Predicting Program Phases and Defending against Side-Channel Attacks using Hardware Performance Counters

Junaid Nomani and Jakub Szefer

Computer Architecture and Security Laboratory
Yale University

junaid.nomani@yale.edu
jakub.szefer@yale.edu

4th Workshop on Hardware and Architectural Support for Security and Privacy
14th June 2014

Outline

- Motivation & Approach
- Side-Channel Example
- Program Behavior and Phases
- Hardware Performance Counters
  - Estimating Interference
- New Scheduler Architecture
  - Machine Learning Module
- Overhead
- Results

Motivation

- Side channel attacks require interference between programs

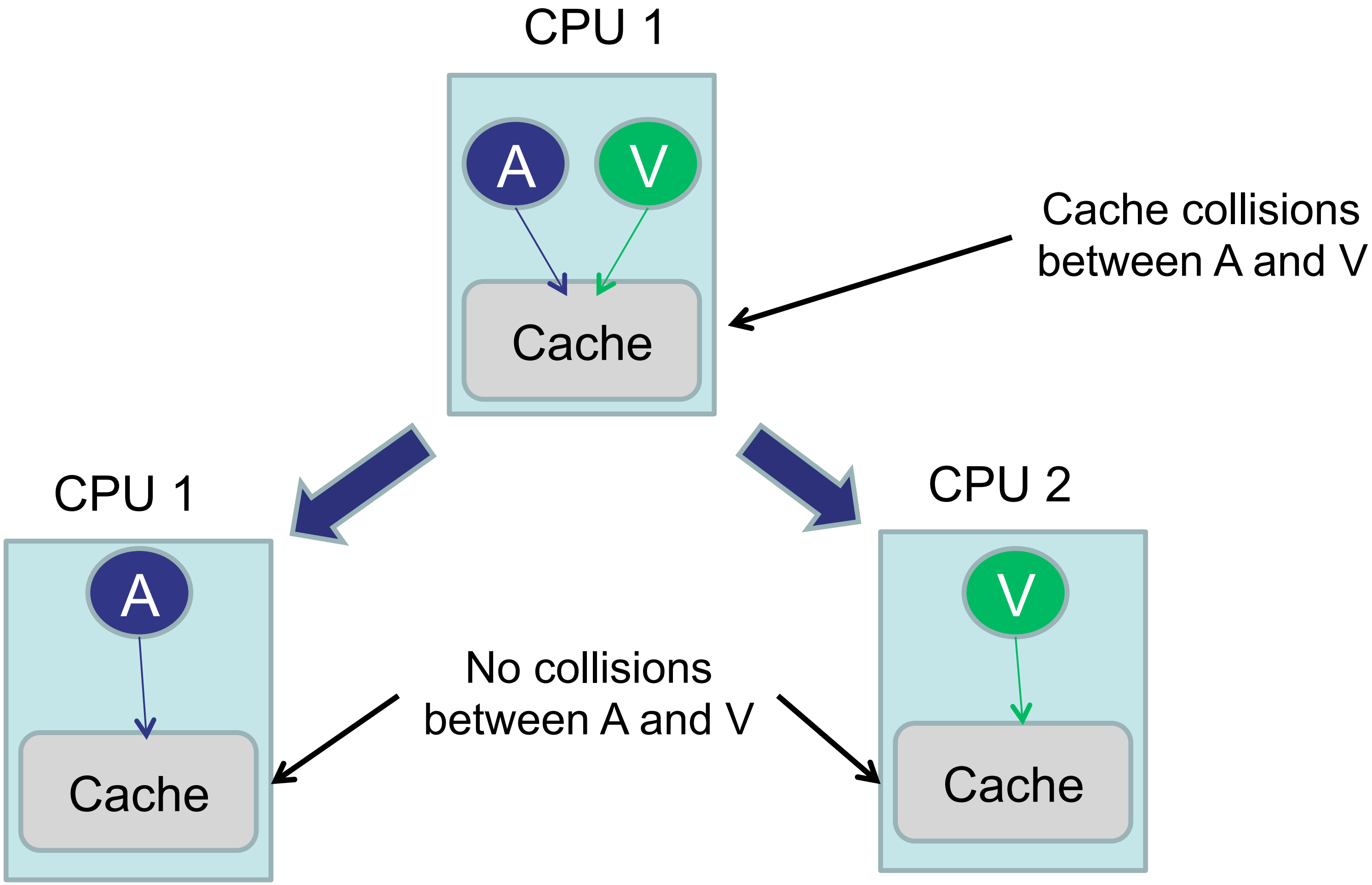
Two programs must share hardware functional units to interfere with another

- Program behavior, or program phases, correlate with hardware functional units used
- Can reduce interference by scheduling interfering programs away from each other so they do not share hardware functional units

Cache Side Channel Example

- Sharing of cache by attacker (A) and victim (V) leads to potential side-channel attacks
- Scheduling the attacker and victim, when they are doing memory accesses, on separate cores means they don’t share caches
- Non-sharing of cache mitigates side-channels

Cache Side Channel Example



Cache collisions between A and V

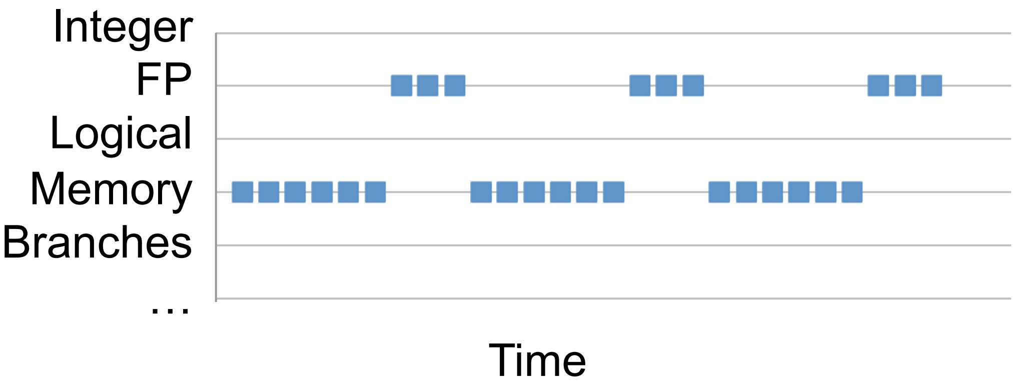
No collisions between A and V

Our Approach

Schedule programs based on predicted program behavior
in order to prevent the interference required
for side channel attacks

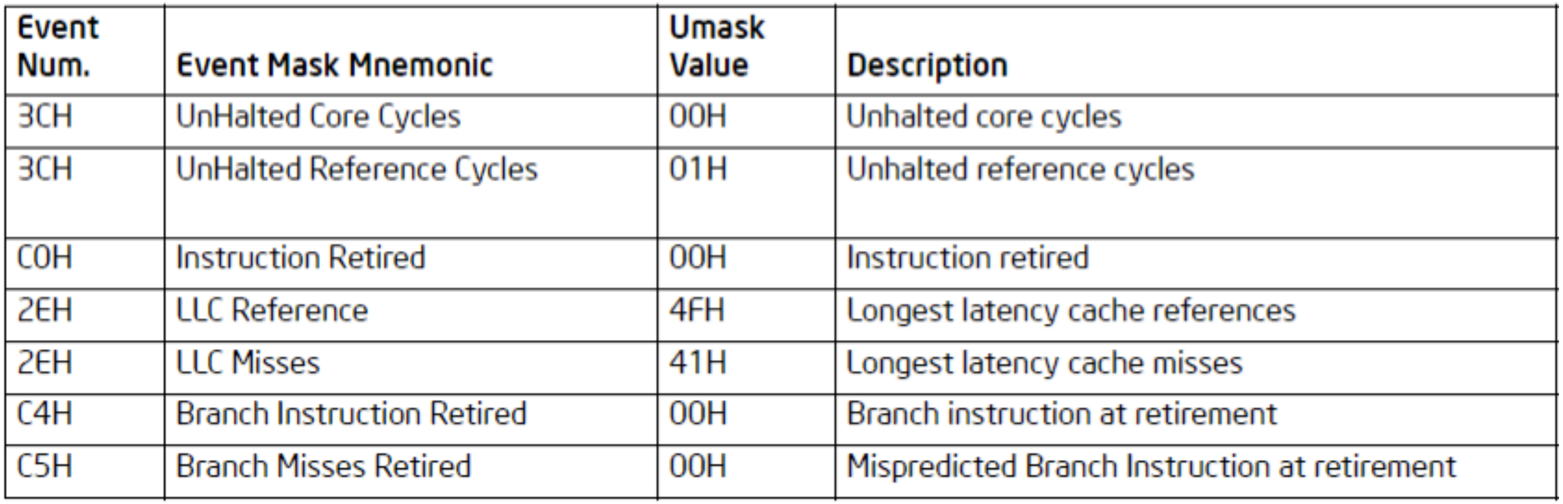
Background: Program Behavior

- Programs tend to exhibit repeating patterns of behavior
  - Program Phases
- Thus by determining past behavior can predict future behavior



Background: Hardware Performance Counters

- Also known as Hardware Performance Monitors
- Can determine current behavior of programs by counting events
- Usually 2 to 4 counters per CPU
- Many events can be counted, e.g. from Intel:



Event Num.	Event Mask Mnemonic	Umask Value	Description
3CH	UnHalted Core Cycles	00H	Unhalted core cycles
3CH	UnHalted Reference Cycles	01H	Unhalted reference cycles
C0H	Instruction Retired	00H	Instruction retired
2EH	LLC Reference	4FH	Longest latency cache references
2EH	LLC Misses	41H	Longest latency cache misses
C4H	Branch Instruction Retired	00H	Branch instruction at retirement
C5H	Branch Misses Retired	00H	Mispredicted Branch Instruction at retirement

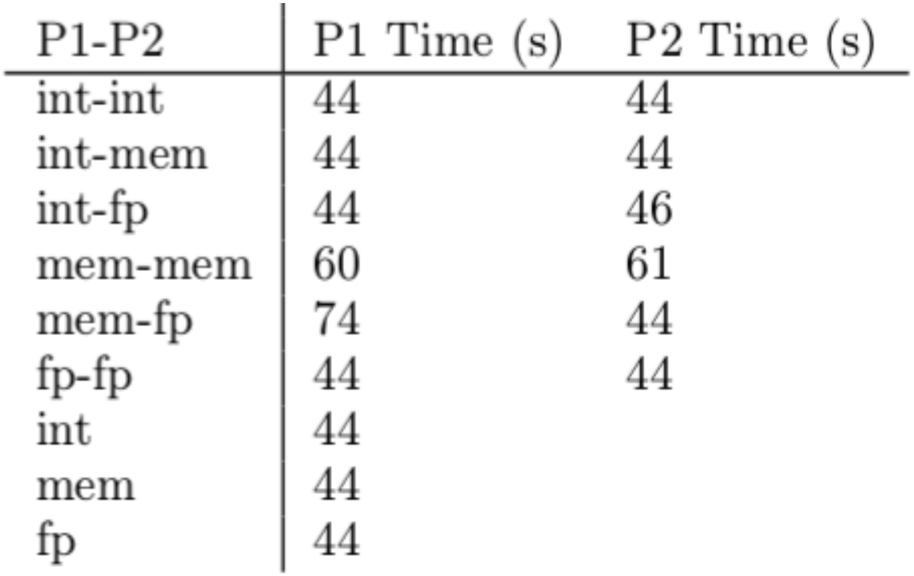
Performance & Interference

- Observed performance changes as programs interfere

- Scheduling of programs affects interference, e.g. mem-mem vs. mem only

- Preliminary tests to correlate performance counter data with interference

Benchmark Performance
(Higher time means more interference)

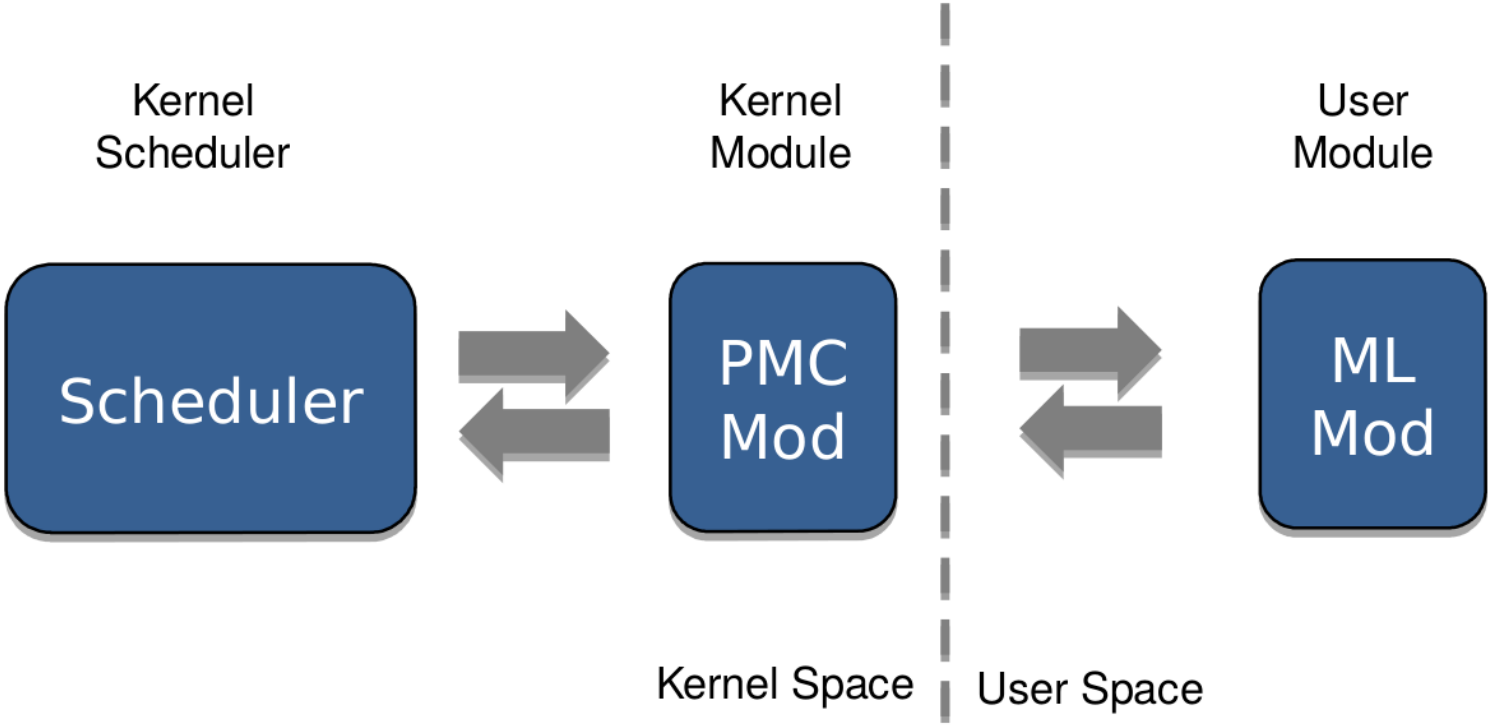


P1-P2	P1 Time (s)	P2 Time (s)
int-int	44	44
int-mem	44	44
int-fp	44	46
mem-mem	60	61
mem-fp	74	44
fp-fp	44	44
int	44	
mem	44	
fp	44	

Outline

- Motivation & Approach
- Side-Channel Example
- Program Behavior and Phases
- Hardware Performance Counters
    - Estimating Interference
- New Scheduler Architecture
    - Machine Learning Module
- Overhead
- Results

New Scheduler Architecture



New Scheduler Architecture

- Modified scheduler
  - Collect performance counter data
  - Uses prediction of upcoming program phase to separate memory programs
  - Attempt to minimized side-channels

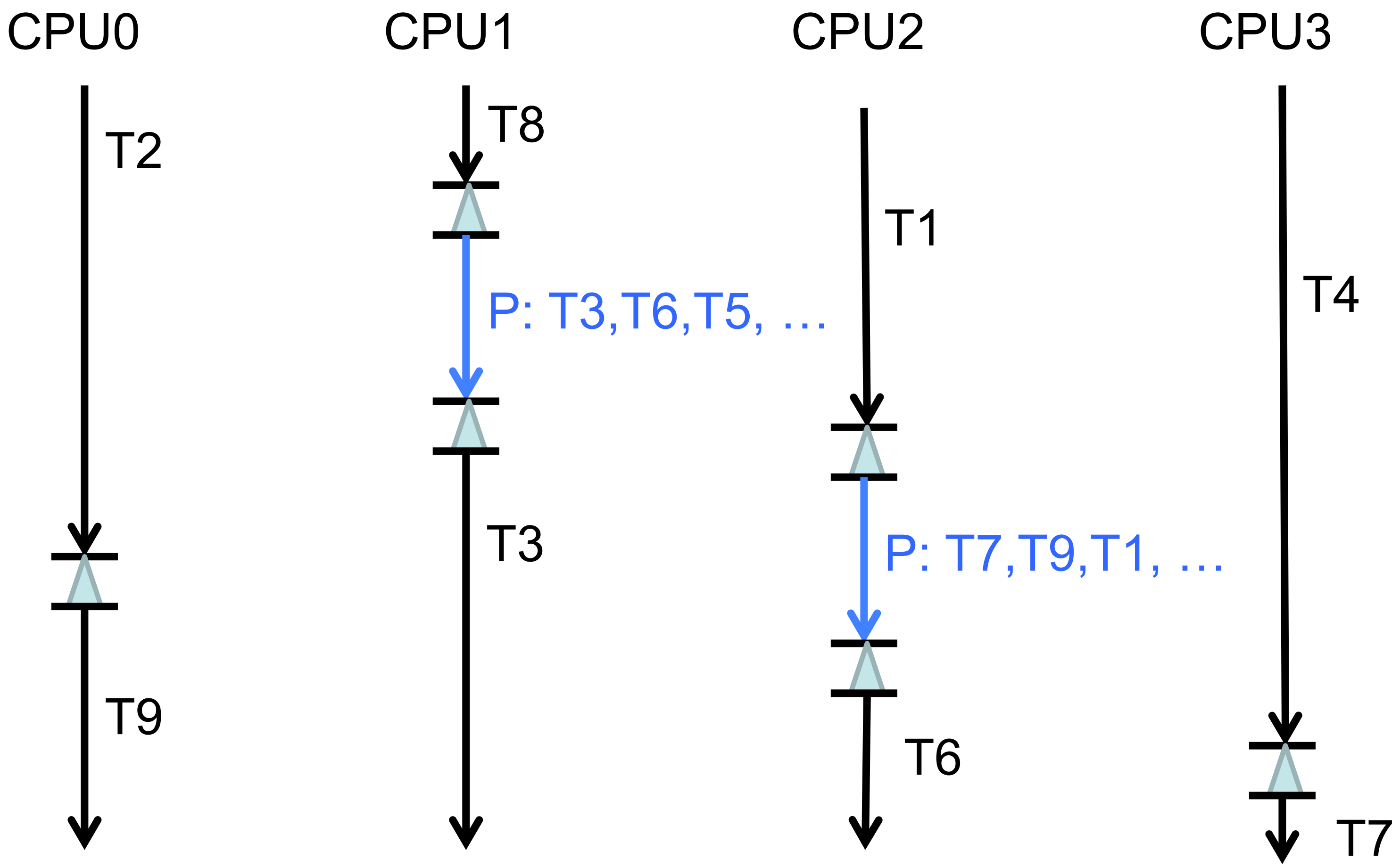
- PMC Module
  - Interface between kernel data structures and ML Module

- ML Module
  - Machine learning module responsible for predicting upcoming program phase for each program

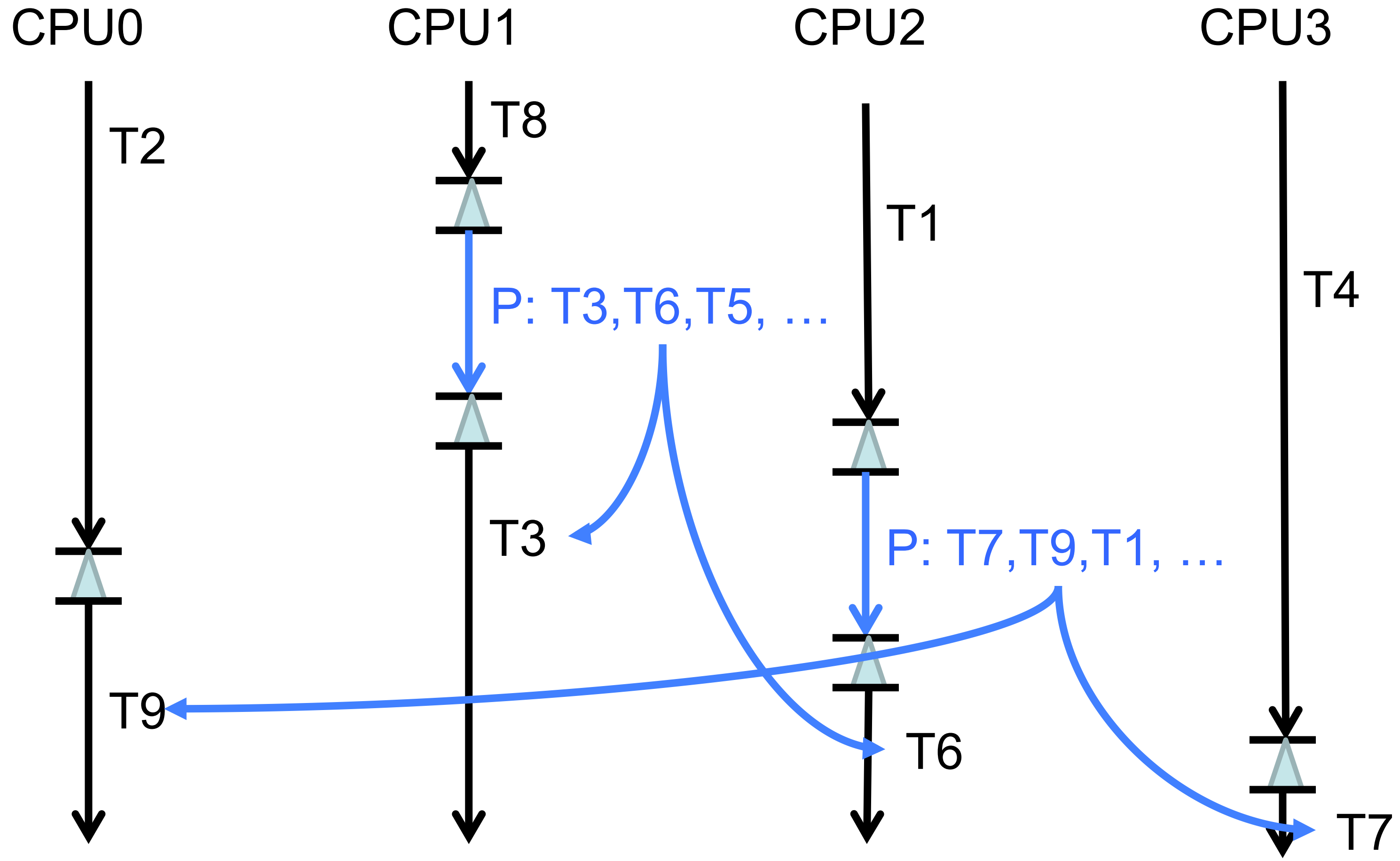
ML Module

- Predict upcoming program phase using ML module
- Uses neural network
    - 7 Layers (5 Hidden)
    - Input layer receives counter data from last 15 context switches
    - Counter data and output clustered using K-Means into 5 categories
    - Outputs which category the next context switch will be in

Asynchronous ML Module Execution



Asynchronous ML Module Execution



Evaluation – Scheduler Overhead

- Counter recording only takes 50 instructions per context switch. Negligible

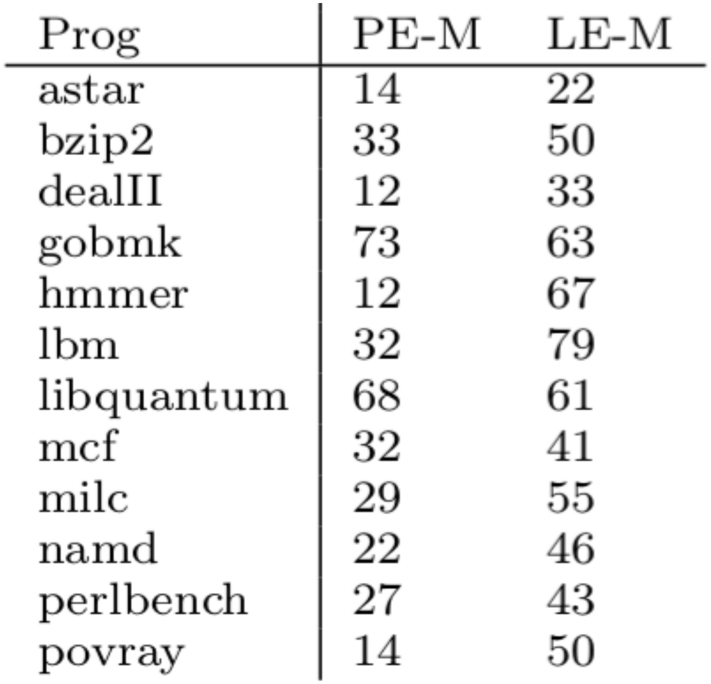
- ML module prediction takes about 210us with about 10us of communication overhead. Context switches occur every ~2500us. Have enough time to predict future behavior of ~10 threads.

- ML module training done off-line. Similar to updating a user application when a new version is released.

Evaluation – Prediction Error Rates

- PE-M: Our predictor leverages the machine learning algorithms
- LE-M: Base predictor using last phase to predict next phase
- Memory phase prediction error rates: PE-M ~30% avg vs. LE-M: ~50% avg

Prediction Error Rates
(Less is better)



Prog	PE-M	LE-M
astar	14	22
bzip2	33	50
dealII	12	33
gobmk	73	63
hmmer	12	67
lbm	32	79
libquantum	68	61
mcf	32	41
milc	29	55
namd	22	46
perlbench	27	43
povray	14	50

Summary and Ongoing Work

- Can use prediction of program behavior to schedule tasks on different cores to eliminate interference and minimize side channels

- Ongoing Work: Develop scheduler to utilize this prediction directly into the Linux scheduler with minimal overhead:

SOFT: Soft, low-Overhead, Fair Transfer scheduler

Thank you!

Questions?

Junaid Nomani and Jakub Szefer

Computer Architecture and Security Laboratory
Yale University

junaid.nomani@yale.edu
jakub.szefer@yale.edu