RPPM: Rapid Performance Prediction of Multithreaded Workloads on Multicore Processors

Sander De Pestel*, Sam Van den Steen*,
Shoaib Akram, Lieven Eeckhout

*Intel, Belgium Ghent University

ISPASS — March 25-26, 2019
Madison—Wisconsin

Analytical Modeling

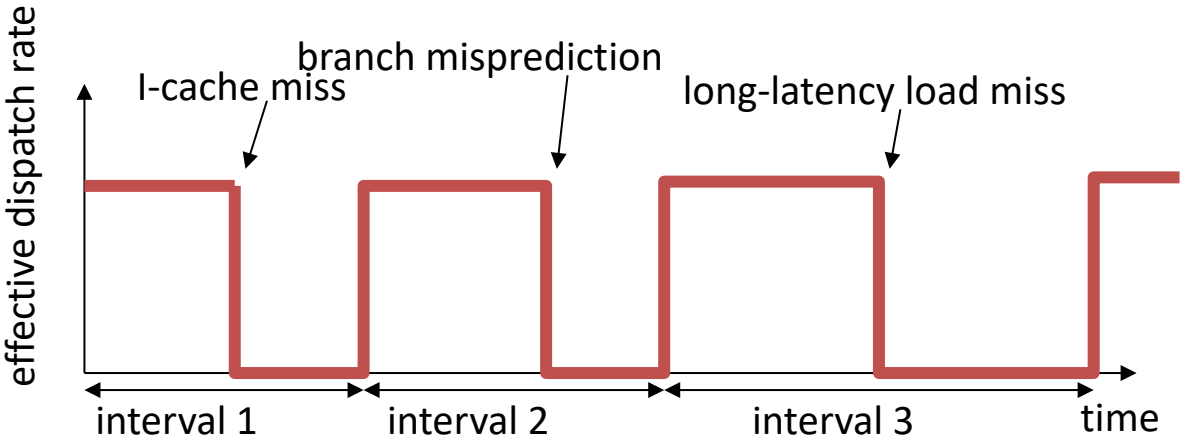
Key features

- Super fast
- Useful complement to simulation
- Quickly explore large design spaces in early design stages

Three types:

- Empirical modeling
  - Black-box model, Easy to build, Needs training examples
- **Mechanistic modeling → this paper**
  - White-box model, Insight
- Hybrid modeling
  - Parameter fitting of semi-mechanistic model, Needs training

Prior Work in Mechanistic Modeling



Interval analysis for OOO cores

Michaud [PACT'99], Karkhanis [ISCA'04], Eyerman [TOCS'09]

Microarchitecture-independent model

Van den Steen [ISPASS'15]

Limited to single-core processors

Prior Work in Multicore Models

Amdahl's Law: high abstraction

- Hill/Marty [Computer'08]

Hybrid models:

- Popov [IPDPS'15]: Amdahl's Law + simulation

Multi-programmed workloads: no inter-thread communication nor synchronization

- Jongerius [TC'18]

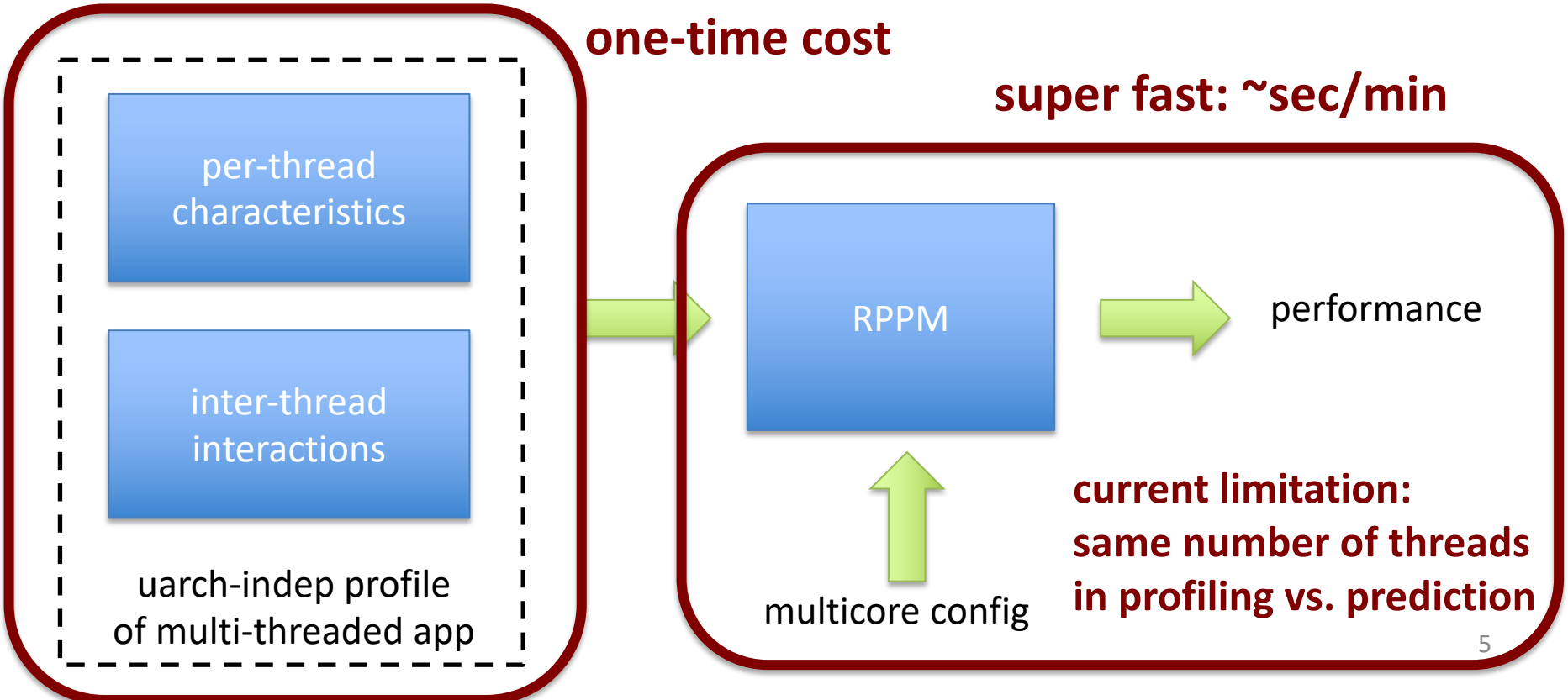
Machine learning: empirical, black-box model

- Ipek [ASPLOS'06], Lee [MICRO'08]

This work: multicore, multithreaded, mechanistic (white-box), microarchitecture-independent profile

Paper Contribution

Microarchitecture-independent mechanistic performance model for multithreaded workloads on multicore processors

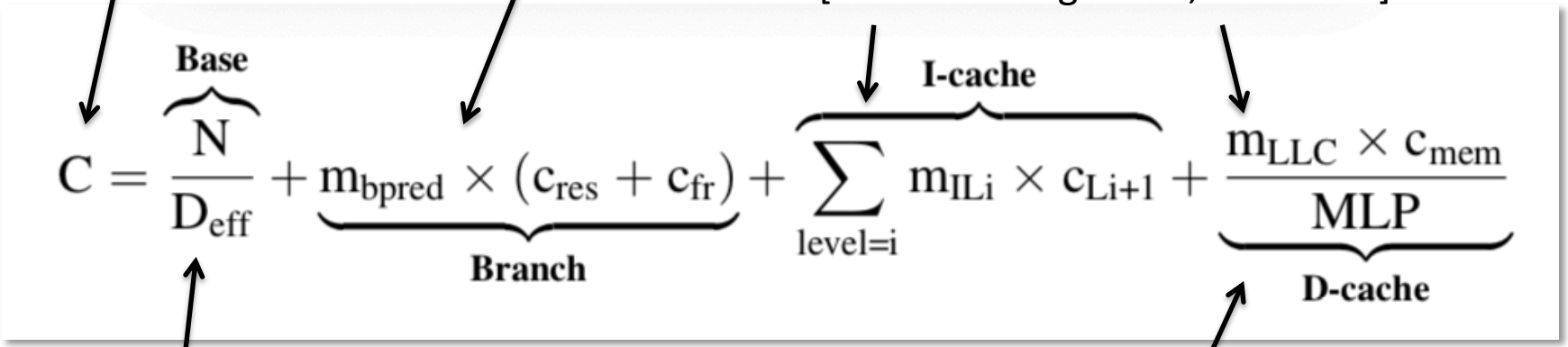


Single-Threaded Interval Model

uarch-indep branch predictor model [De Pestel, ISPASS'15]

total cycle count

miss rates predicted using StatStack [Eklov and Hagersten, ISPASS'10]



$$C = \overbrace{\frac{N}{D_{eff}}}^{\text{Base}} + \underbrace{m_{bpred} \times (c_{res} + c_{fr})}_{\text{Branch}} + \overbrace{\sum_{level=i} m_{ILi} \times c_{Li+1}}^{\text{I-cache}} + \underbrace{\frac{m_{LLC} \times c_{mem}}{MLP}}_{\text{D-cache}}$$

uarch-indep MLP model [Van den Steen, CAL'18]

N = dynamic instruction count

D_{eff} = effective dispatch rate; is function of ILP, I-mix, ALU contention

[Van den Steen et al., ISPASS'15]

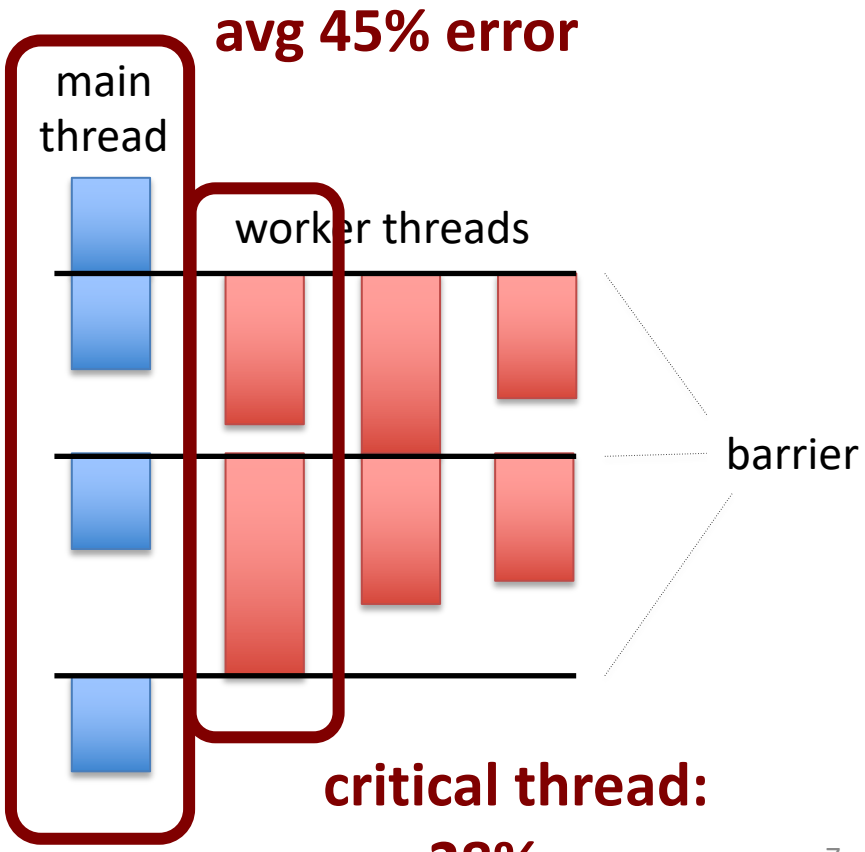
Naïve Extensions

Apply single-threaded model [Van den Steen, ISPASS'15] to

- Main thread
- Critical thread

Fails to model

- Synchronization
- Coherence effects
- Resource contention



Modeling Multithreaded Performance is Fundamentally Difficult

Need **super accurate** per-thread performance prediction

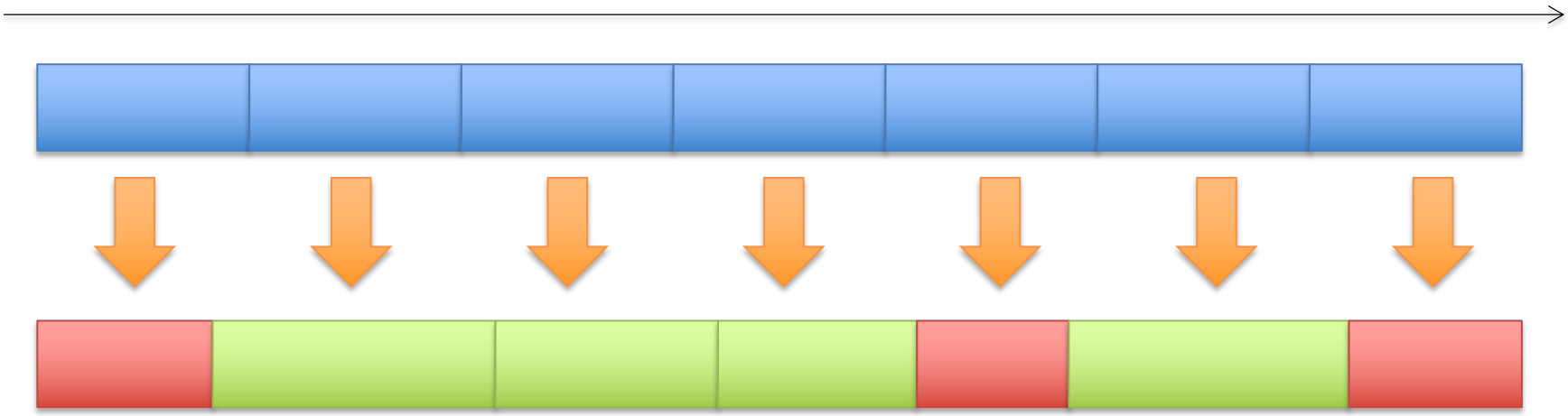
- Accumulating errors because of synchronization

Need to accurately model

- Inter-thread synchronization
  - Barriers, critical sections, producer/consumer, etc.
- Inter-thread communication
  - Cache coherence
- Inter-thread interference
  - Shared resources (e.g., LLC)

Accumulating Random Errors

Predicting single-thread performance

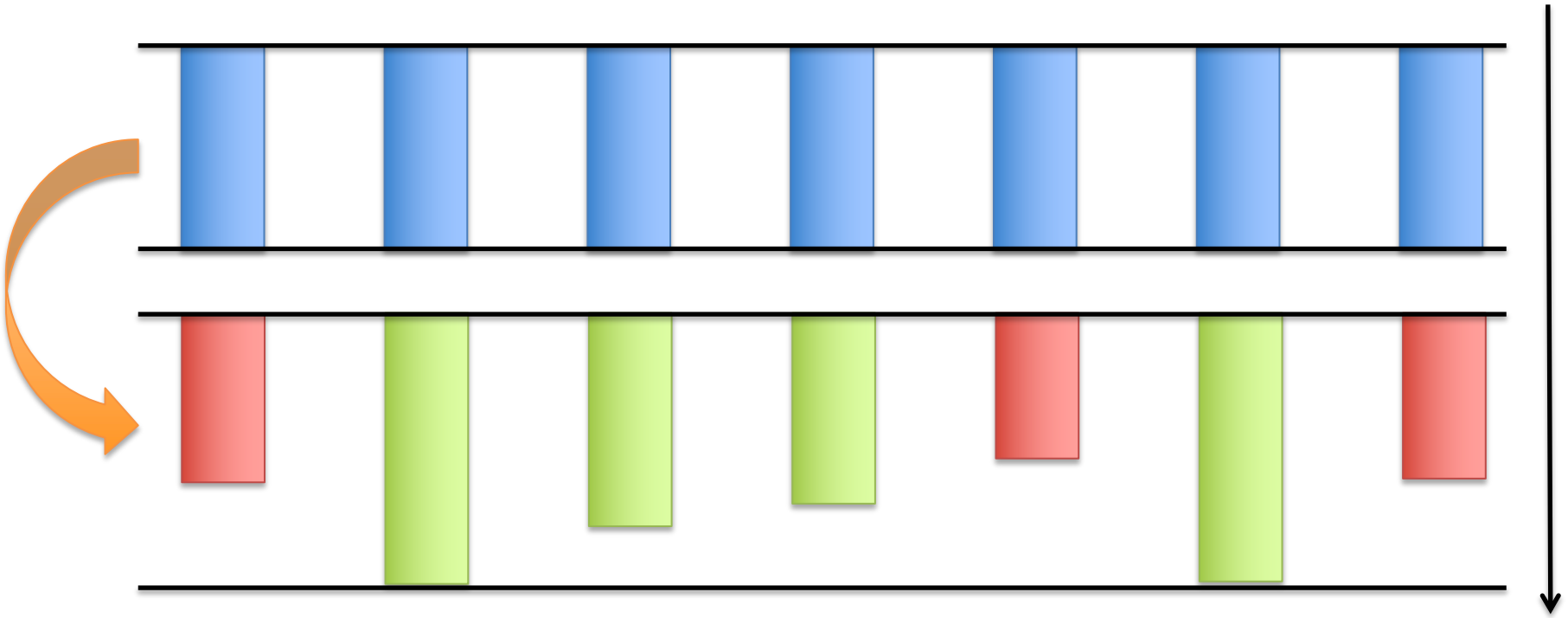


Random errors across short intervals cancel out

Systematic errors (obviously) don't

Accumulating Errors (cont'd)

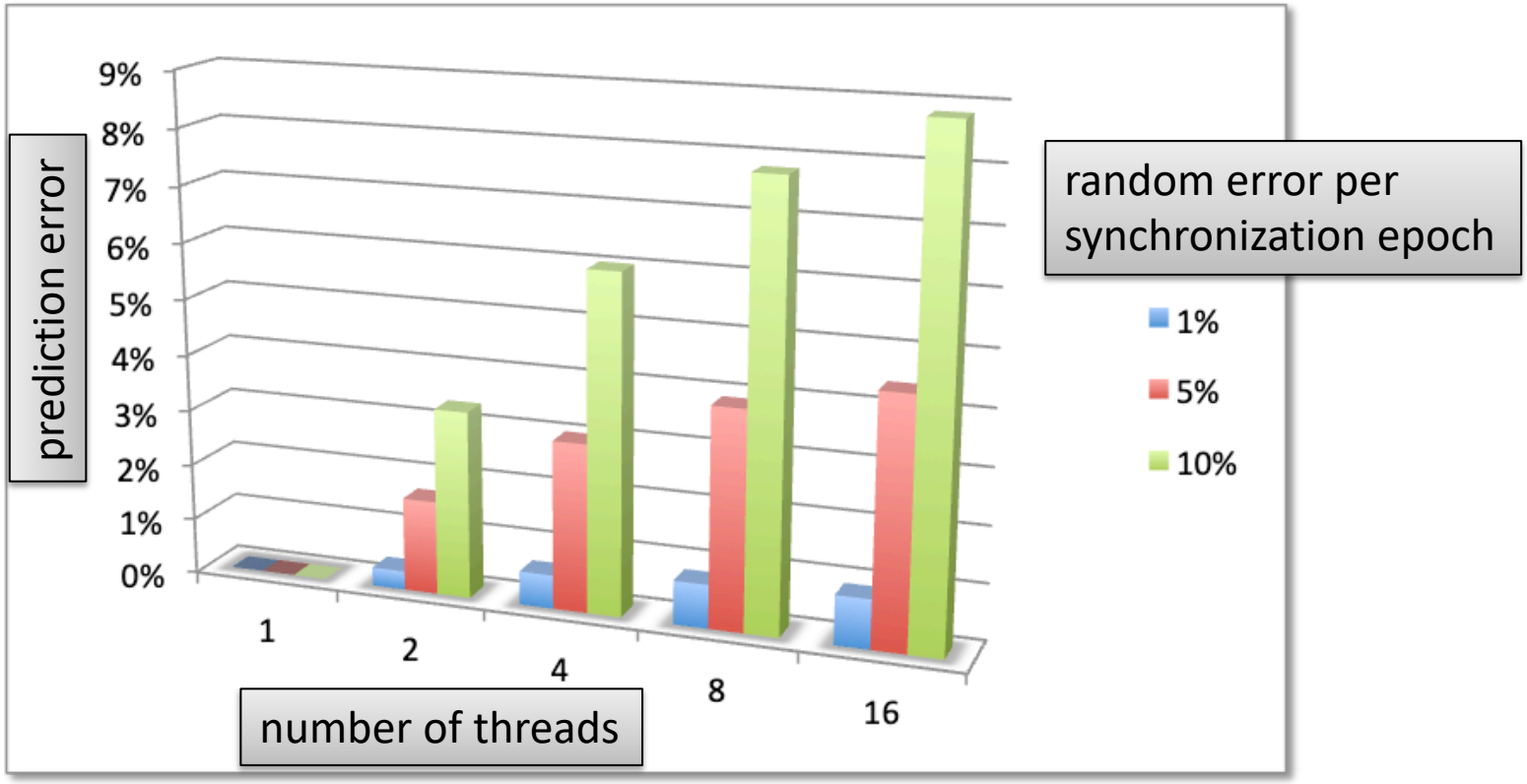
Predicting multithreaded performance b/w barriers



Random errors do not cancel out

Problem Exacerbates with Thread Count

Synthetic barrier-synchronized loop w/ 1M iterations and fixed work per iteration



RPPM Model

Profiling

Per-thread characteristics

Van den Steen [ISPASS'15]

Synchronization

Shared memory accesses

Pin-based; measured per synchronization epoch

Prediction

Predict per-thread performance per synchronization epoch

Predict impact of synchronization

Profiling Synchronization

Intercept library function calls in Pin

- `pthread` and OpenMP
- Automatic

For example

- Critical sections (pthread)

  ```
  pthread_mutex_lock
  pthread_mutex_unlock
  ```

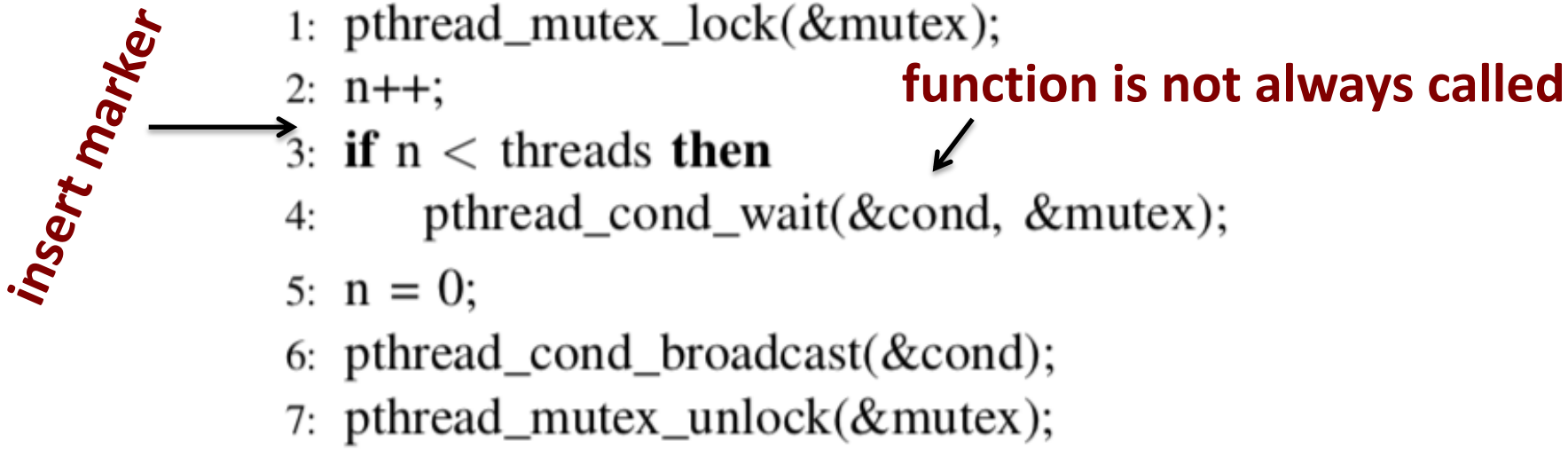
- Barriers (OpenMP)

  ```
  gomp_team_barrier_wait (gomp_barrier_t)
  ```

User-level synchronization: annotate manually

Condition Variables

Barrier using condition variables:



Similar solution for producer-consumer, semaphores, etc.

Too cumbersome? No!

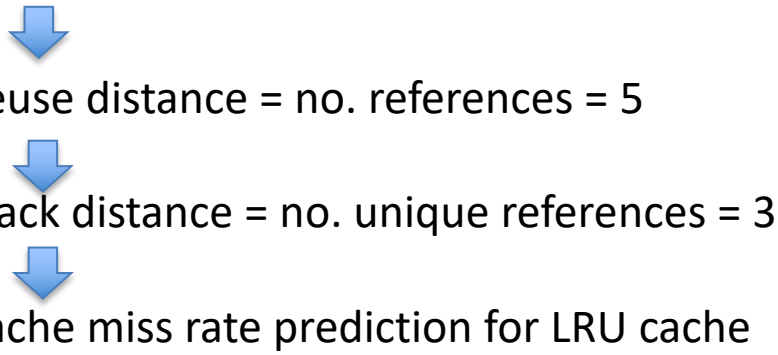
- 4 Parsec benchmarks: `pthread_cond_wait`
- facesim: `pthread_cond_wait` and `pthread_cond_broadcast`

Shared Memory Behavior

Cold misses: first reference

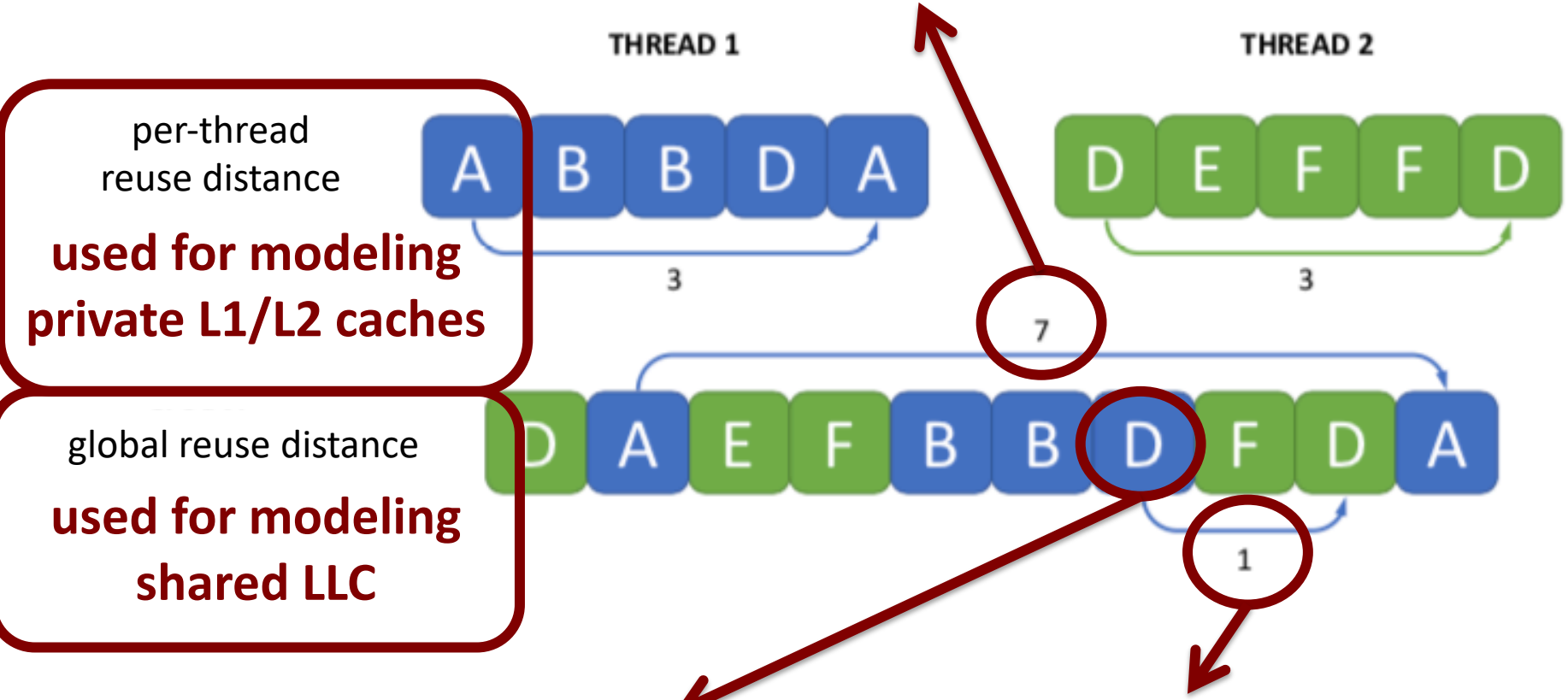
Conflict/capacity misses: StatStack [Eklov ISPASS'10]

ABCDAABDDCAABEFCAB



Shared Memory Behavior cont'd

larger reuse distance →
possibly negative interference



per-thread
reuse distance

used for modeling private L1/L2 caches

global reuse distance

used for modeling shared LLC

if a write → write invalidation for D (infinite reuse distance)

shorter reuse distance → positive interference

Prediction

Per-epoch active execution time

Single-threaded model

$$C = \overbrace{\frac{N}{D_{eff}}}^{\text{Base}} + \underbrace{m_{bpred} \times (c_{res} + c_{fr})}_{\text{Branch}} + \overbrace{\sum_{level=i} m_{ILi} \times c_{Li+1}}^{\text{I-cache}} + \underbrace{\frac{m_{LLC} \times c_{mem}}{MLP}}_{\text{D-cache}}$$

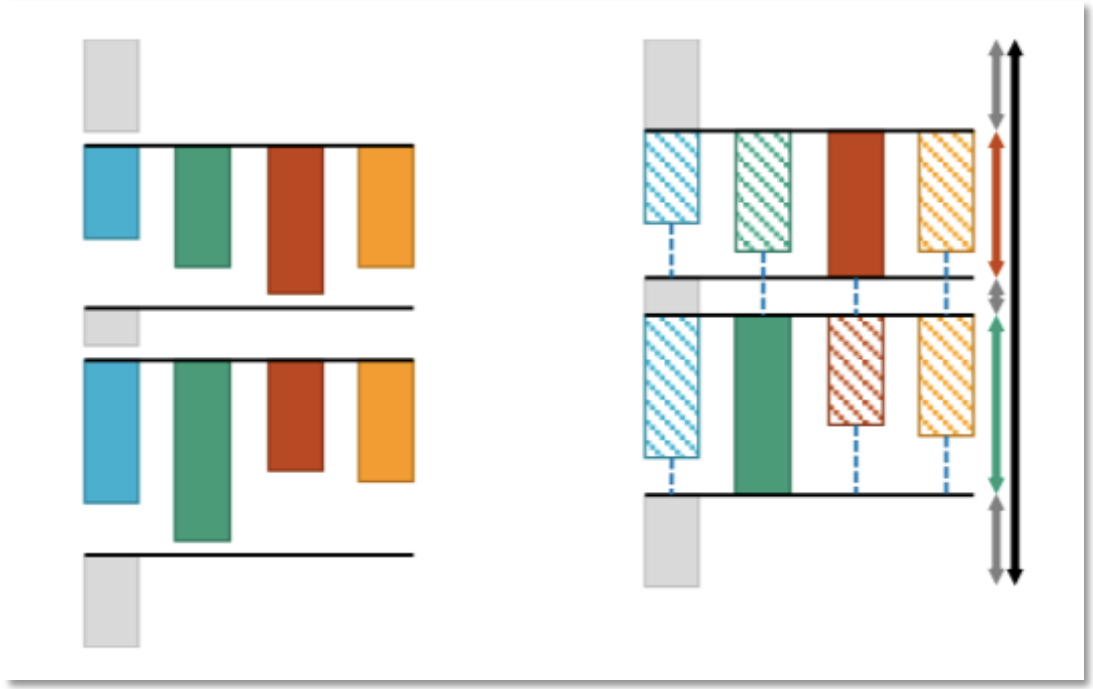
- To predict active execution time per synchronization epoch
- Miss rates account for interference and coherence

Prediction

Synchronization overhead

Symbolic execution: fastest to slowest thread

- Fastest thread(s)
  - experience(s)
  - idle time
- Slowest thread
  - determines
  - execution time



Critical sections, barriers, condition variables, thread create/join, etc.

Experimental Evaluation

Rodinia (OpenMP)

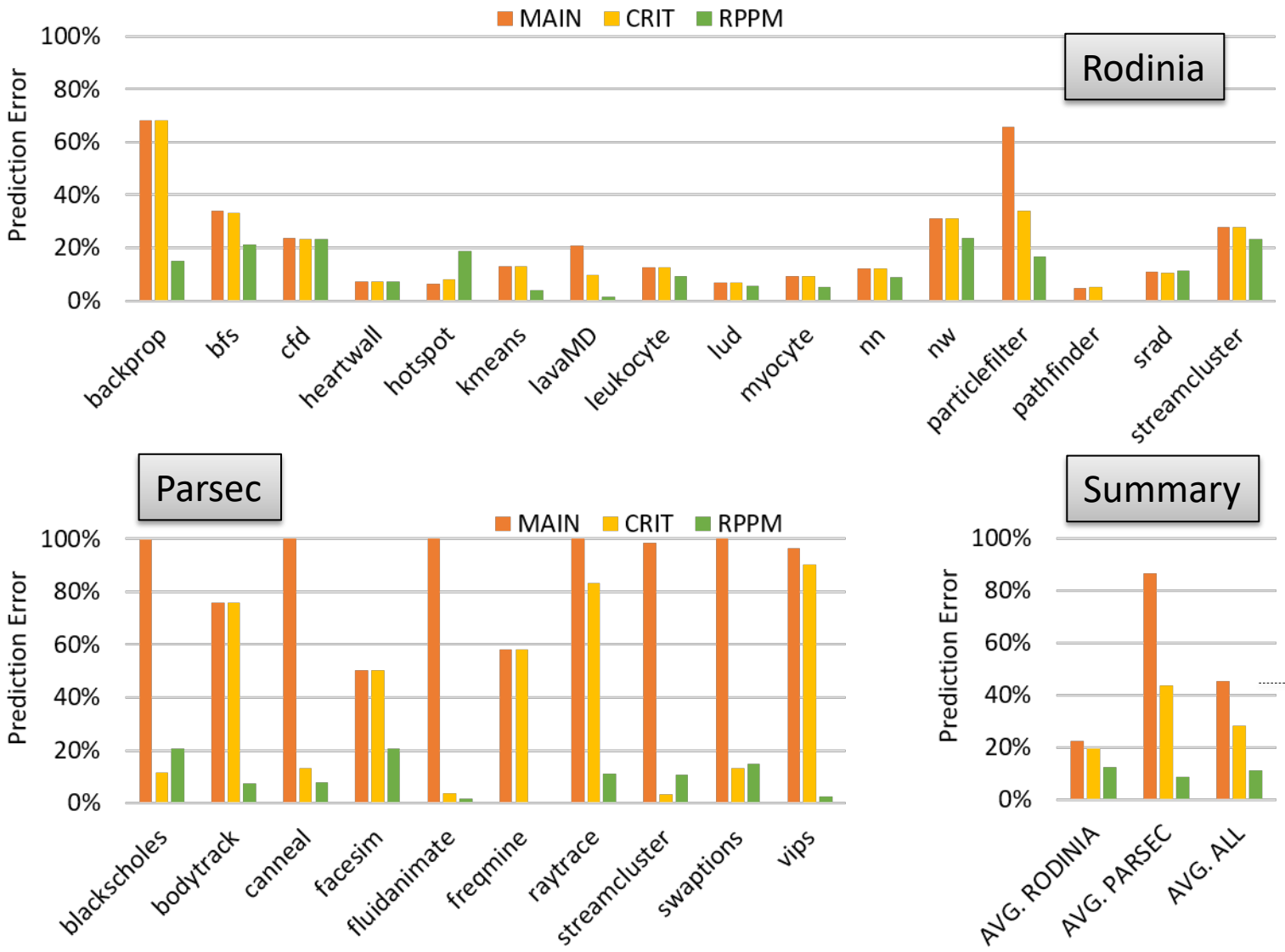
- Barrier synchronization

Parsec (pthread)

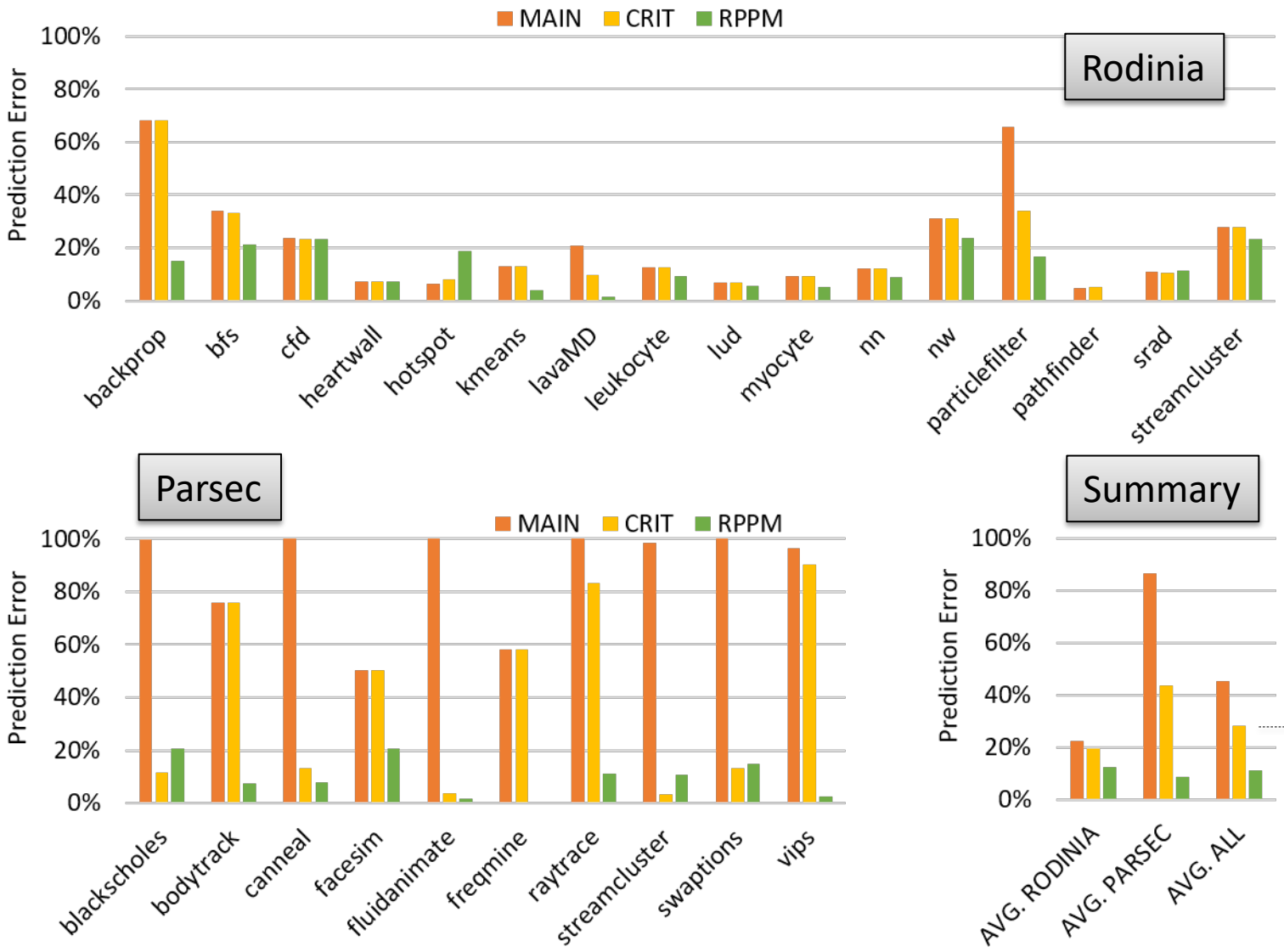
Benchmark	Critical Sections	Barriers	Cond. var.
Blackscholes	–	–	–
Bodytrack	6,700	98	25
Canneal	4	64	–
Facesim	10,472	–	1,232
Fluidanimate	2,140,206	50	–
Freqmine	–	–	–
Raytrace	47	–	15
Streamcluster	68	13,003	34
Swaptions	–	–	–
Vips	8,973	–	1,433

Simulator: HW-validated x86 **Sniper**, quad-core 4-wide OOO [Carlson TACO'14]

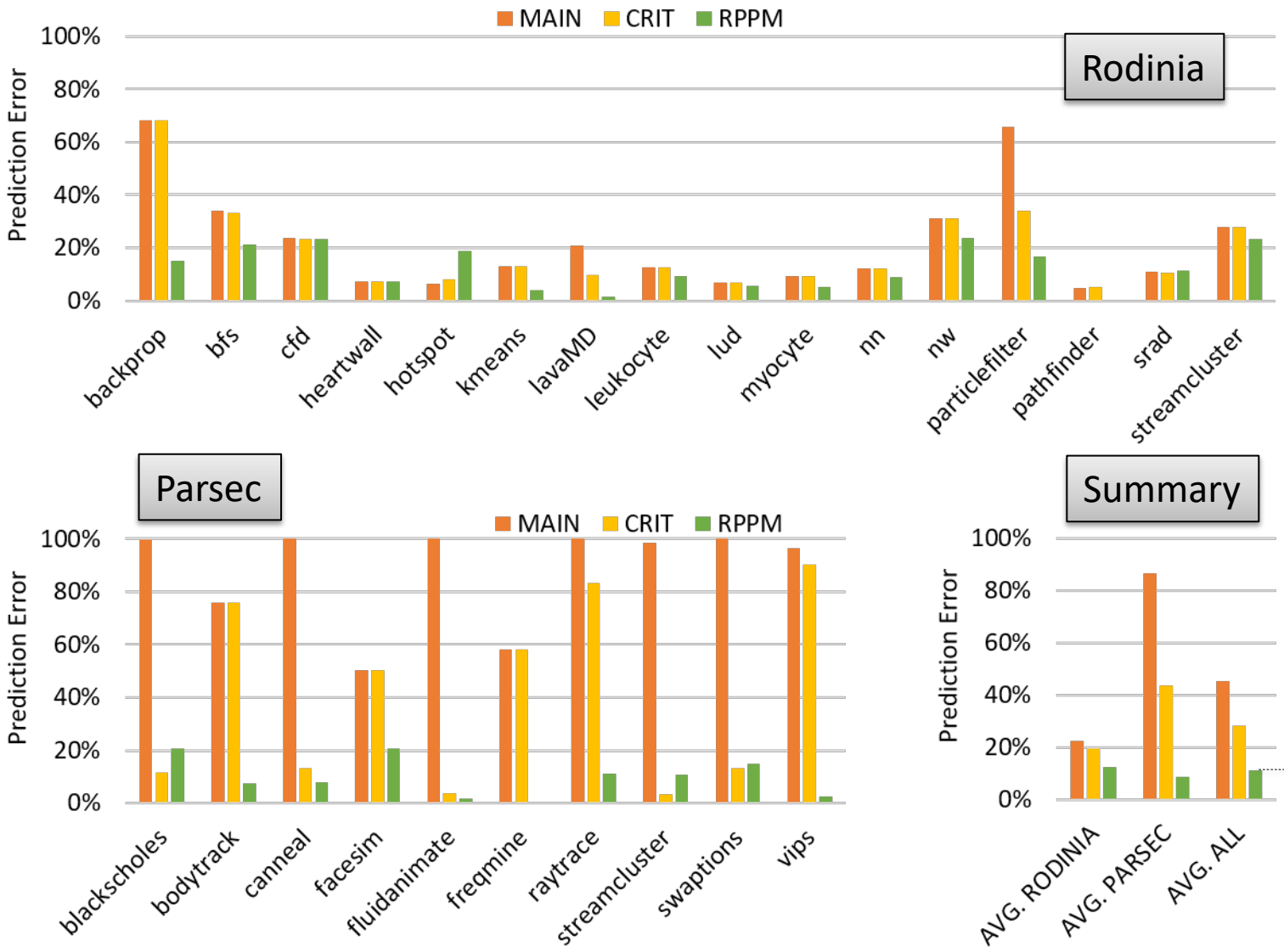
MAIN (models main thread): reasonable accuracy for Rodinia but highly inaccurate for Parsec



CRIT (models critical thread): more accurate for Parsec



RPPM (models critical thread per sync-epoch): 11% avg error versus MAIN (45%) and CRIT (28%)



Design Space Exploration

Which is the best performing 10-GOPS processor?

	smallest	small	base	big	biggest
frequency (GHz)	5.0	3.33	2.5	2.0	1.66
width	2	3	4	5	6

Hybrid exploration strategy:

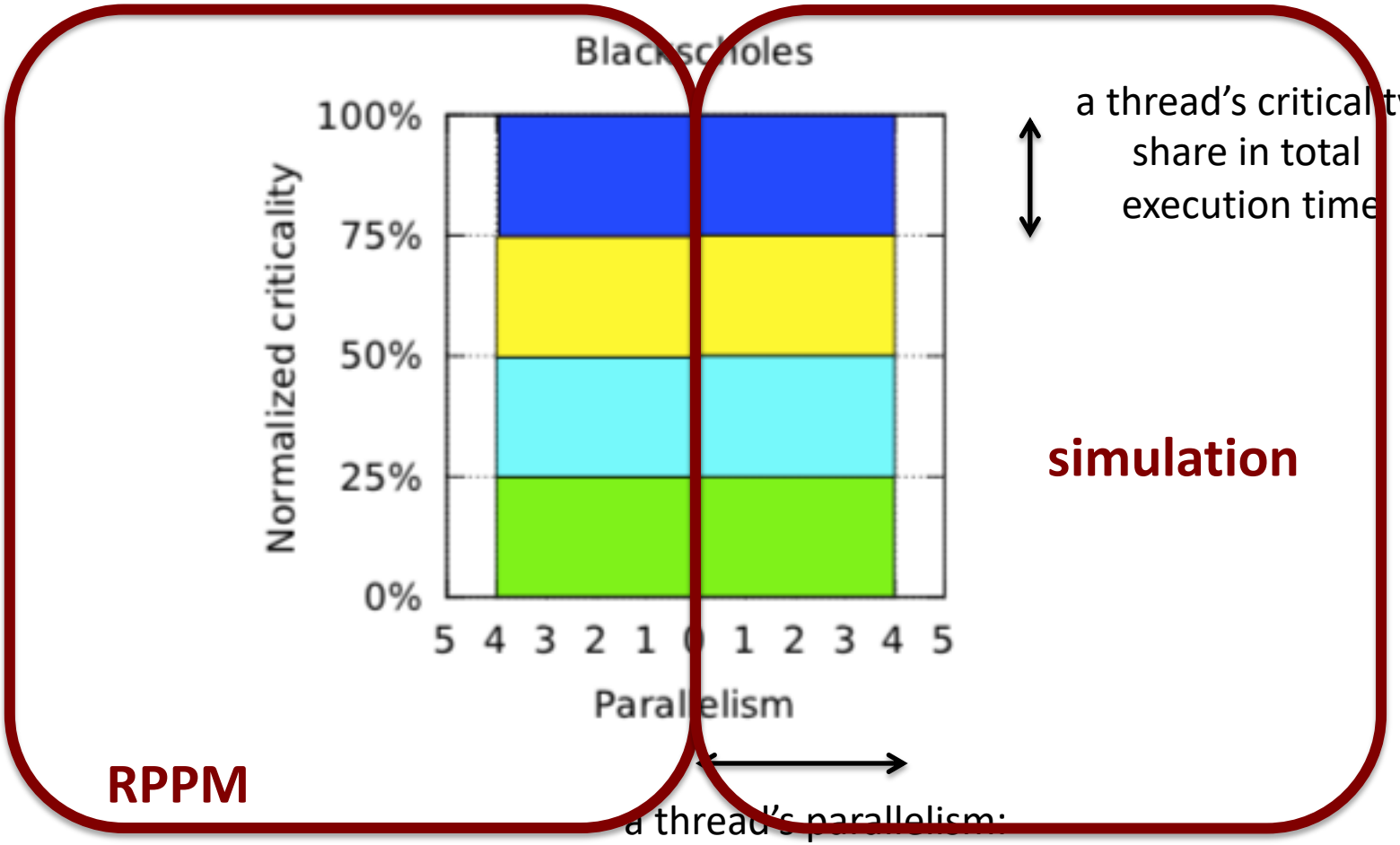
- Use RPPM to predict optimum design
- Simulate designs within 5% of predicted optimum

Identify true optimum for all but one benchmark

- RPPM predicts optimum for vast majority of benchmarks
- Handful benchmarks need two simulation runs
- `pathfinder`: within 2% of true optimum

Bottlegraphs:

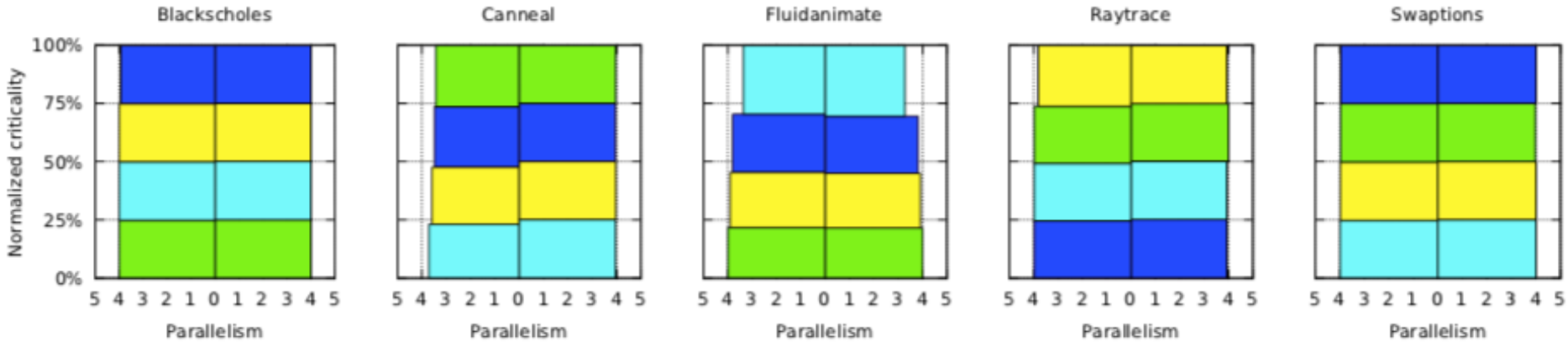
Visualizing a thread's criticality and parallelism



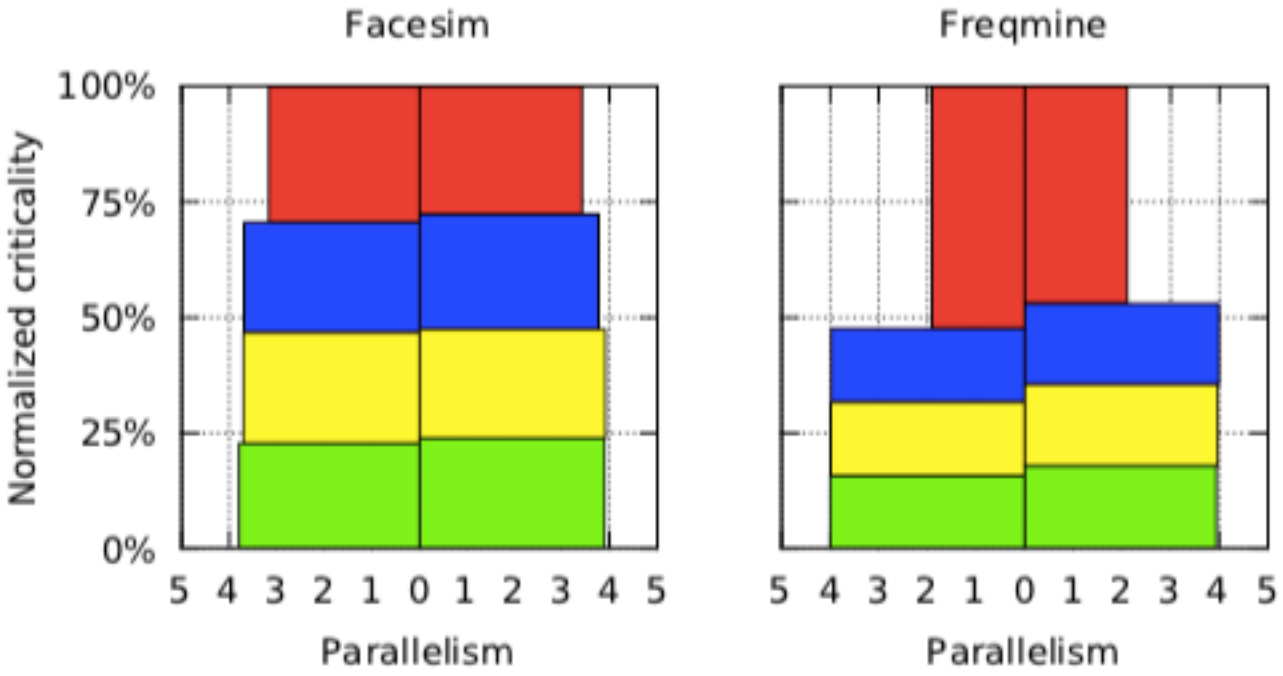
[Du Bois, OOPSLA'13]

Bottlegraphs:
Balanced workloads

Main thread distributes and co-works with worker threads



Bottlegraphs:
Imbalanced workloads

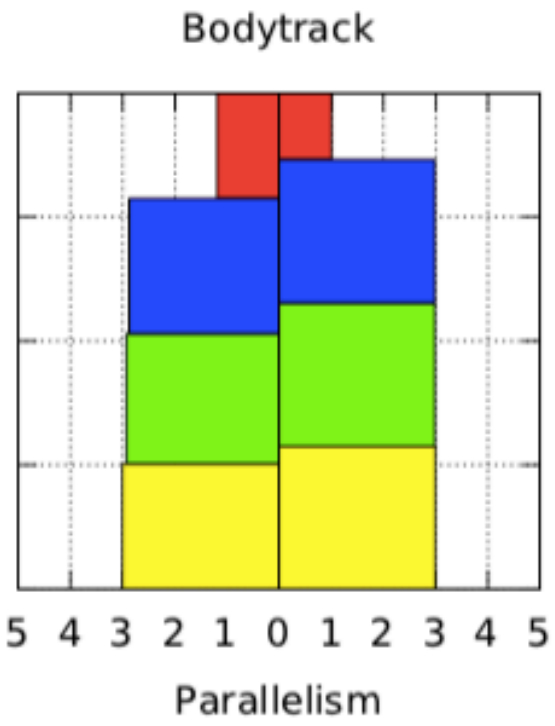


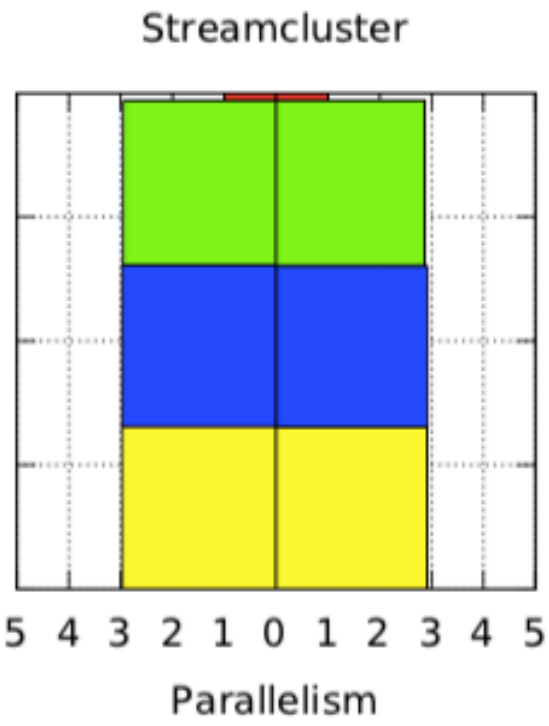
`facesim`: main thread performs slightly more work

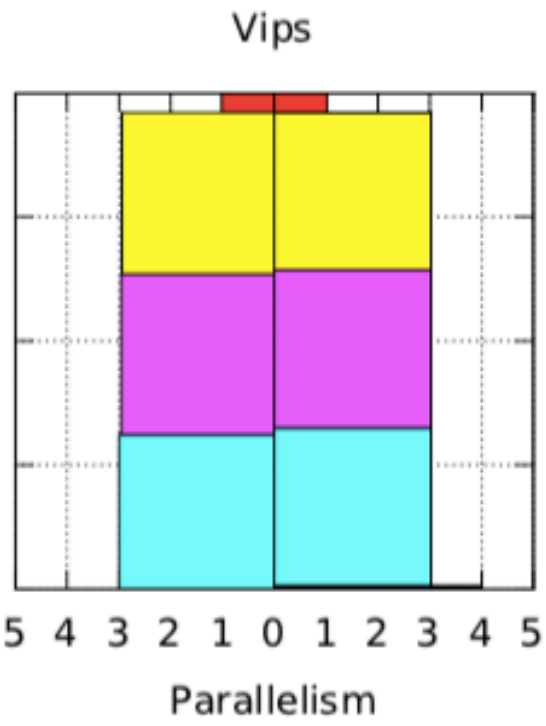
`freqmine`: main thread is bottleneck (but does parallel work)

Bottlegraphs:

Highly imbalanced workloads







Main thread does not perform any parallel work

Conclusions

Microarchitecture-independent mechanistic performance model for multithreaded workloads on multicore processors

- Accumulating random errors
- Inter-thread synchronization, communication, interference

Evaluation against simulation: 11% avg error versus MAIN (45%) and CRIT (28%)

Use cases

- Design space exploration
- Workload characterization

Future work: predict across thread counts

- Predict Y-thread performance from X-thread profile (Y>X)
- Predict Y-thread performance on X-core system (Y>X)

RPPM: Rapid Performance Prediction of Multithreaded Workloads on Multicore Processors

Sander De Pestel*, Sam Van den Steen*,
Shoaib Akram, Lieven Eeckhout

*Intel, Belgium Ghent University

ISPASS — March 25-26, 2019
Madison—Wisconsin