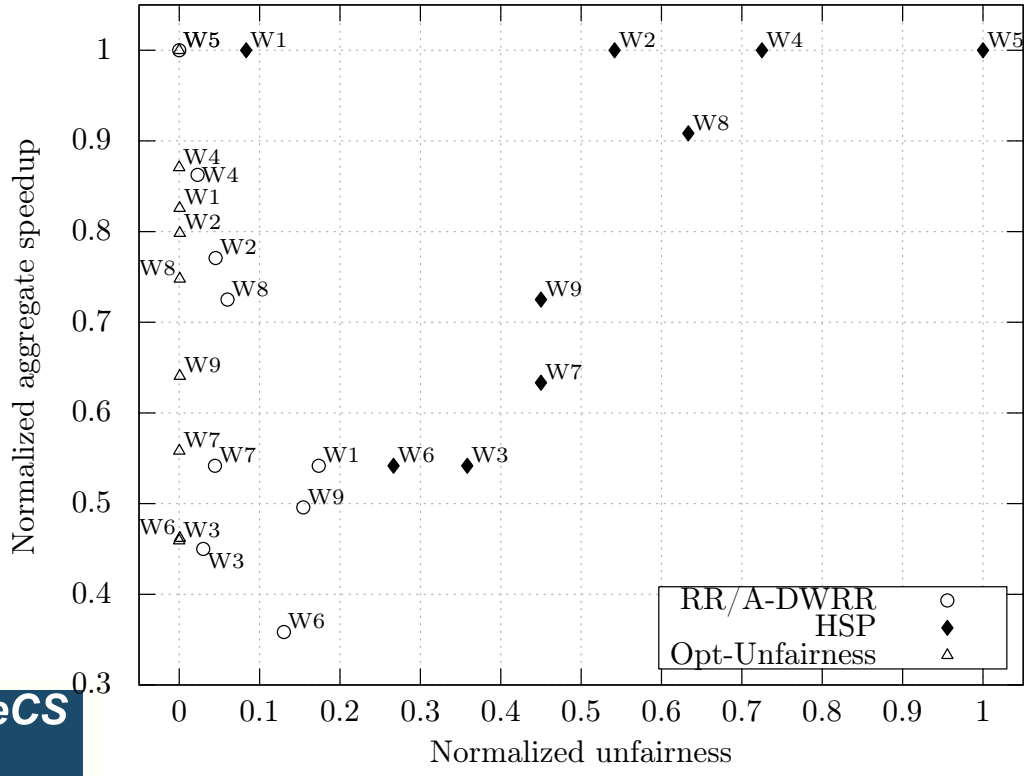
- 9 workloads with 4 single-threaded programs each on 2FC-2SC
- Analytical formulas for the ASP and Unfairness



Motivation: Analytical study

Synthetic scenario

- 9 workloads with 4 single-threaded programs each on 2FC-2SC
- Analytical formulas for the ASP and Unfairness

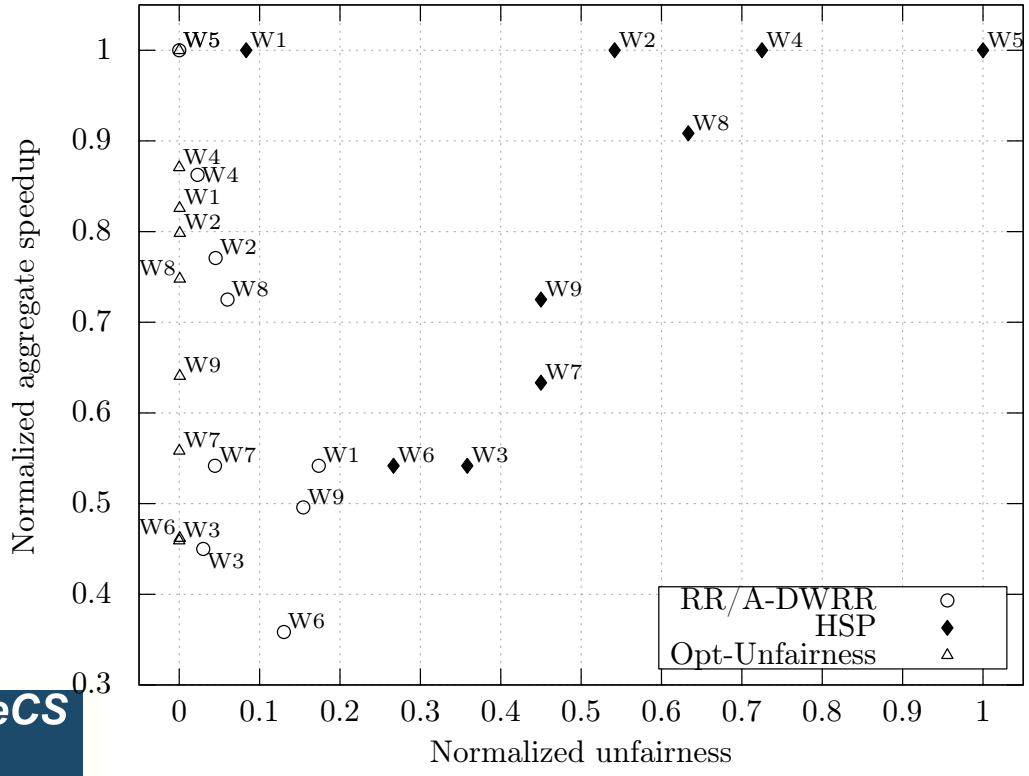


- HSP achieves the better throughput but delivers the worst unfairness values

Motivation: Analytical study

Synthetic scenario

- 9 workloads with 4 single-threaded programs each on 2FC-2SC
- Analytical formulas for the ASP and Unfairness



- HSP achieves the better throughput but delivers the worst unfairness values
- Fair-sharing big cores (RR/A-DWRR) suboptimal solution
  - Unequal slowdowns
  - Throughput degradation

Our proposal: the Prop-SP scheduler

Goals

1. Provide a better balance between fairness and throughput
2. Effectively enforce priorities
3. OS-level solution not requiring HW support nor changes in the applications

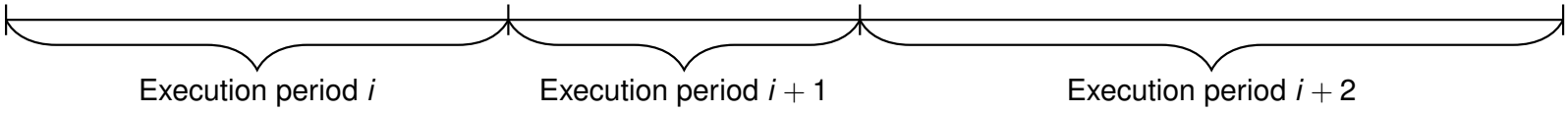
ArTeCS

Outline

Outline

The algorithm

- Prop-SP divides time into variable-length *execution periods* (EPs)



- In an execution period each application receives a fast-core fraction proportional to its *dynamic weight* (DW_{app})

$$DW_{app} = SW_{app} \cdot (SP_{app} - 1)$$

- $SW_{app} \rightarrow$ static weight (application priority)
- $SP_{app} \rightarrow$ *speedup* AMP vs. using slow cores only
  - Estimated at runtime

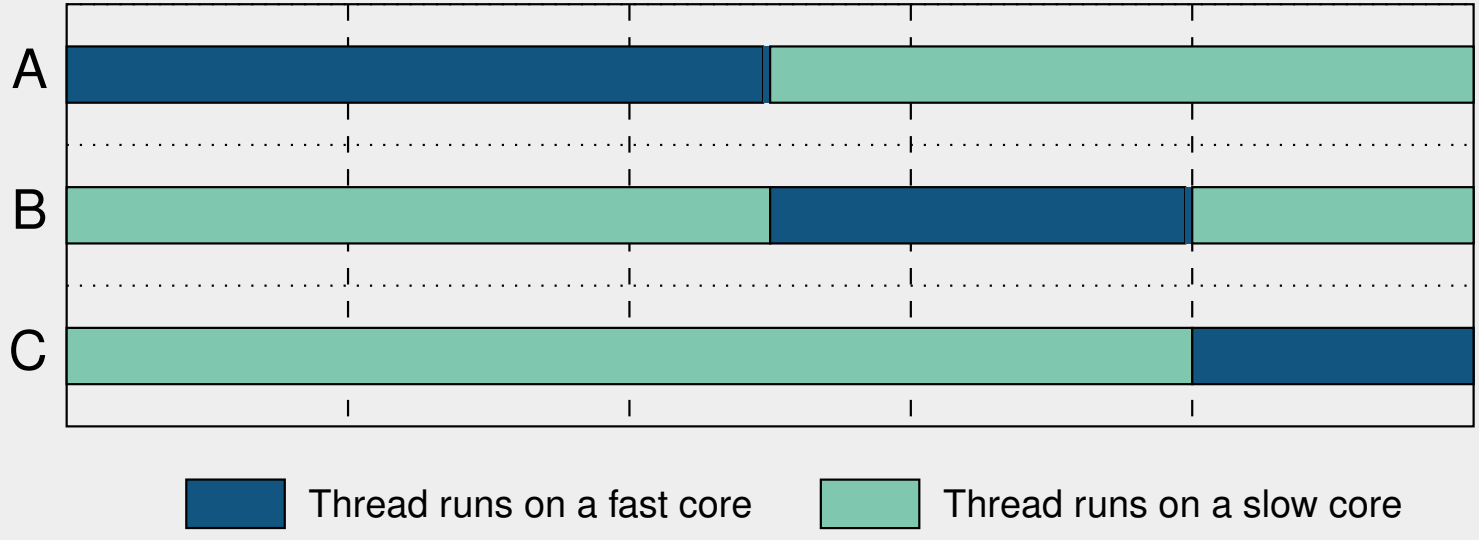
The algorithm (II)

- Fast-core cycle allocation enforced by means of *credits*
  1. Each thread has a fast-core credit counter associated with it
  2. Credits are awarded at the beginning of each EP $\propto DW_{app}$
  3. As a thread runs on a fast core (FC) it consumes credits
  4. When a thread uses up its FC credits $\rightarrow$ thread is *swapped*

The algorithm (II)

- Fast-core cycle allocation enforced by means of *credits*
  1. Each thread has a fast-core credit counter associated with it
  2. Credits are awarded at the beginning of each EP $\propto DW_{app}$
  3. As a thread runs on a fast core (FC) it consumes credits
  4. When a thread uses up its FC credits $\rightarrow$ thread is *swapped*

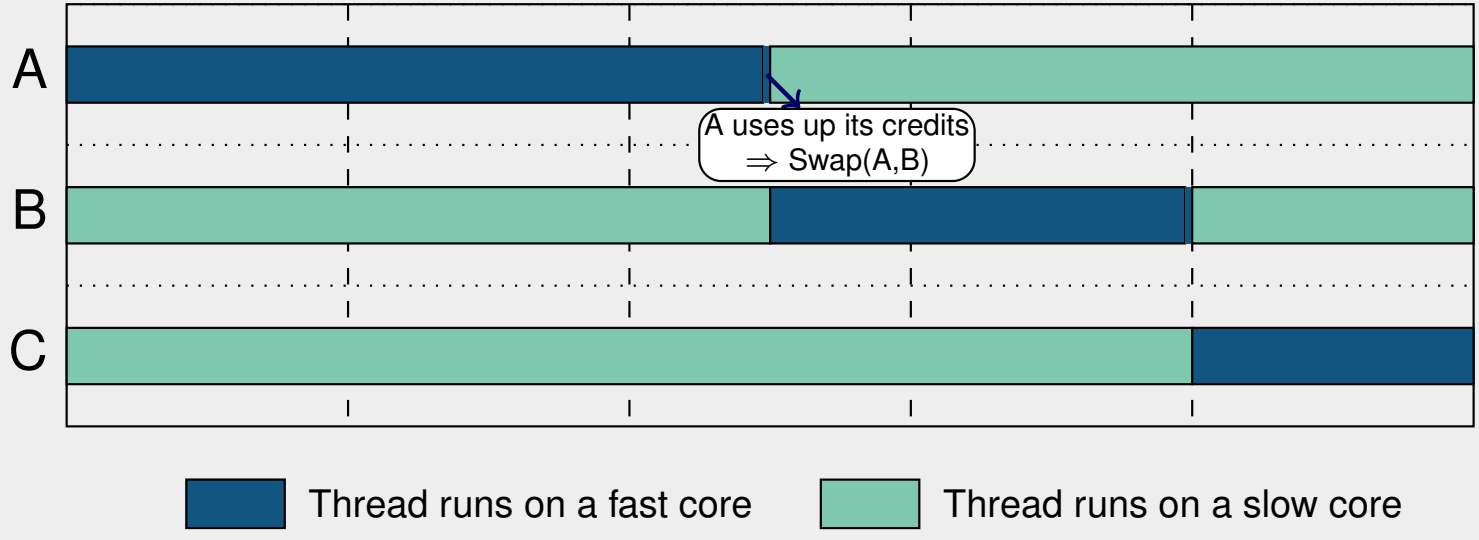
Example: EP for 3 single-threaded apps. on 1FC-2SC



The algorithm (II)

- Fast-core cycle allocation enforced by means of *credits*
  1. Each thread has a fast-core credit counter associated with it
  2. Credits are awarded at the beginning of each EP $\propto DW_{app}$
  3. As a thread runs on a fast core (FC) it consumes credits
  4. When a thread uses up its FC credits $\rightarrow$ thread is *swapped*

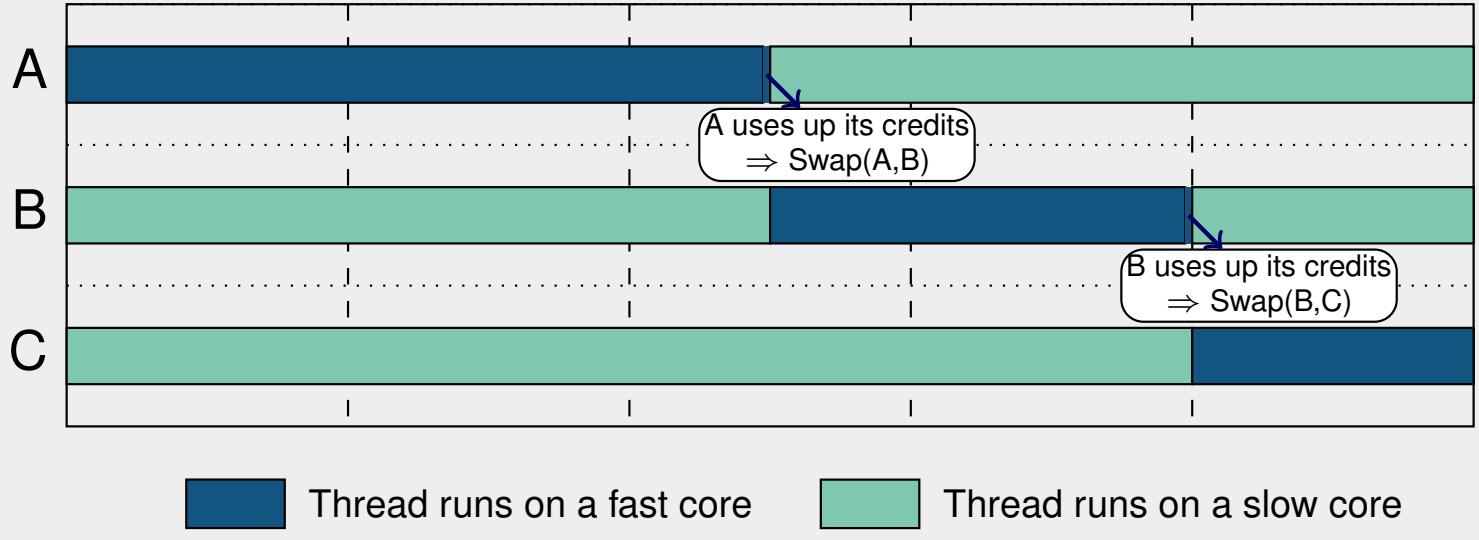
Example: EP for 3 single-threaded apps. on 1FC-2SC



The algorithm (II)

- Fast-core cycle allocation enforced by means of *credits*
  1. Each thread has a fast-core credit counter associated with it
  2. Credits are awarded at the beginning of each EP $\propto DW_{app}$
  3. As a thread runs on a fast core (FC) it consumes credits
  4. When a thread uses up its FC credits $\rightarrow$ thread is *swapped*

Example: EP for 3 single-threaded apps. on 1FC-2SC



Determining the speedup

- Obtaining *speedups* at runtime is one of the major challenges
- For single-threaded apps. *speedup* $\equiv$ *SF* (*Speedup factor*) of its single-runnable thread

$$SF = \frac{IPS_{fast}}{IPS_{slow}}$$

- Previously-proposed schemes to obtain the SF:
  1. Direct measurement on both core types (aka *IPC sampling*)
  2. Profiling-based estimation
  3. **Estimation via performance models (using HW counters)**
     - We rely on the technique proposed in earlier work [ACM TOCS '12]

Support for multithreaded applications

- Fast-core credits allotted to the MT application must be distributed among its runnable threads
  - Several credit-distribution schemes available
  - Application performance is sensitive to the distrib. choice
  - Prop-SP equipped with several credit-distribution schemes

Support for multithreaded applications

- Fast-core credits allotted to the MT application must be distributed among its runnable threads
  - Several credit-distribution schemes available
  - Application performance is sensitive to the distrib. choice
  - Prop-SP equipped with several credit-distribution schemes

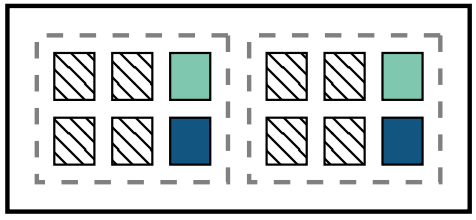
- $SF \neq$ speedup of a MT application in the AMP
  - Also affected by the amount of thread-level parallelism (TLP)
  - Speedup estimation in Prop-SP via analytical models
    - $f(SF_{threads}, TLP, N_{FC})$

Outline

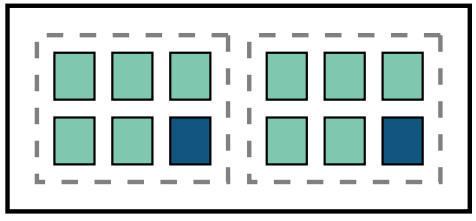
Experimental platform

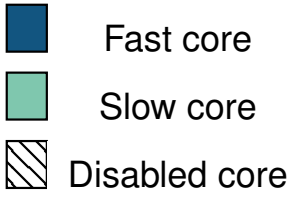
Property	Value
Operating system	OpenSolaris
HW Platform	2x hex-core “Istanbul” AMD Opteron Processor (12-core NUMA system)
DVFS	*fast* cores @ 2.6 GHz *slow* cores @ 800 MHz
Other algorithms	HSP, CAMP, RR, A-DWRR
Benchmarks	SPEC CPU2006, PARSEC, SPEC OMP2001, NAS, Minebench, FFTW, BLAST

2FC-2SC

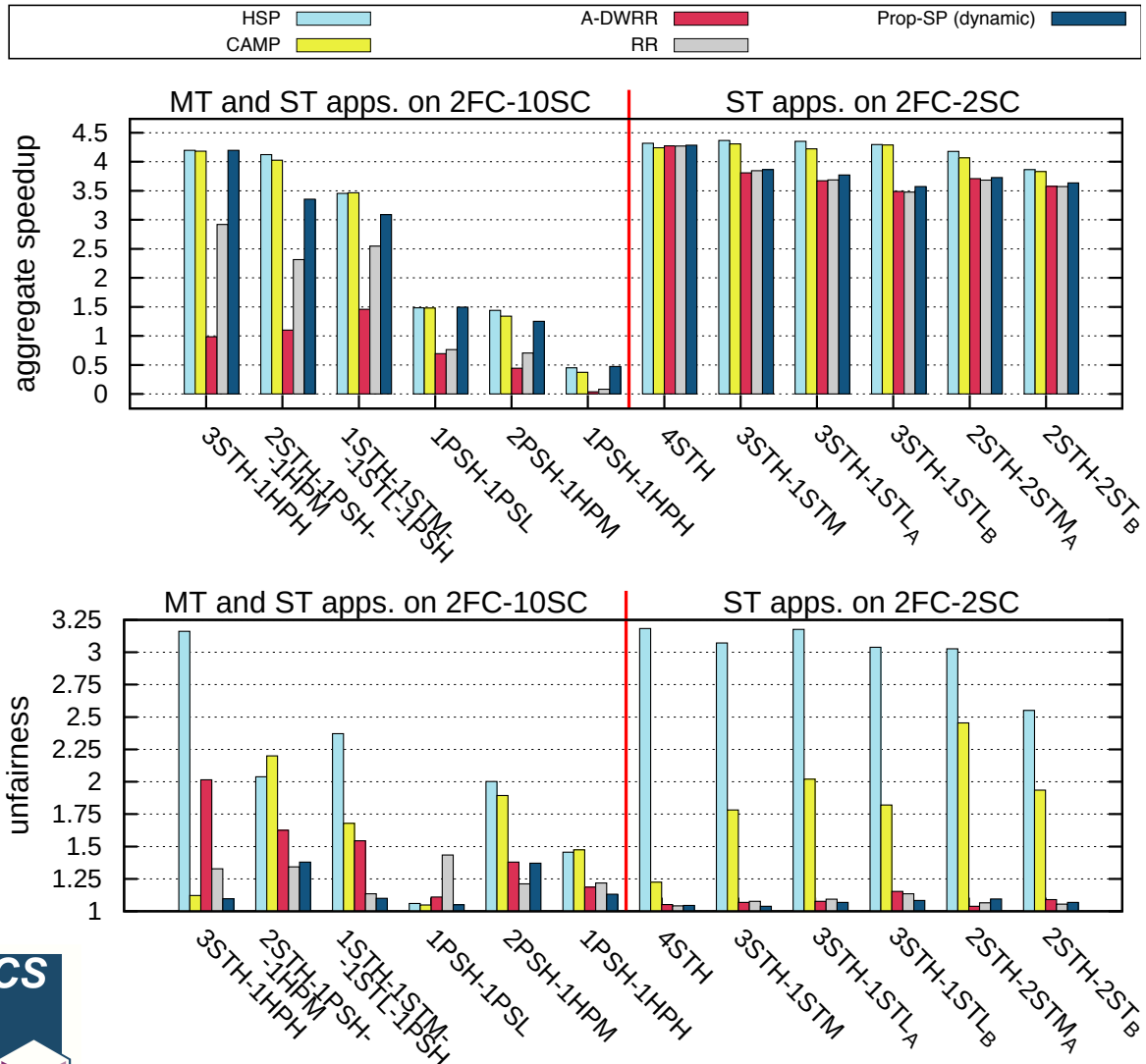


2FC-10SC



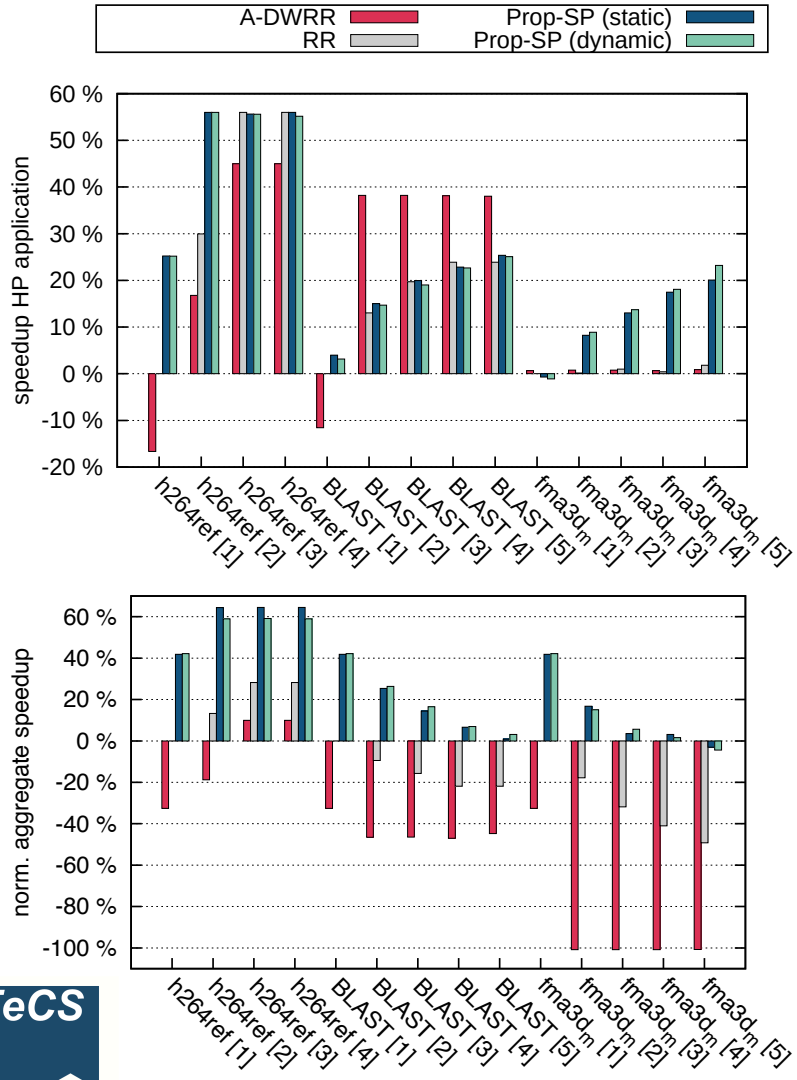


Equal-priority applications



- HSP/CAMP deliver the worst unfairness values
- Fair-sharing big cores (RR/A-DWRR) degrade the system thoughput
- Benefits of Prop-SP are especially pronounced when MT apps. included in the workload

Applications with different priorities



- Scenario:
  - 3 apps. running on 2FC-10SC (1 ST + 2 MT)
  - Select a high-priority app. (HPA)
  - Gradually increase HPA's prio.

- When HPA's prio grows $\rightarrow$ Prop-SP reduces HPA's completion time
- Not always the case for A-DWRR and RR

Outline

ArTeCS

Conclusions

- Our analytical and experimental evaluation reveal:
  1. Existing schemes aimed to optimize throughput are inherently unfair
  2. RR and A-DWRR degrade the system throughput and often exhibit unfair behavior
- Prop-SP ensures fair use of the AMP while achieving a better throughput-fairness tradeoff than RR and A-DWRR
  - It also makes an effective priority-based scheme
- Prop-SP' Key design aspects
  1. Evens out applications' *slowdowns* based on speedups
  2. The scheduler determines the *speedup* at runtime
  3. Specific support for multithreaded applications

Questions

Mitigating migration overheads

- Thread swaps (migrations) introduce overheads
  - We can enforce a target average migration rate by adjusting the length of each EP
  - # of swaps in the next EP can be predicted at runtime
    - Higher swap count $\rightarrow$ longer EP

Credit-distribution schemes

Available Schemes for multi-threaded applications

1. **Even** $\rightarrow$ Credits evenly distributed among runnable threads
   - Beneficial for coarse-grained parallel applications
2. **BusyFCs** $\rightarrow$ A few threads receive all available credits
   - Beneficial for several application types:
     - Applications with substantial serial fraction
     - Fine- and Mid-grained parallel applications
     - Regular OpenMP applications combined with the `dynamic` schedule
     - Job-stealing based applications (e.g.: Cilk)
   - Reduces migration overheads
     - Some threads may receive no fast-core credits

Multithreaded applications

Additional support

1. Fast-core credits allotted to the application must be distributed among runnable threads
   - Several distribution schemes available
     - Application performance is sensitive to the distrib. choice
   - In some cases OS-runtime interaction may be in order
2. Speedup estimation via analytical models
   - Each distribution scheme comes with a different speedup formula $\rightarrow f(SF_{threads}, TLP, N_{FC})$
3. Threads may block due to synchronization
   - Unused credits of a thread become available to other application's threads
     - Enable acceleration of sequential phases