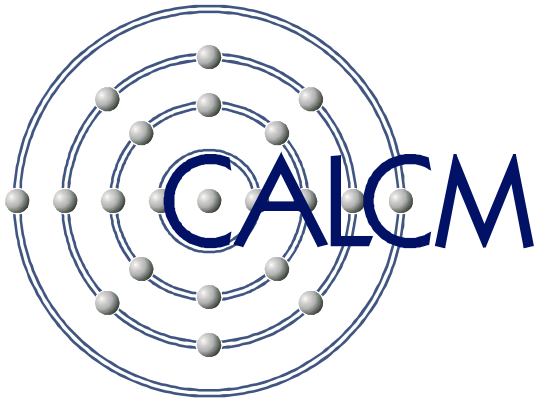
Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems

**Eiman Ebrahimi, Chang Joo Lee,
Onur Mutlu, Yale N. Patt**

Presented by Michael K. Papamichael



Computer Architecture Lab at
Carnegie Mellon

March 3, 2010

Motivation

- **CMP apps contend in shared mem. subsystem**
  - for last-level cache space
  - for DRAM memory bandwidth
- **Existing fairness mechanisms resource-based**
  - can lead to contradictory decisions
  - low fairness and loss of performance
- **Need single mechanism for fairness at all levels**
  - Make decisions in tandem in the entire mem. System
  - Support thread priorities/weights

Goal
(as defined in paper)

"Our goal in this paper is to develop a low-cost architectural technique that allows system software fairness policies to be achieved in CMPs by enabling fair sharing of the entire memory system, without requiring multiple complicated, specialized, and possibly contradictory fairness techniques for different shared resources."

A little terminology

- **Slowdown**
  - T_{shared}/T_{alone} ,(where $T_{alone} = T_{shared} - T_{excess}$)
- **Unfairness**
  - Max Slowdown **/** Min Slowdown (out of running apps)
- **Fair Memory System**
  - Equal-priority apps have equal slowdowns
- **Mem intensiveness** (measured in Misses / 1K Insts)
  - **>10** MPKI: High memory intensity
  - **1-10** MPKI: Medium memory intensity
  - **<1** MPKI: Low memory intensity

Fairness via Source Throttling

Split time into intervals and during each interval:

1. **Gather dynamic info about unfairness in system**
2. **If unfairness > unfairness threshold**

   Adapt memory request injection rate of cores

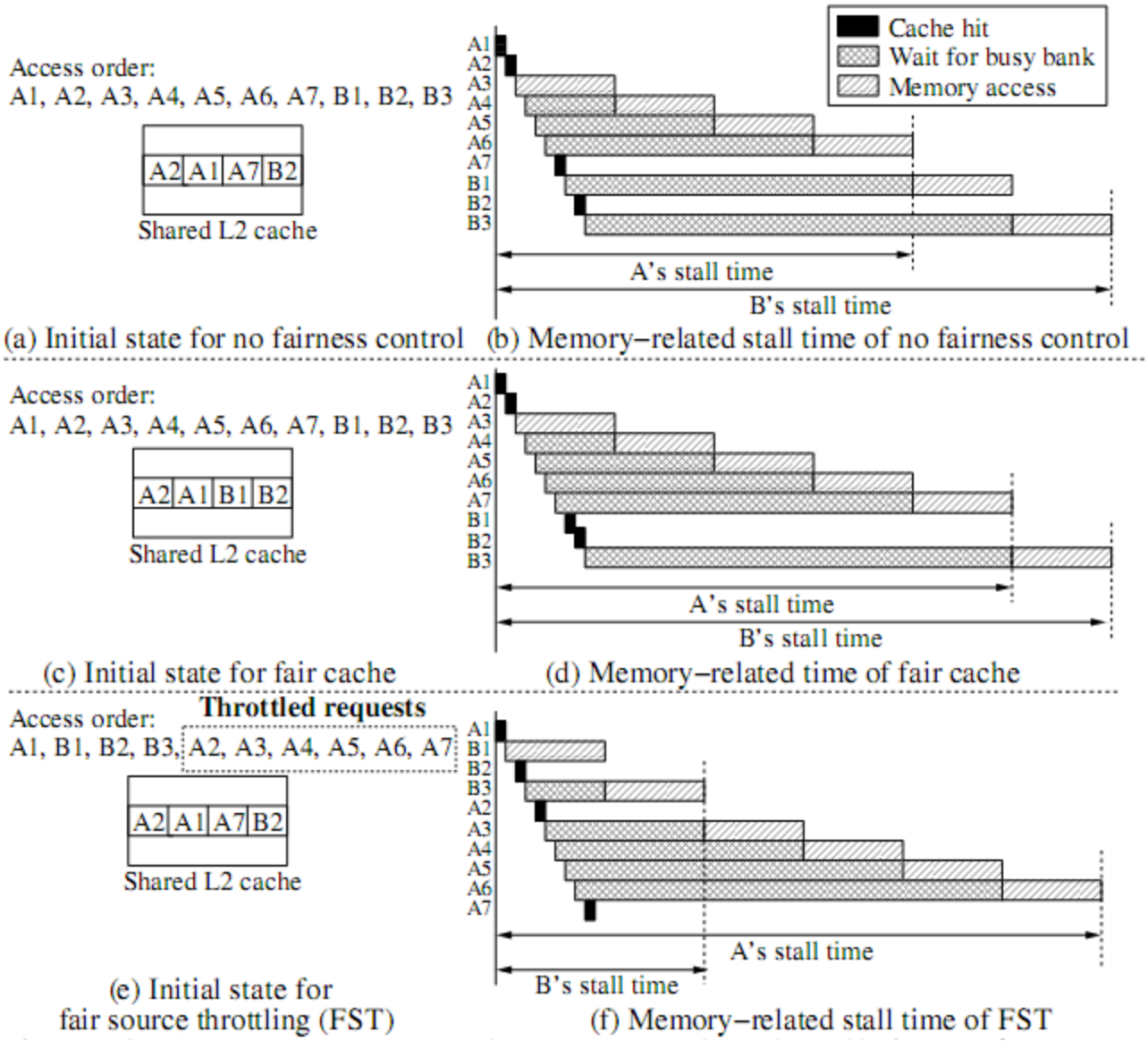
   – by reducing available MSHRs

   – by reducing the freq. of injecting requests in mem. system

3. **Else** (system fairness goal is met)

   Allow cores to throttle up to improve system throughput

Motivating Example



Interference Sources

- **Cache Interference**
  - Use pollution filter
- **DRAM Bus and Bank Conflict Interference**
- **DRAM Row-Buffer Interference**
  - Use Shadow Row-buffer Address Register
- **Preventing Denial of Bank Service**
  - Throttled-down app w/ high row-buffer locality can deny service to another app continuously
  - If interference above some threshold stop prioritizing interfering app based on row buffer hit status in MC

Workloads

- **SPEC CPU multi-programmed workloads**
  - 18 two-application workloads
  - 10 four-application workloads

Benchmark	Type	IPC	MPKI	Benchmark	Type	IPC	MPKI
art	FP00	0.10	90.89	milc	FP06	0.30	29.33
soplex	FP06	0.28	21.24	leslie3d	FP06	0.41	20.88
lbm	FP06	0.45	20.16	bwaves	FP06	0.46	18.71
GemsFDTD	FP06	0.46	15.63	astar	INT06	0.37	10.19
omnetpp	INT06	0.36	10.11	gcc	INT06	0.45	6.26
zeusmp	FP06	0.82	4.69	cactusADM	FP06	0.60	4.51
bzip2	INT06	1.14	2.61	h264ref	INT06	1.46	1.28
vortex	INT00	1.01	1.24	gromacs	FP06	1.06	0.29
namd	FP06	2.25	0.18	calculix	FP06	2.28	0.05
gamess	FP06	2.32	0.04	povray	FP06	1.88	0.02

Table 2: Characteristics of 20 SPEC 2000/2006 benchmarks: IPC and MPKI (L2 cache Misses Per 1K Instructions)

Metrics

- **Harmonic mean of Speedups** (Hspeedup)
  - Balanced measure between fairness and throughput
  - Hspeedup = $\frac{N}{\sum_i \frac{IPC_i^{alone}}{IPC_i^{shared}}}$
- **Weighted Speedup** (Wspeedup)
  - Wspeedup = $\sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}$
- **Unfairness**
  - $T_{shared} / (T_{shared} - T_{excess})$,where T_{excess} is estimated

Results (2-core workloads)

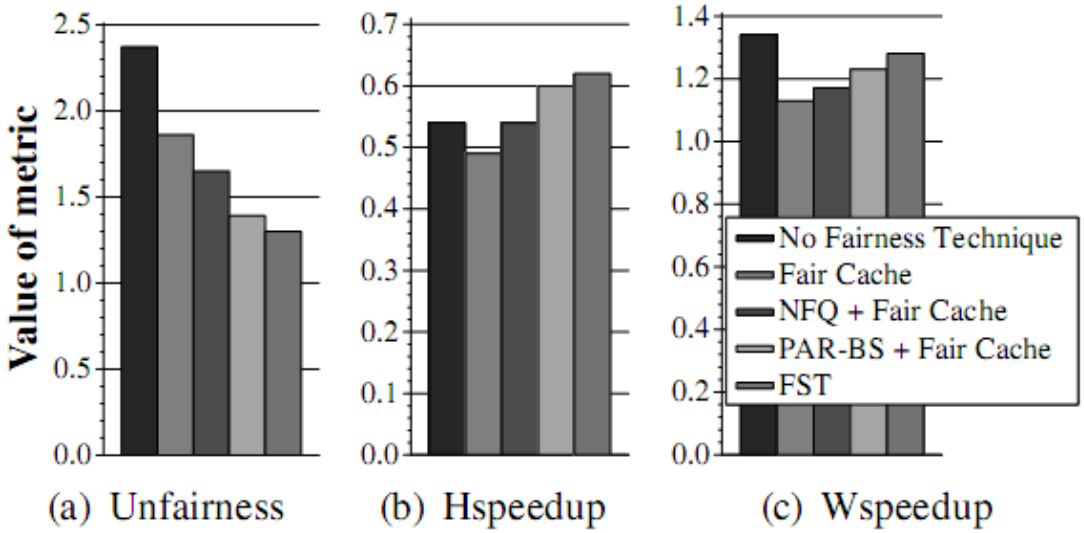


Figure 4: Average performance of FST on the 2-core system

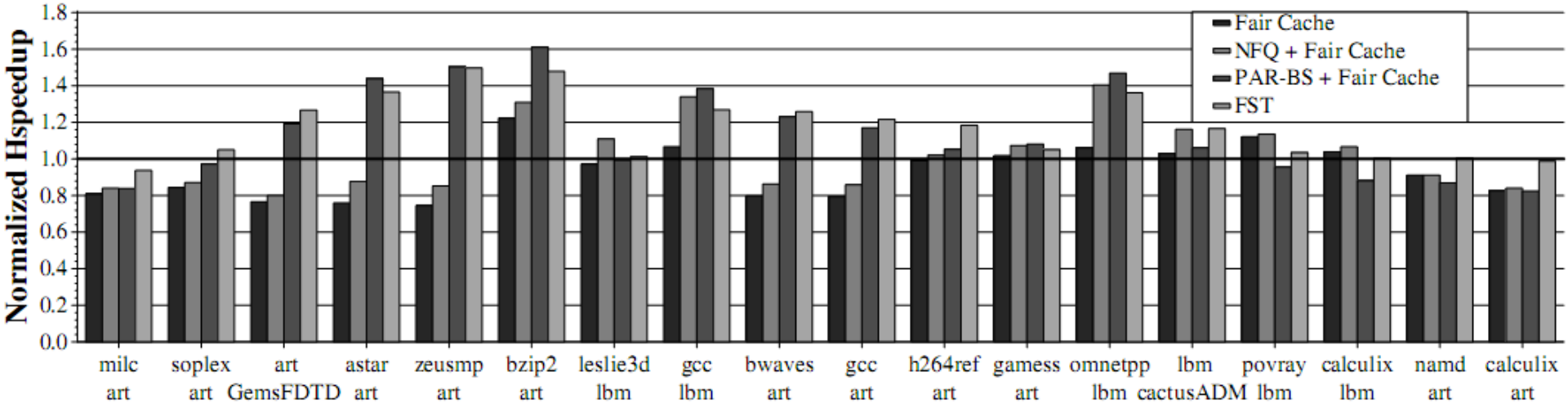


Figure 5: Hspeedup of 18 2-core workloads normalized to no fairness control

Results (4-core workloads)

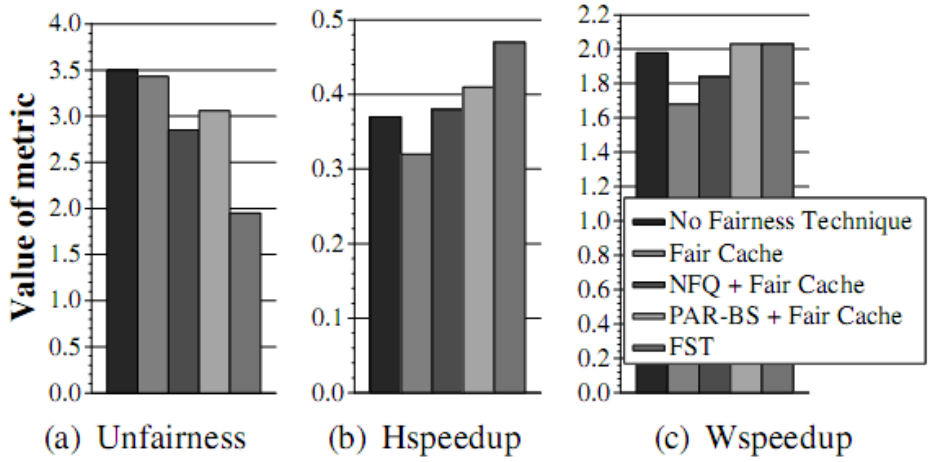


Figure 6: Average performance of FST on the 4-core system

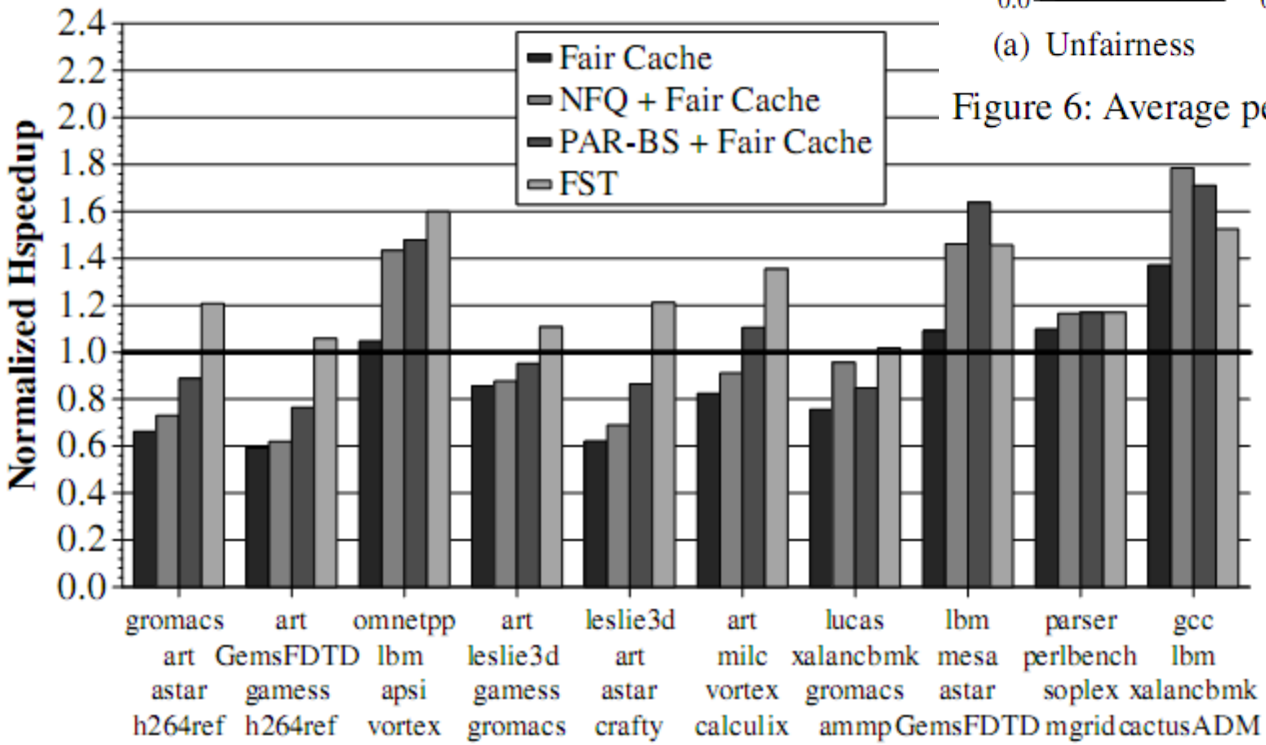


Figure 7: Normalized speedup of 10 4-core workloads

Results (Case Study)

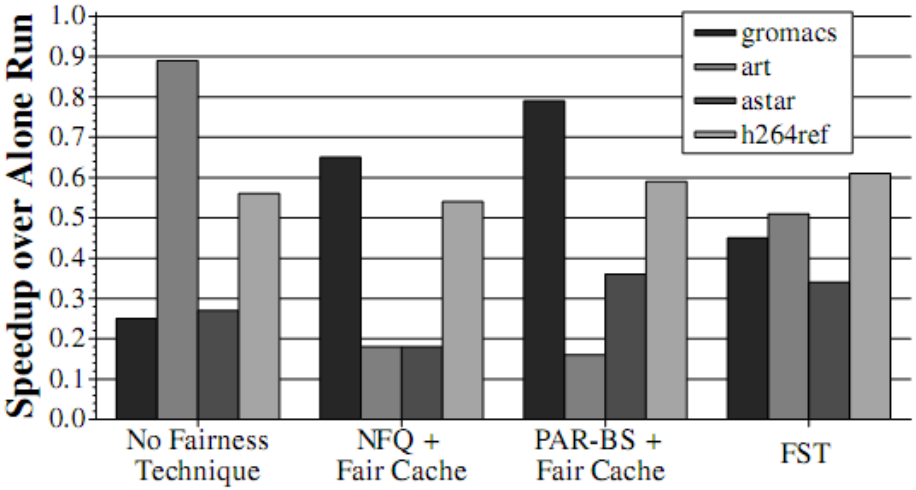
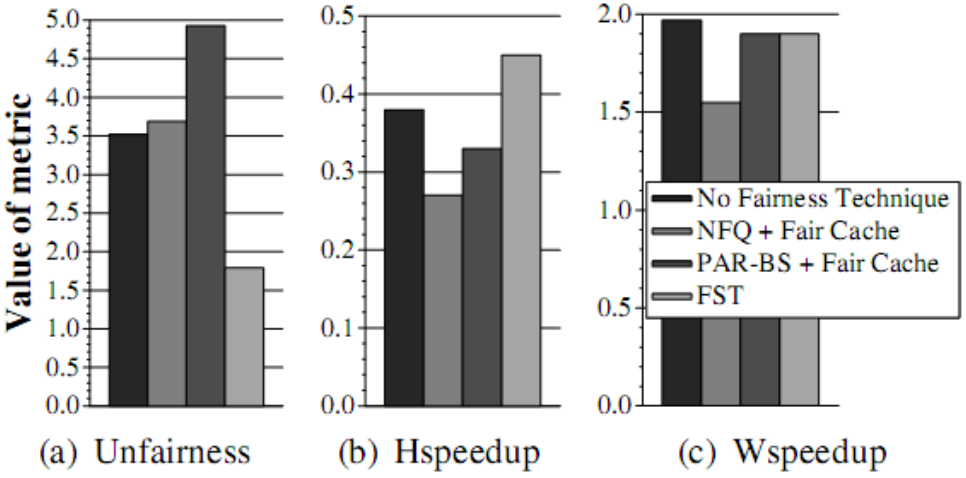


Figure 8: Case Study: individual application behavior



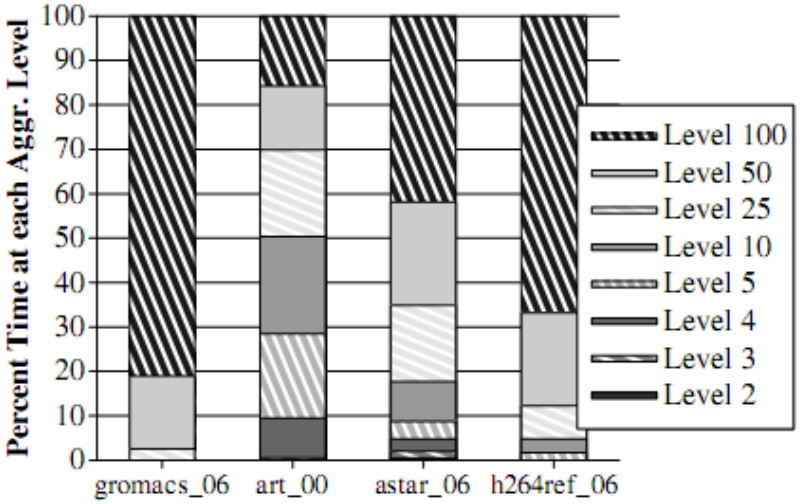
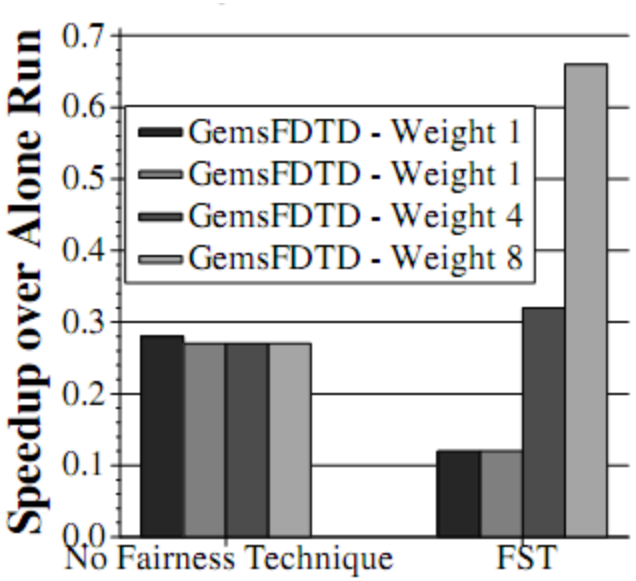


Figure 10: Case Study: application throttling levels

Results (Thread Weights/Priorities)



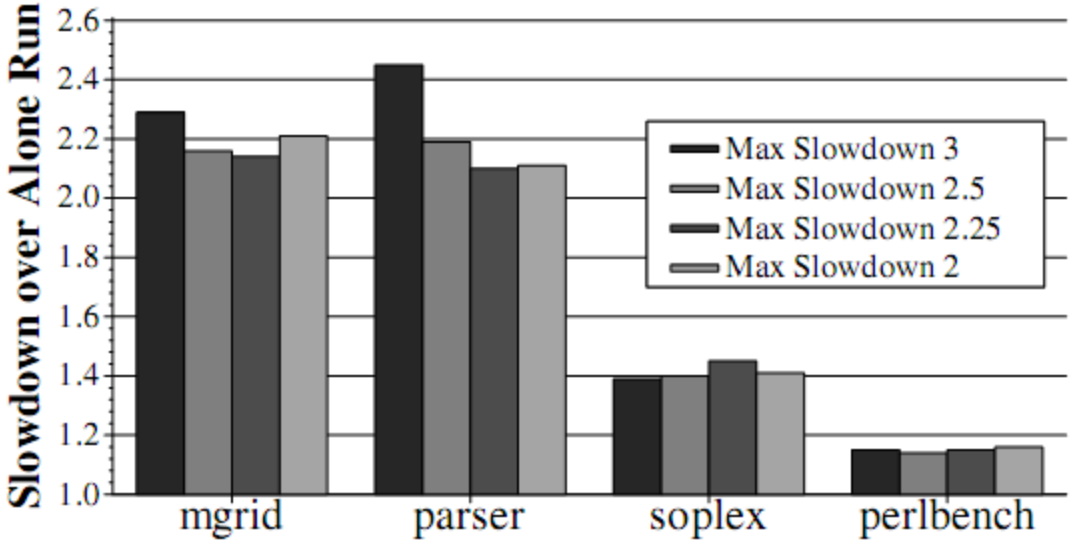


Figure 13: Enforcing maximum slowdown with FST

Conclusions/Comments

- Elegant, comprehensive solution
  - Simpler and better than previous work
  - Can replace multiple resource sharing mechanisms
  - Provides general fairness substrate & supports QoS
- Performance under other configurations/workloads
  - Needs to be evaluated against shared mem. workloads
  - >1 apps per core?
- Scalability?
  - Interference tracking HW scales by $O(N^2)$