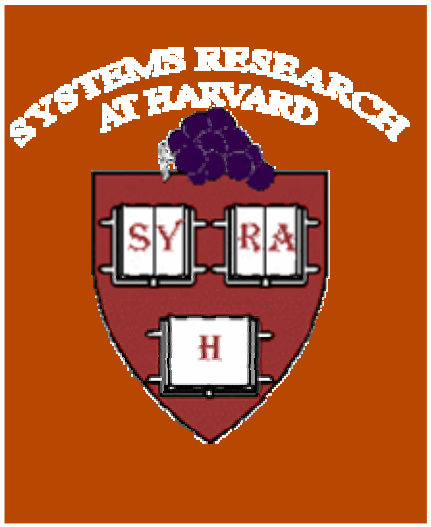
Chip-Multithreading Systems Need A New Operating Systems Scheduler

Alexandra Fedorova — Harvard University, Sun Microsystems
Christopher Small — Sun Microsystems
Daniel Nussbaum — Sun Microsystems
Margo Seltzer — Harvard University



Introduction

Chip Multithreading = Chip Multiprocessing + Hardware Multithreading

- Several processors on a physical dye
- Several hardware threads on a processor

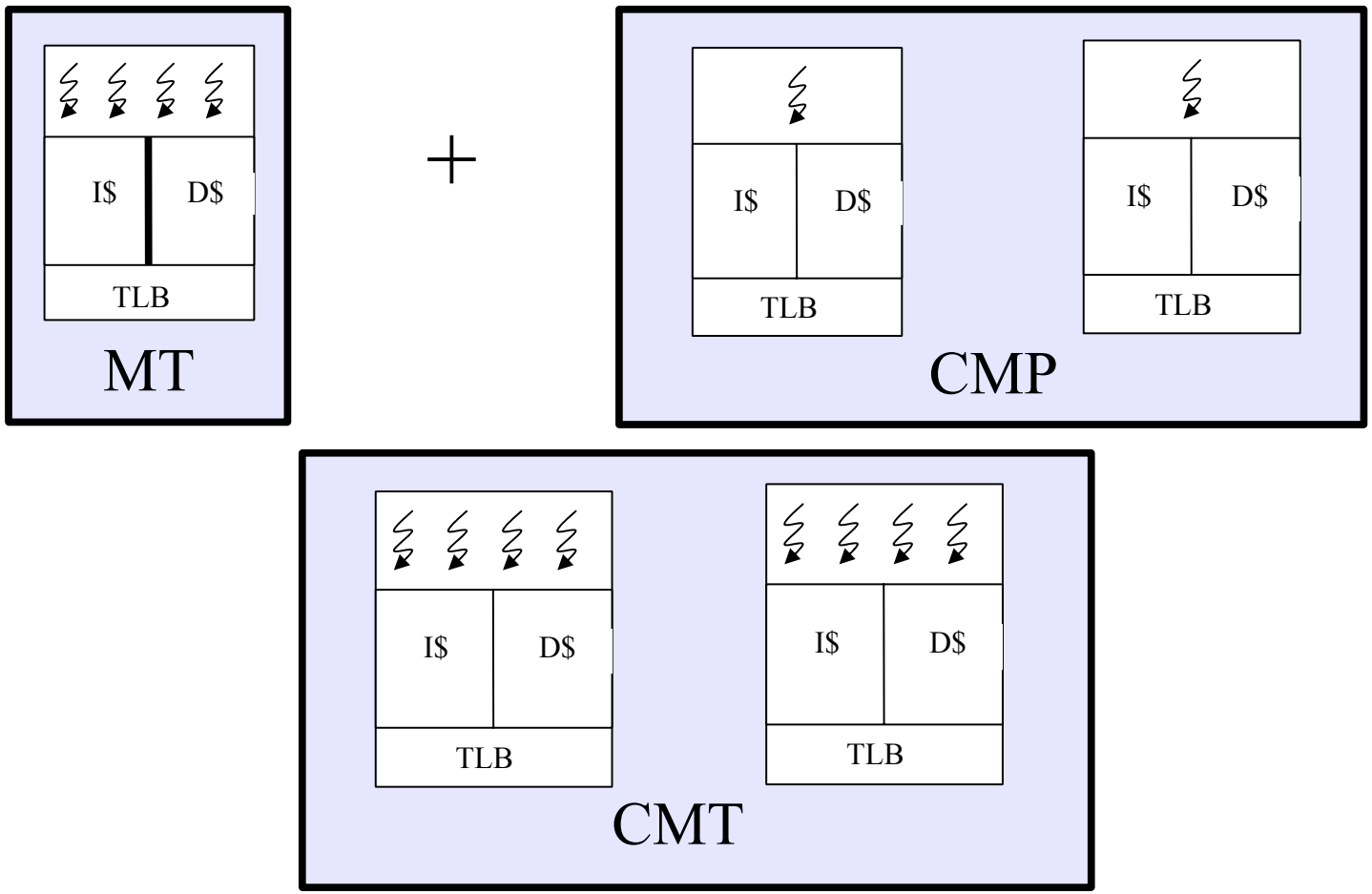
- Lots of contention for processor pipeline

Can we use operating system scheduling to reduce contention?

Outline

- Chip Multithreading Processors
- CPI-based scheduling
- Performance Results
- Discussion

Chip Multithreading Processor

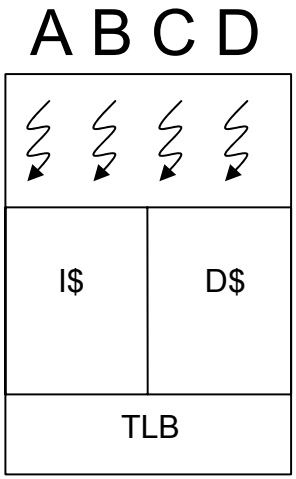


Why CMT?

- Modern workloads achieve pipeline utilization of < 20% on modern super-scalar OOO processors
- Why? They block on memory.
- CMT processors hide latency: while a thread context is blocked on memory, others can proceed.

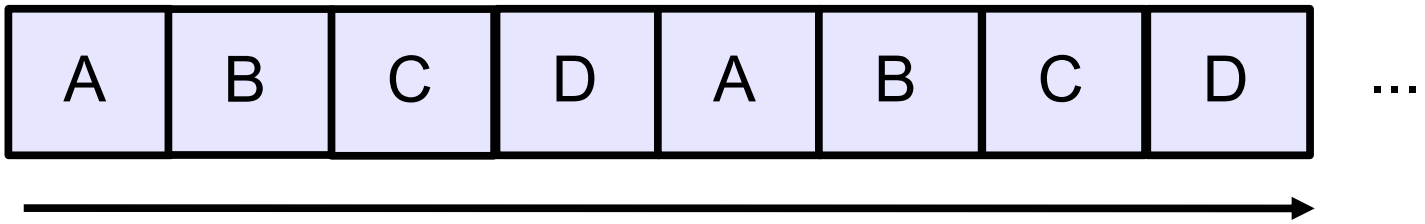
Having multiple thread contexts means that there is less space for other resources, and more contention for them.

Sharing Processor Pipeline



Switch among threads on every cycle (interleaving)

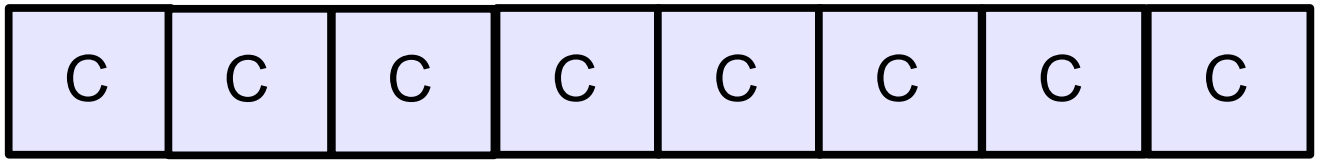
Pipeline state



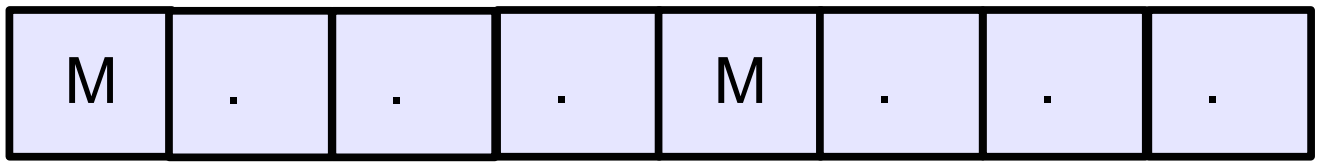
time

Different kinds of workloads

Compute thread: can issue on every cycle

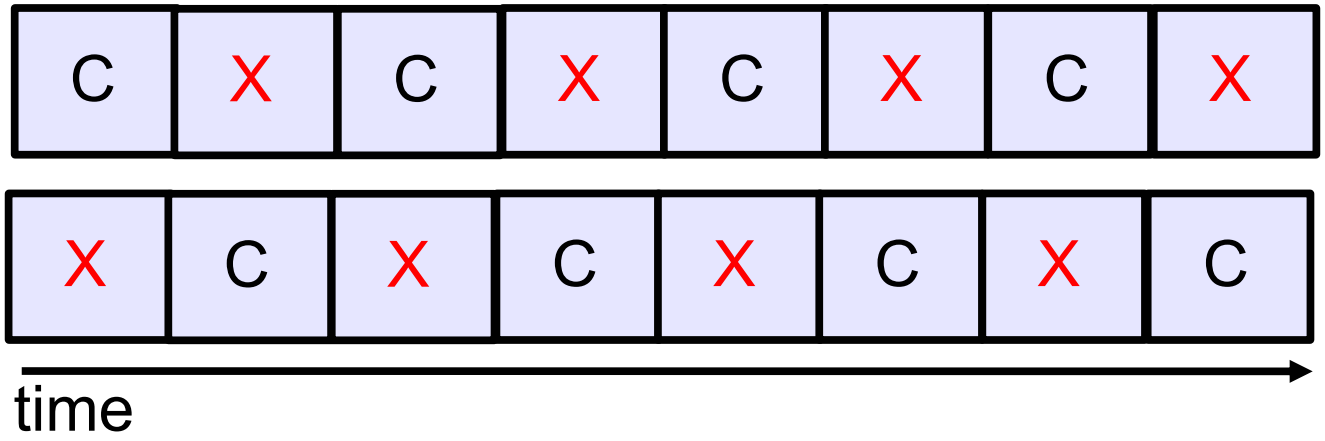


Memory thread: often blocks



Co-scheduling: option 1

Co-schedule compute threads

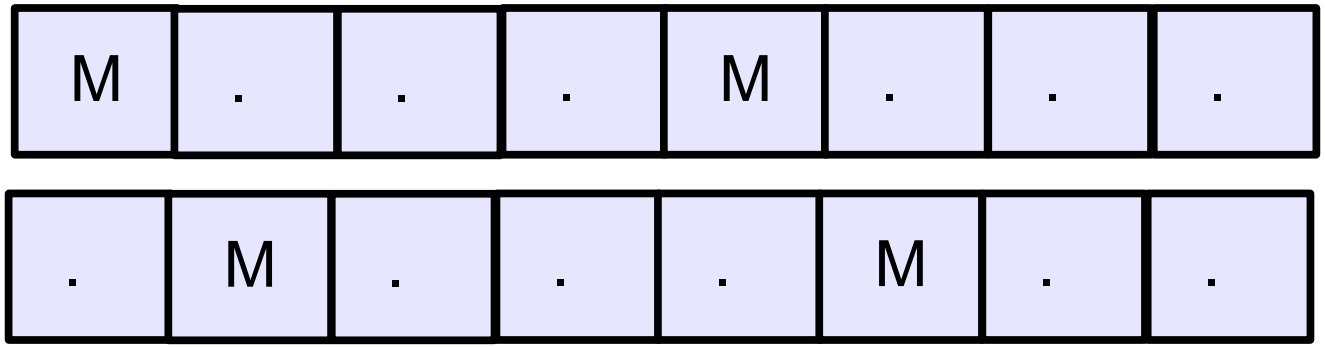


C – compute
X – idle (waste time)

Results in slowing down threads!

Co-scheduling: option 2

Co-schedule memory threads

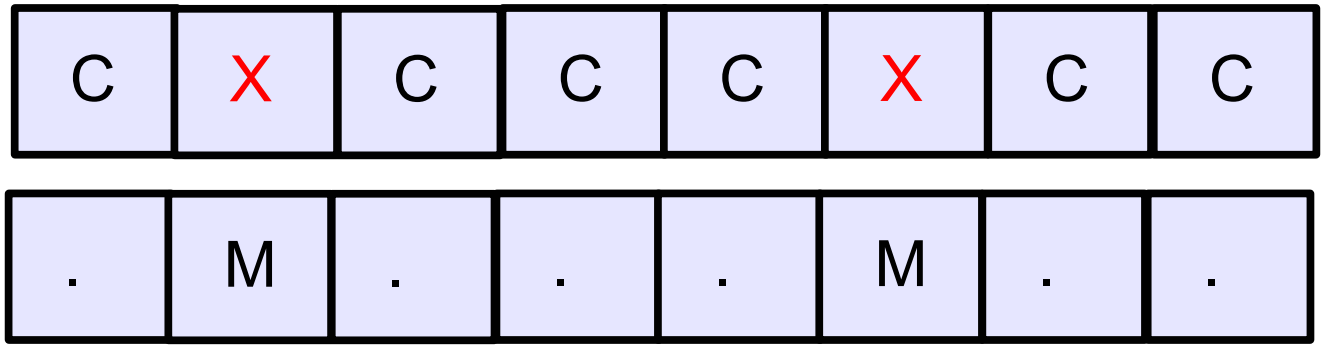


M – issue a memory operation
. – wait on memory

Results in wasting pipeline resources!

Bingo!

Co-schedule memory and compute threads



CPI-based Scheduling

- Use threads' cycle per instruction (CPI) to identify compute-bound and memory-bound threads
- Low CPI – compute bound
- High CPI – memory bound

Pair up low-CPI threads with high-CPI threads on the same core.

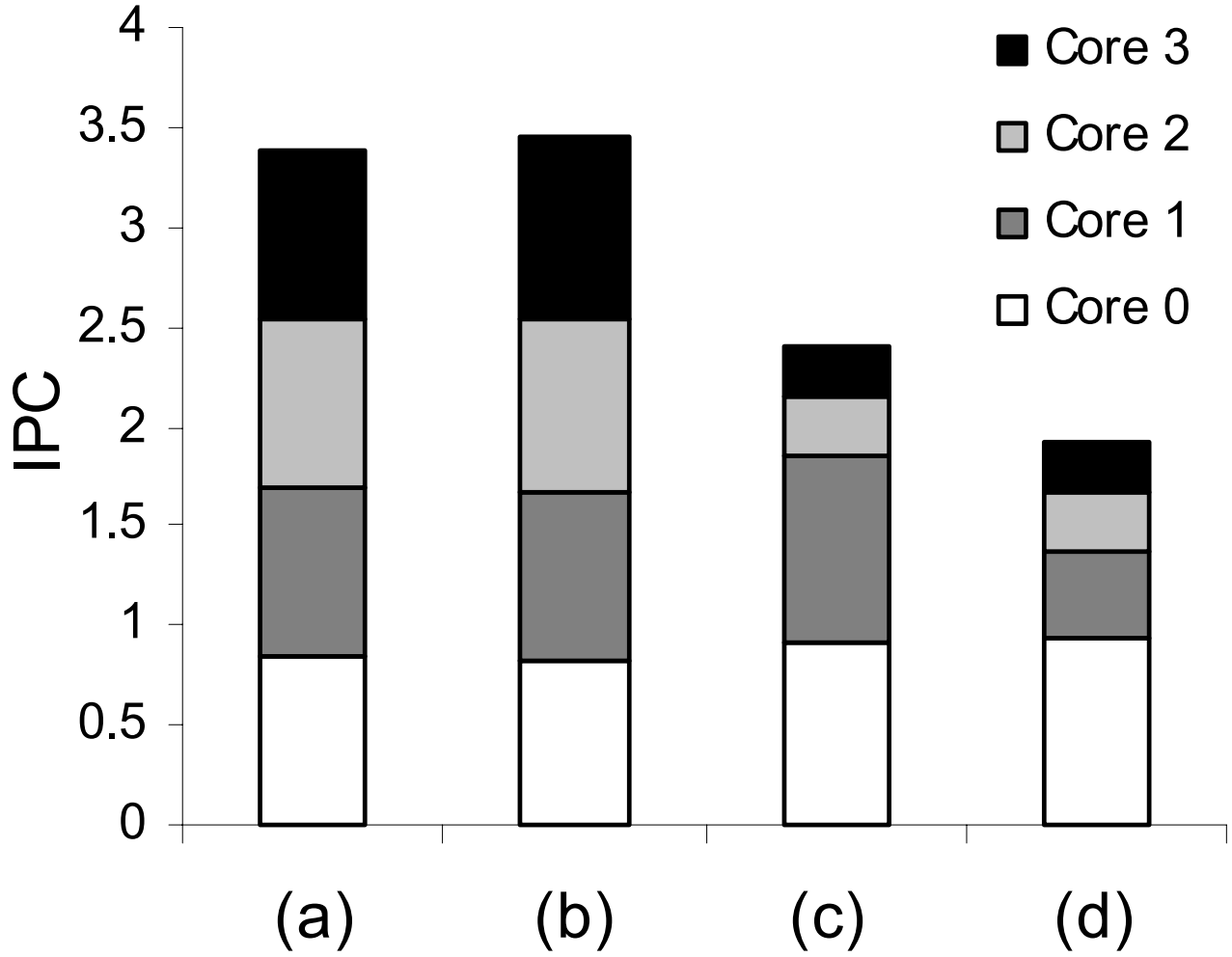
Experimental Methodology

- Construct microbenchmarks with four CPIs: 1, 6, 11, 16
- Four threads with each CPI (16 threads)
- Experiment with different thread assignments on a simulated machine with 4 cores and 4 hw contexts per core

Different Schedules

	Core 0	Core 1	Core 2	Core 3
(a)	1, 6, 11, 16	1, 6, 11, 16	1, 6, 11, 16	1, 6, 11, 16
(b)	1, 6, 6, 6	1, 6, 11, 11	1, 11, 11, 16	1, 16, 16, 16
(c)	1, 1, 6, 6	1, 1, 6, 6	11, 11, 11, 11	16, 16, 16, 16
(d)	1, 1, 1, 1	6, 6, 6, 6	11, 11, 11, 11	16, 16, 16, 16

Performance Results



Caveat

- Our workload had benchmarks with big difference in relative CPIs (1 to 16)

What about real workloads?

SPEC JVM	Average CPI	St. dev.
compress	2.87	0.62
db	3.62	0.61
jack	3.78	0.71
javac	3.58	0.51
jess	3.65	0.49
mpeg	4.51	1.61
mtrt	3.31	0.72

Server benchmarks	Average CPI	St. dev.
SPEC Web Apache	4.33	0.31
SPEC JBB	3.93	0.17

SPEC CPU	Average CPI	St. dev.
bzip2	2.50	0.80
crafty	2.98	0.25
gap	3.70	1.48
gcc	3.23	0.53
gzip	2.37	0.37
mcf	5.11	4.15
parser	3.55	0.30
perl	3.70	0.63
vortex	3.35	0.71
vpr	4.22	0.50
wolf	3.93	0.21

Conclusion

- Benefits from CPI-based scheduling for real workloads are modest

Current Project Status

- Continue work on resource-conscious scheduling
- L2 cache is a critical shared resource
- Developed and evaluated a scheduling technique that reduced L2 miss ratio by 19-37% and improved processor IPC by 27-45%.

More information

http://www.eecs.harvard.edu/~fedorova