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

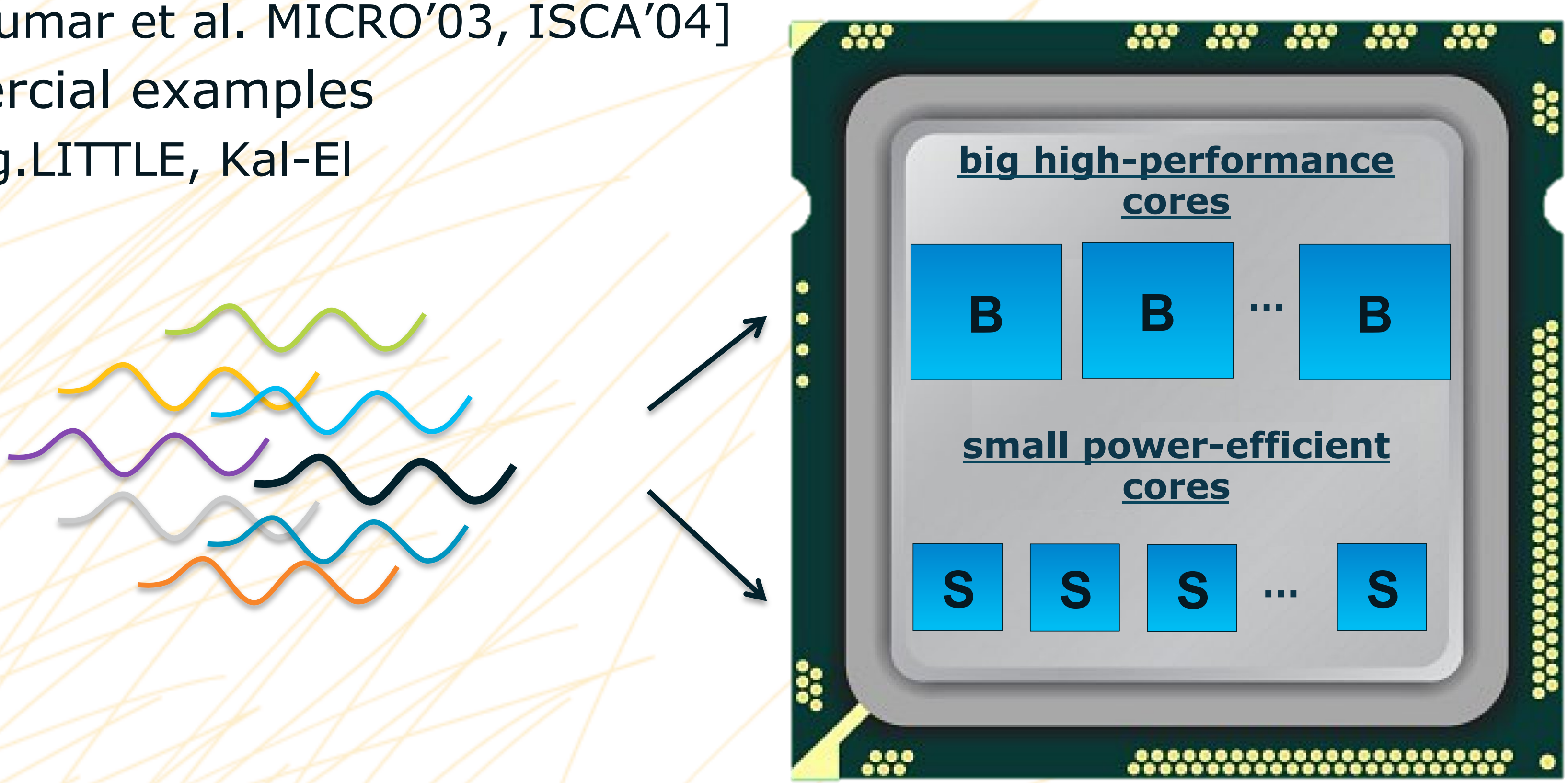
PACT 2013 - Edinburgh- September 11th 2013

Single-ISA heterogeneous multi-cores

Multiple core types
- representing different power/performance trade-offs

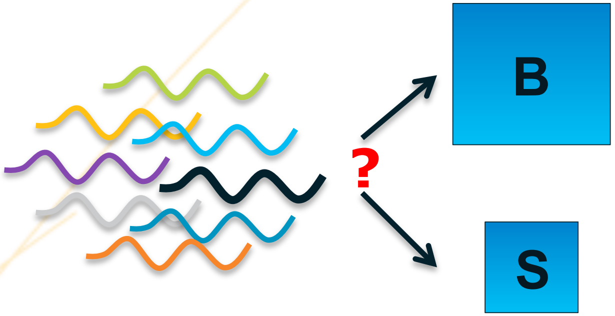
Well-established power benefits
- [Kumar et al. MICRO'03, ISCA'04]

Comercial examples
- Big.LITTLE, Kal-El



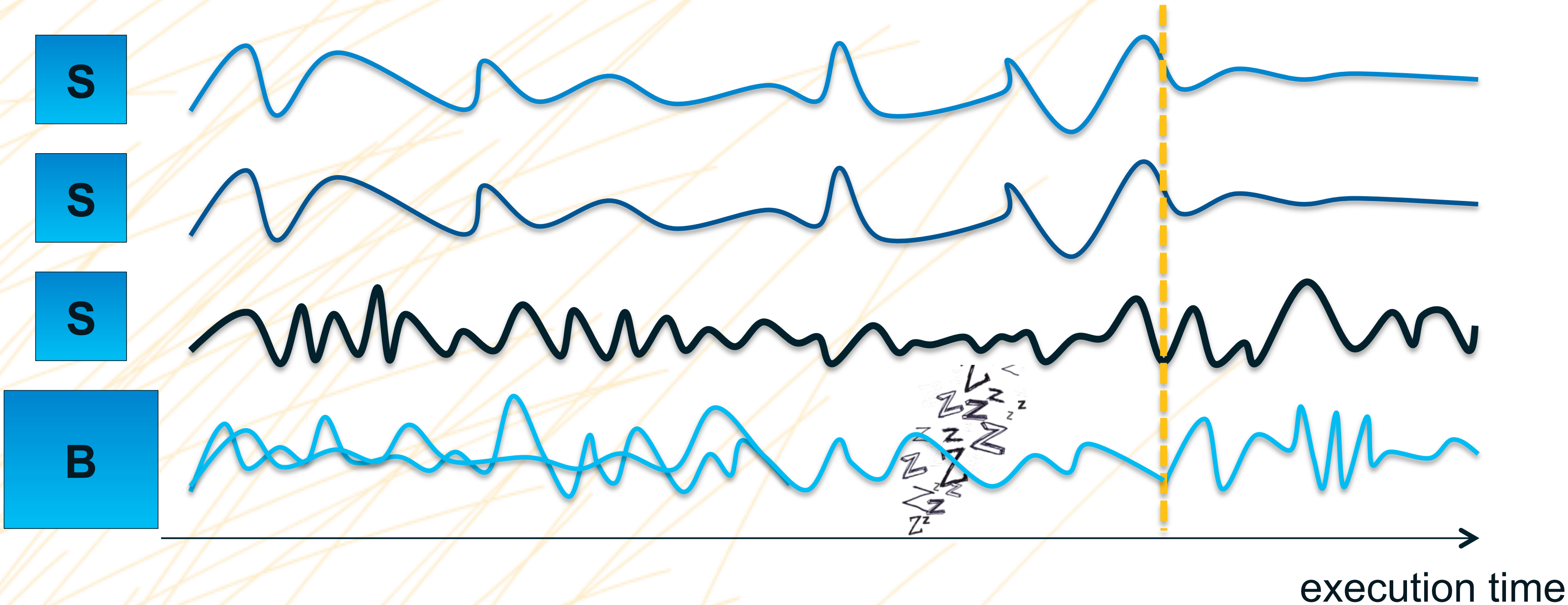
Prior Work: Put the Thread That Will Benefit the Most on the Big Core

Many different scheduling techniques

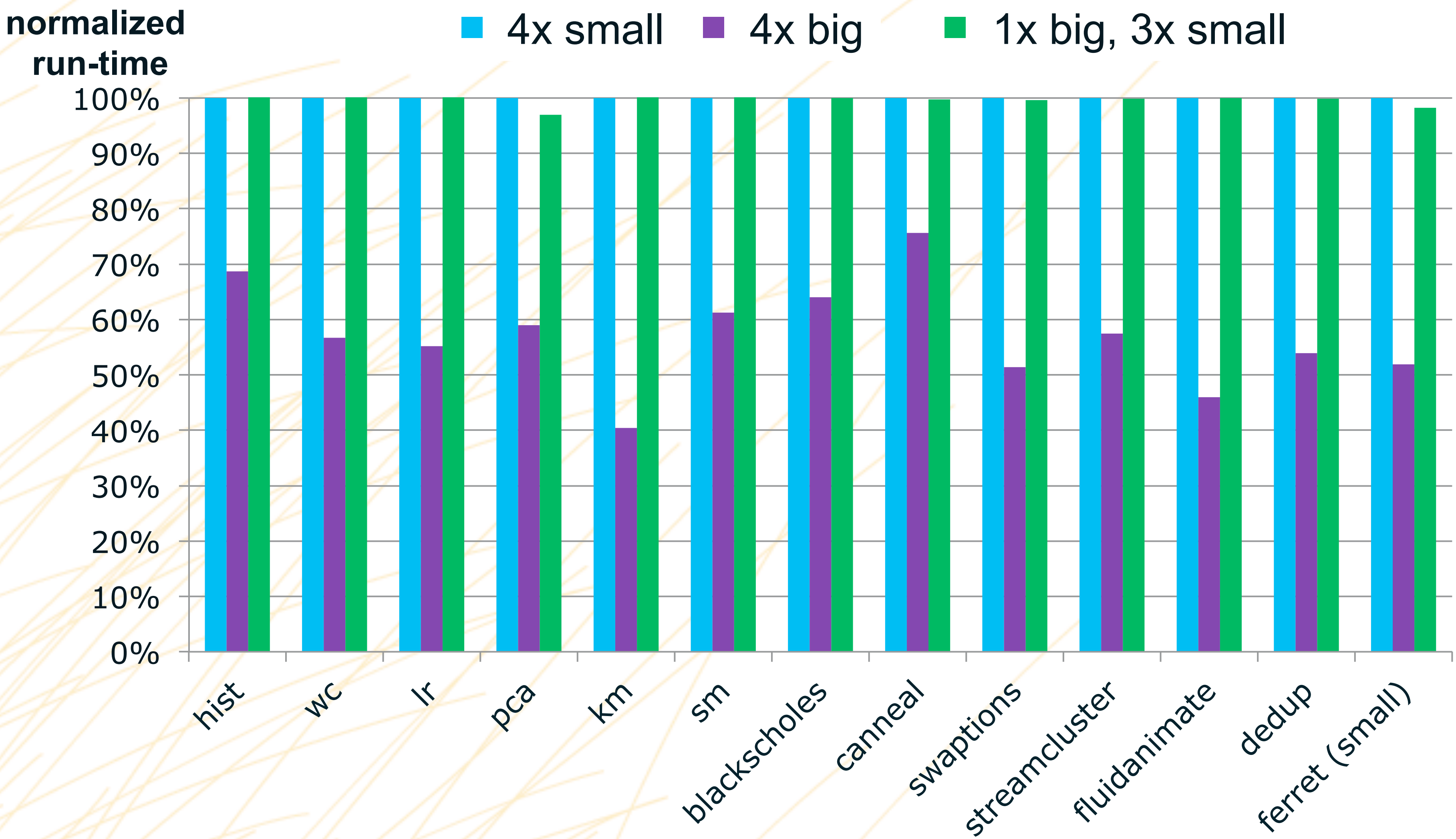


- Static scheduling
  - *Chen and John, DAC'08*
- Sampling-based scheduling
  - *Kumar et al., ISCA'04; Patsilaras et al., TACO'12*
- Proxies for performance
  - *Memory-domance (Becchi et al., JILP'08; Koufaty et al., EuroSys'10; Shelepov et al., OS Review'09)*
  - *Age-based Scheduling (Lakshminararayana et al., SC'09)*
- Model-based scheduling
  - *Van Craeynest et al., ISCA'12; Lukefahr et al., MICRO'12*

Traditional Scheduling can be Suboptimal



Threads pinned on Small Cores Determine Performance



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*

Fairness for Heterogeneous Multi-Cores

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

$$slowdown = S_i = T_{het,i} / T_{big,i}$$

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

Schedule is fair if slowdown of all running threads is the same

$$fairness = 1 - c_S = 1 - \sigma_S / \mu_S = 1 - std\_dev(S) / avg(S)$$

Coefficient of variation, a measure of unfairness

Experimental Setup

Simulated hardware

	small	big
issue width	4-wide	
clock frequency	2.6 GHz	
cache hierarchy	32KB (p) / 256 KB (p)/ 16MB (s)	
µarch	in-order	out-of-order

Sniper:

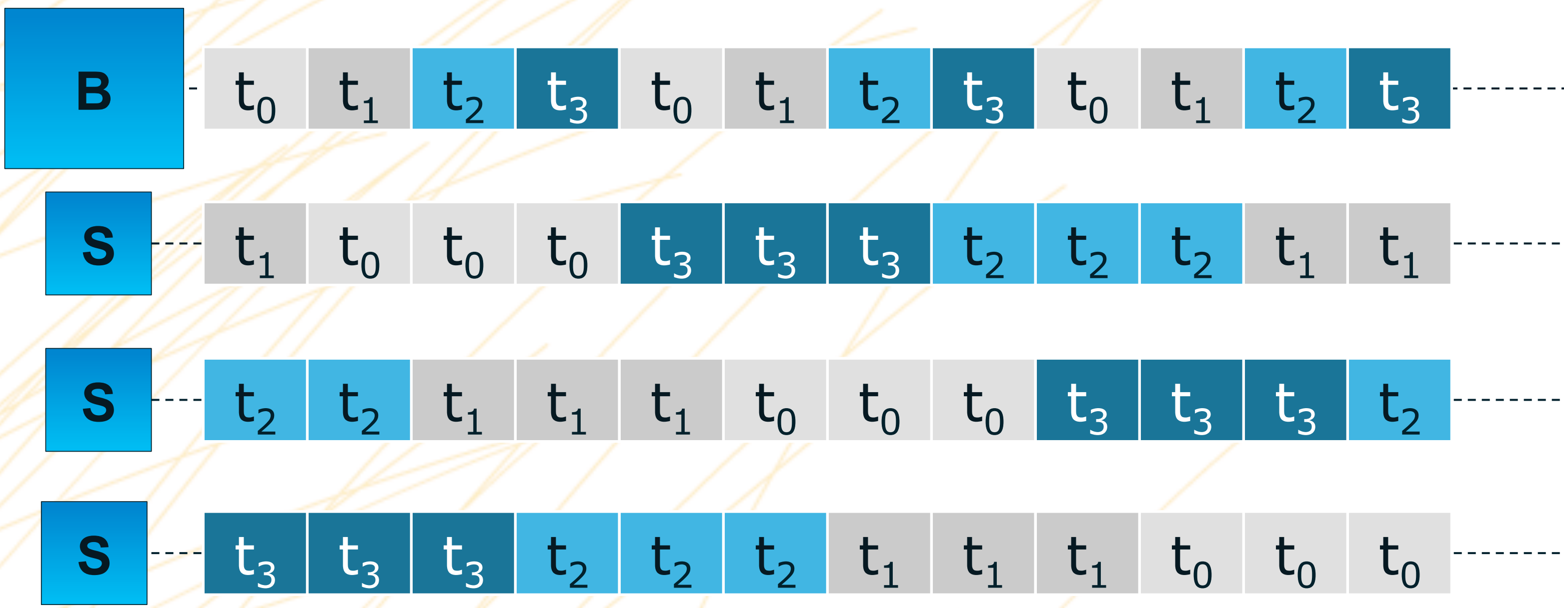
- parallel, hardware-validated x86-64 multi-core simulator

Multi-threaded and multi-programmed workloads

- spec2006, PARSEC and MapReduce

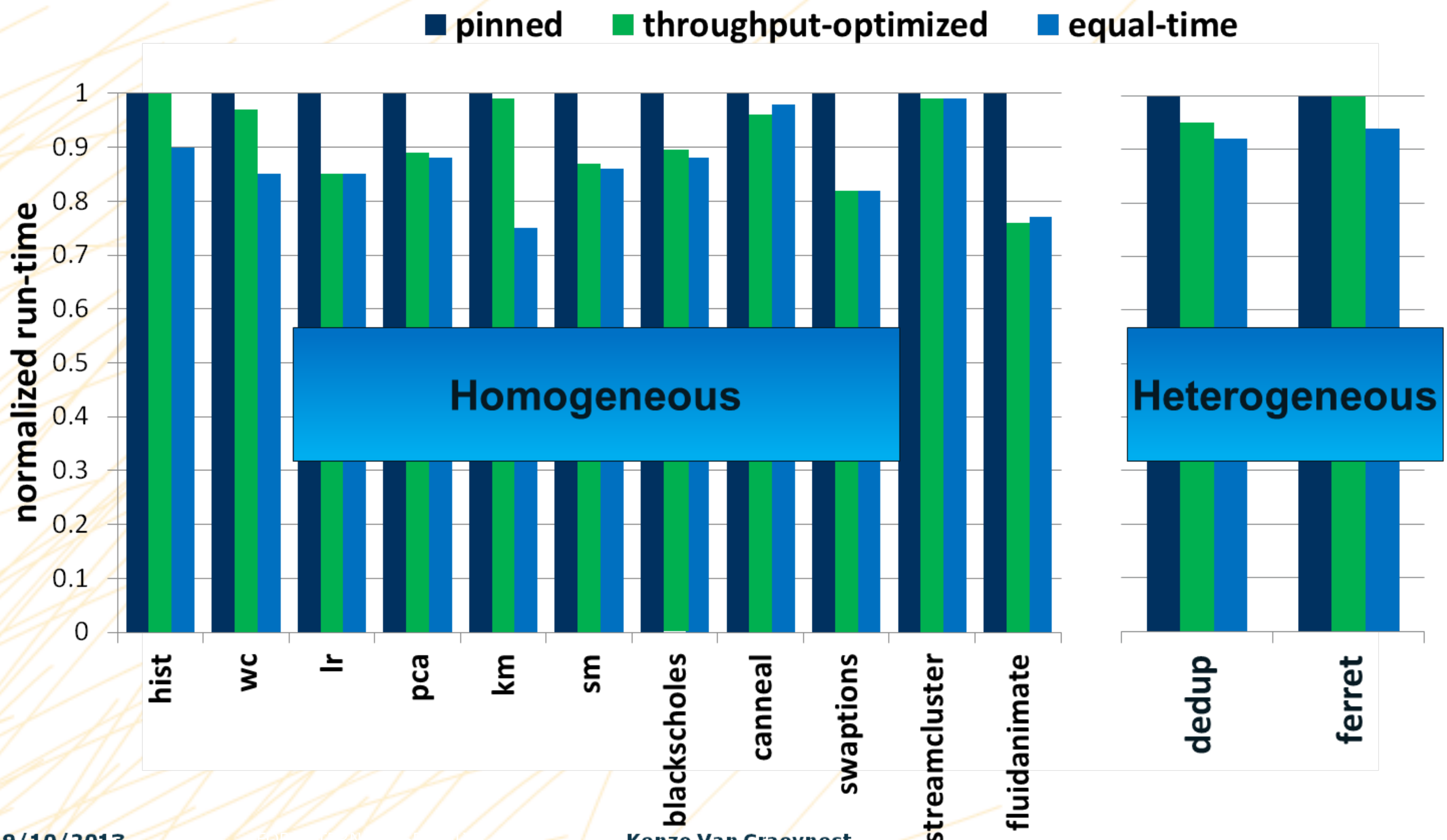
Achieving Fairness: Equal-time Scheduling

- Each thread runs for 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

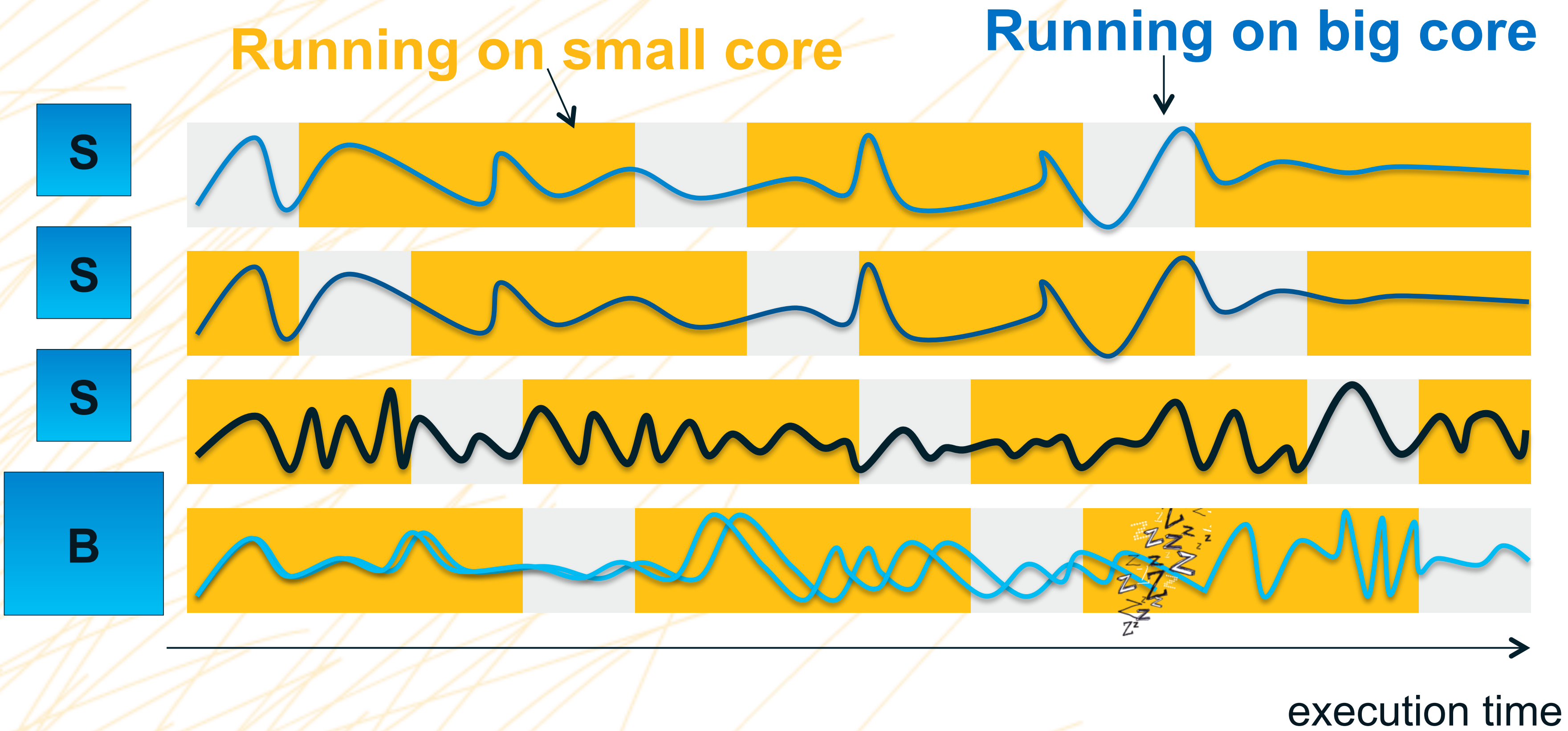
1B3S system



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 <u>not</u> guaranteed



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

$$S_i = \frac{T_{het,i}}{T_{big,i}} = \frac{T_{big,i} + T_{small,i}}{T_{big,i} + T_{small,i}/\boldsymbol{R_i}}$$

Overall slowdown of the thread

Scale execution time on small core

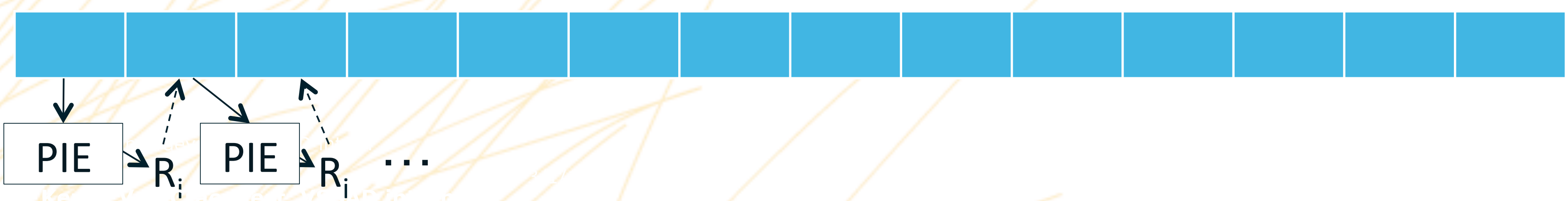
Performance ratio between big and small core

Estimating the Performance Ratio

- Proposed 3 methods
  - sampling-based

  - history-based

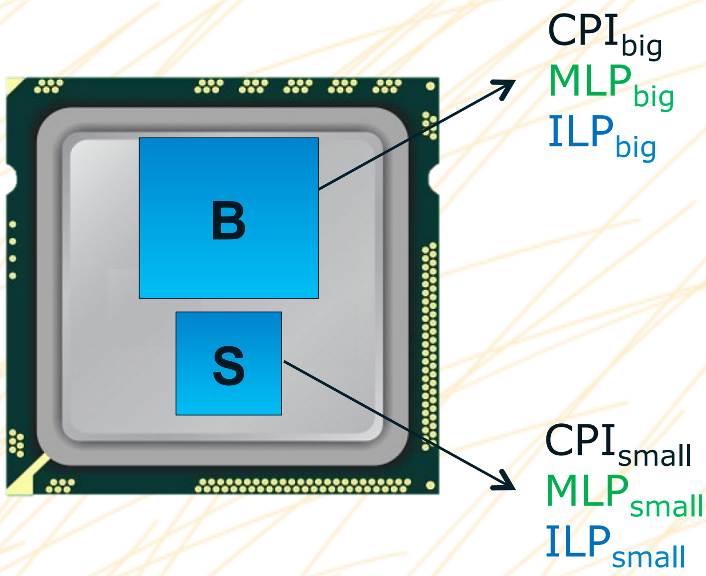
  - model-based

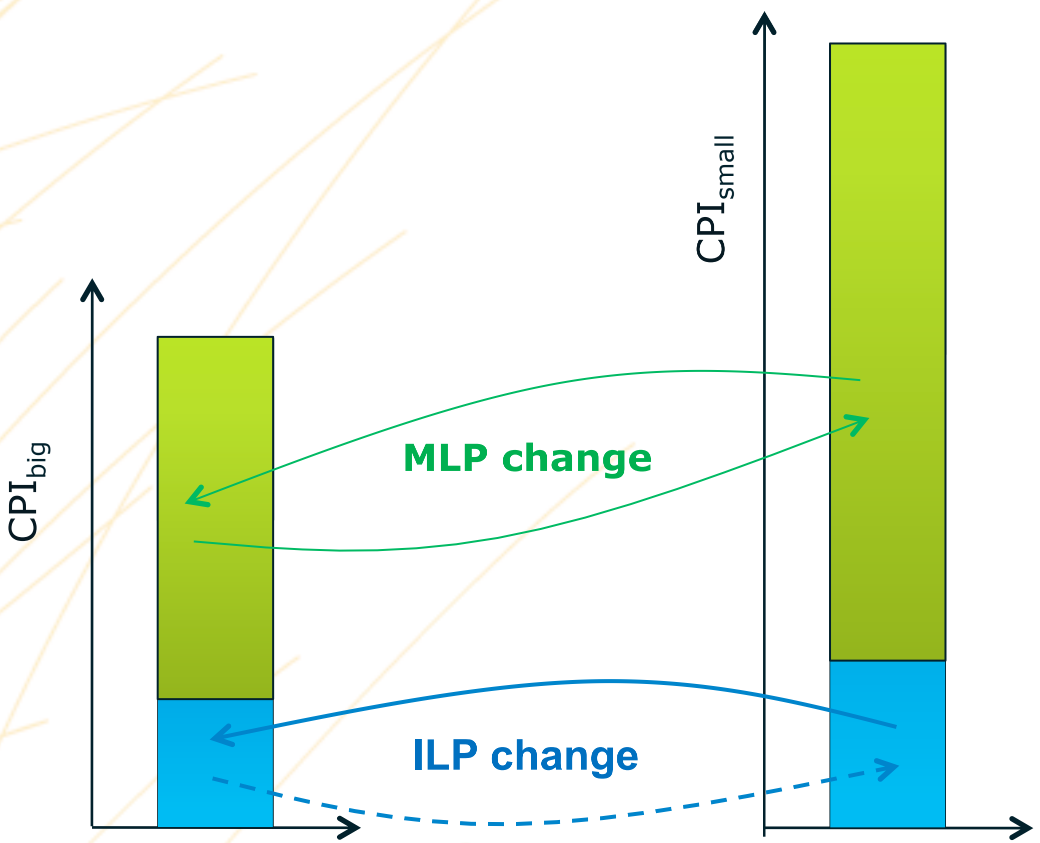


Performance Impact Estimation (PIE)

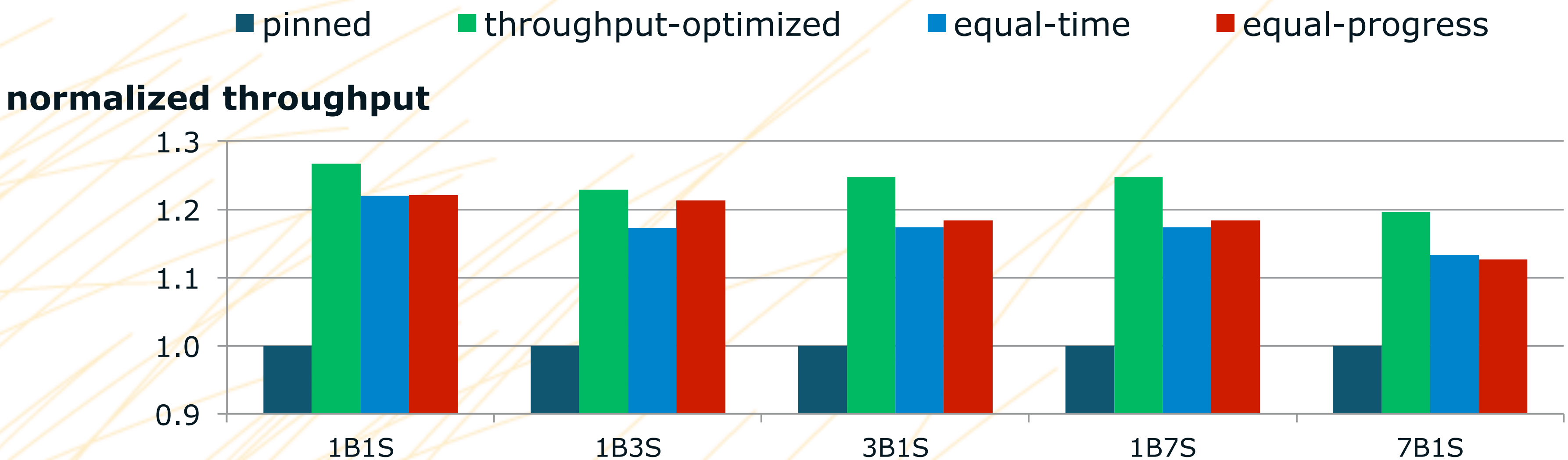
[Van Craeynest et al., ISCA'12]

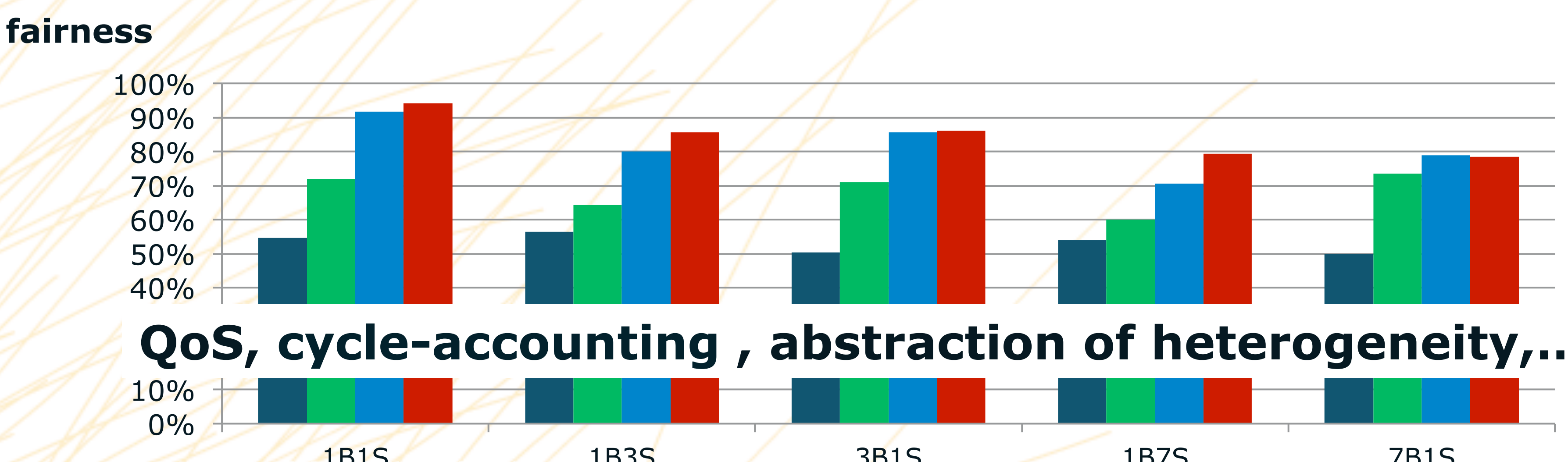
1. Determine where application spends its execution time
2. Use change in MLP exposed to predict change in CPI_{mem}
3. Use change in ILP exposed to predict change in CPI_{base}





Fairness-aware Scheduling Across Configurations for Multi-Programmed Workloads

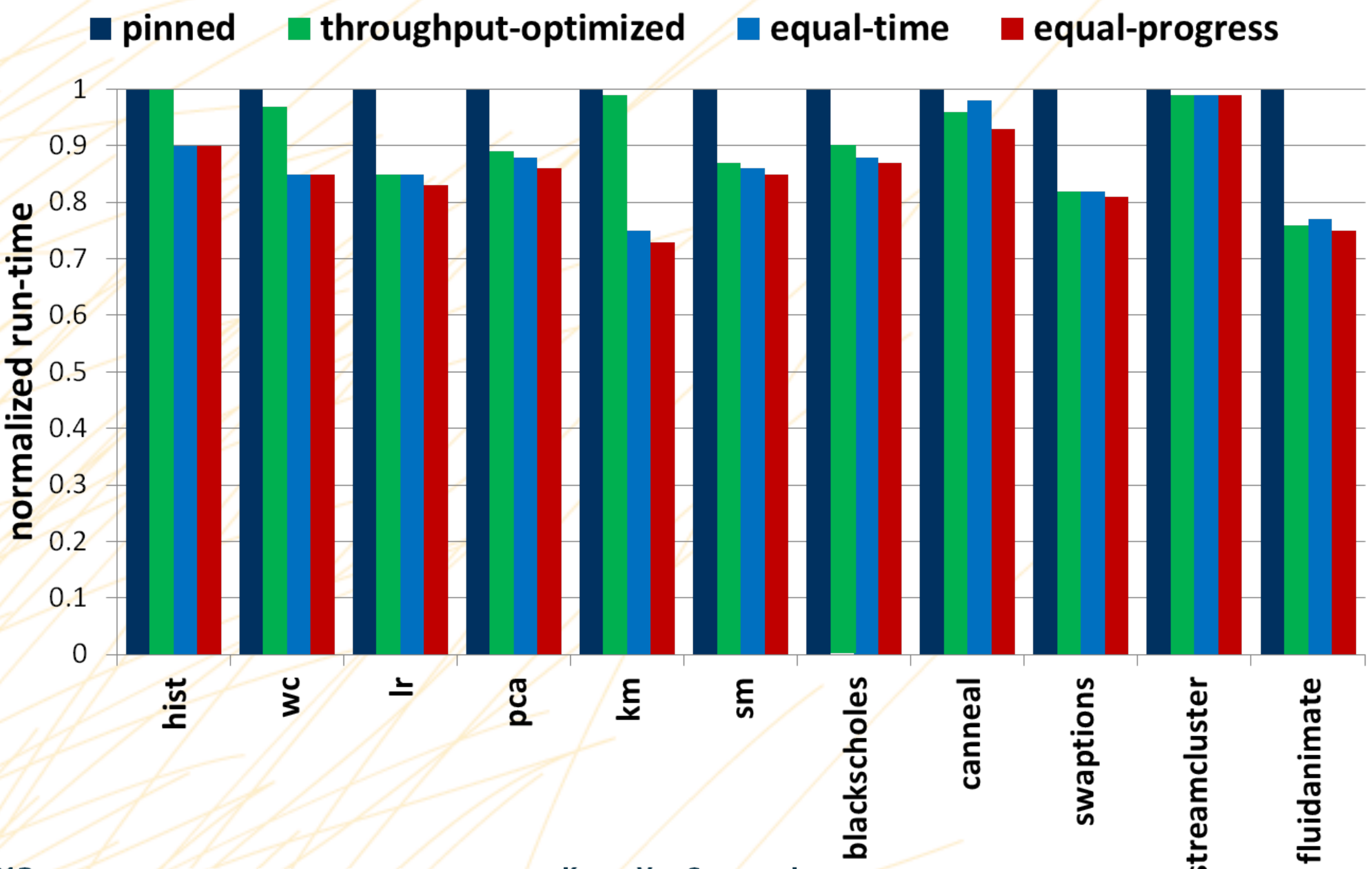




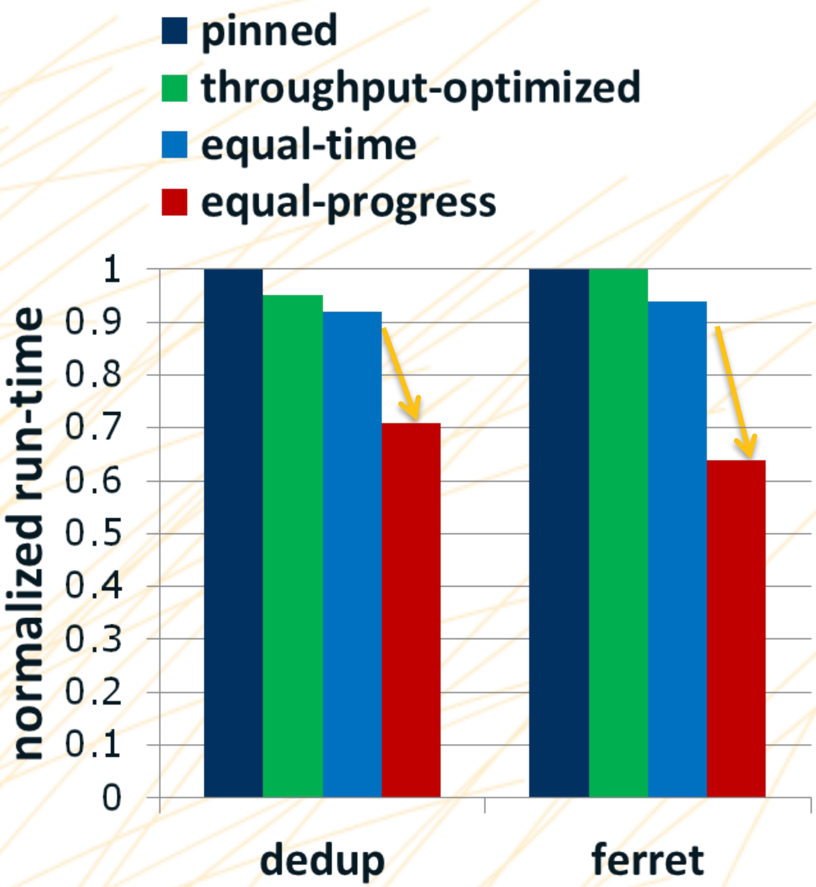
QoS, cycle-accounting , abstraction of heterogeneity,…

Optimizing Fairness Reduces Run-time for Homogeneous Multi-Threaded Workloads

1B3S system



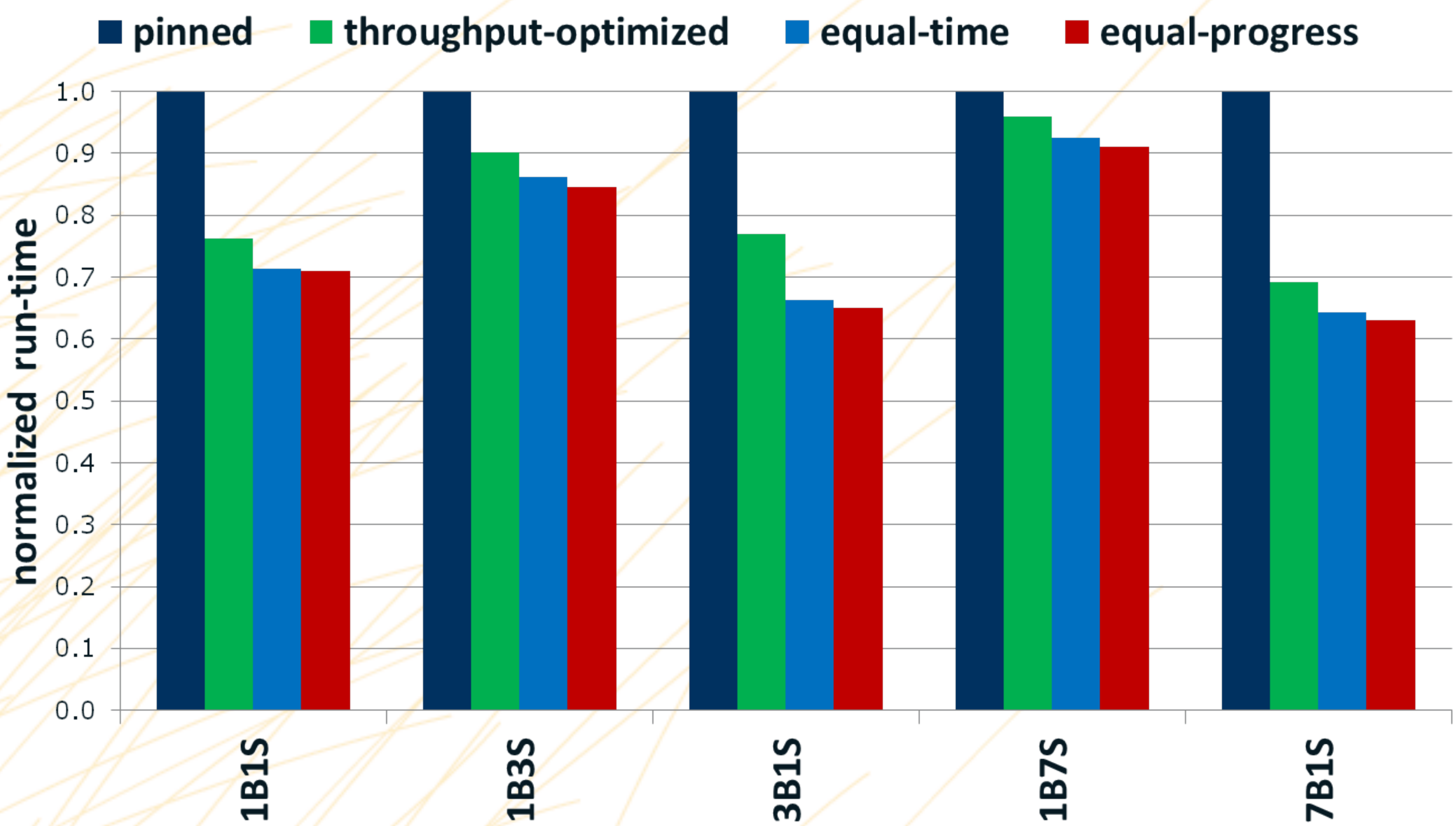
Optimizing for Fairness Reduces Run-time for Heterogeneous Multi-Threaded Workloads



1B3S

- Heterogeneous applications
  - Threads can have different performance ratio
  - Equal-time scheduling does not result in a fair schedule
- Equal progress 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



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?