LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Computer Architecture, June 19th 2022, USA

Agenda

Time	Speaker	Topic
13.20 to 13.30	Alen Sabu	Overview of the tutorial
13.30 to 14.30	Harish Patil	Tools & Methodologies: Pin, PinPlay, SDE, ELFies
14.30 to 15.00		Break
15.00 to 15.50	Wim Heirman	Simulation with Sniper / Sniper 8.0 GitHub release
15.50 to 16.45	Alen Sabu	Single-threaded and Multi-threaded Sampling, LoopPoint
16.45 to 17.30	Alen Sabu	Running LoopPoint Tools

Agenda

Time	Speaker	Topic
13.20 to 13.30	Alen Sabu	Overview of the tutorial
13.30 to 14.30	Harish Patil	Tools & Methodologies: Pin, PinPlay, SDE, ELFies
14.30 to 15.00	Break	
15.00 to 15.50	Wim Heirman	Simulation with Sniper / Sniper 8.0 GitHub release
15.50 to 16.45	Alen Sabu	Single-threaded and Multi-threaded Sampling, LoopPoint
16.45 to 17.30	Alen Sabu	Running LoopPoint Tools

Tools from Intel

- Speaker: Harish Patil
  - Principal Engineer, Intel Corporation
- Topics Covered
  - Binary instrumentation using Pin or writing Pintools
  - PinPlay kit and PinPlay-enabled tools
  - SDE build kit for microarchitecture emulation
  - Checkpointing threaded applications using PinPlay, SDE
  - Detailed discussion on ELFies including its generation and usage

NUS National University of Singapore intel

Simulation and Sampling Overview

- Speaker: Wim Heirman
  - Principal Engineer, Intel Corporation
- Topics Covered
  - Architectural exploration and evaluation
  - Simulation as a tool for performance estimation
  - Methods for fast estimation using simulation
  - Overview of Sniper simulator
  - Sniper 8.0 features and public release

LoopPoint Methodology

- Speaker: Alen Sabu
  - PhD Candidate, National University of Singapore
- Topics Covered
  - Single-threaded sampled simulation techniques
  - Sampled simulation of multi-threaded applications
  - Existing methodologies and their drawbacks
  - Detailed discussion on LoopPoint methodology
  - Experimental results of LoopPoint

NUS National University of Singapore intel

Simulation and Demo

- Speaker: Alen Sabu
  - PhD Candidate, National University of Singapore
- Topics Covered
  - High-level structure of LoopPoint code
  - Demo on how to use LoopPoint tools
  - Integrating workloads to run with LoopPoint

Agenda

Time	Speaker	Topic
13.20 to 13.30	Alen Sabu	Overview of the tutorial
13.30 to 14.30	Harish Patil	Tools & Methodologies: Pin, PinPlay, SDE, ELFies
14.30 to 15.00	Break	
15.00 to 15.50	Wim Heirman	Simulation with Sniper / Sniper 8.0 GitHub release
15.50 to 16.45	Alen Sabu	Single-threaded and Multi-threaded Sampling, LoopPoint
16.45 to 17.30	Alen Sabu	Running LoopPoint Tools

LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu[1], <u>Harish Patil</u>[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Computer Architecture, June 19th 2022, USA

Session 1

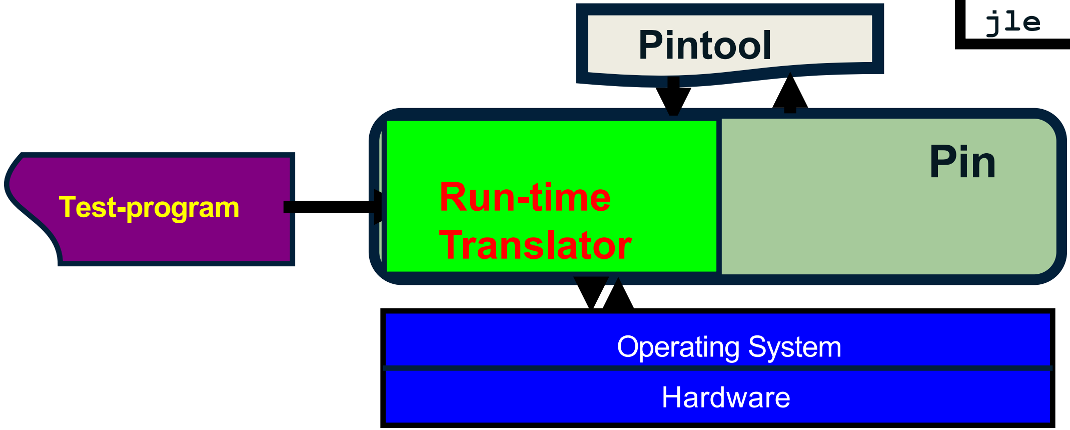
Tools and Methodologies

HARISH PATIL, PRINCIPAL ENGINEER (DEVELOPMENT TOOLS SOFTWARE)
INTEL CORPORATION

Pin: A Tool for Writing Program Analysis Tools

```
sub     $0xff, %edx
movl    0x8(%ebp), %eax
jle     <L1>
```

```
counter++; print(IP)
sub     $0xff, %edx
counter++; print(EA)
movl  0x8(%ebp), %eax
counter++;print(br_taken)
jle     <L1>
```




```
$ pin -t pintool -- test-program
```

Normal output +
Analysis output

http://pintool.intel.com

PinPlay: Software-based User-level Capture and Replay

Logger → **pinball** → Replayer + Pintool

Program/ Libraries

Input

License

No binaries/inputs

No application setup

No license checking

Platforms : Linux, Windows, MacOS

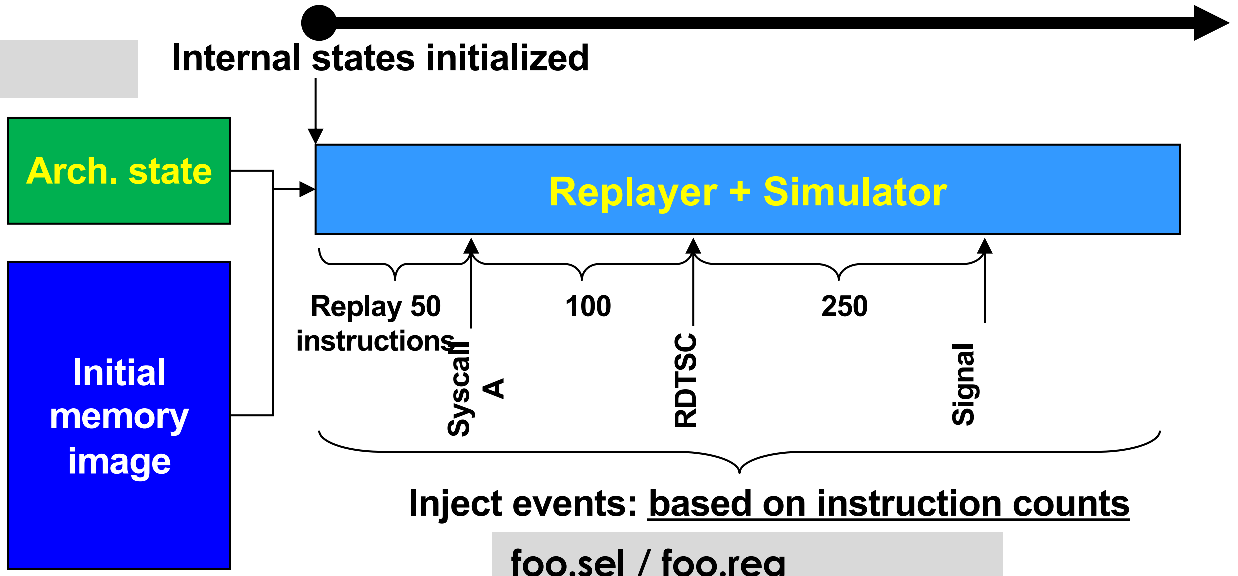
Upside : It works! Large OpenMP / MPI programs, Oracle

Downside : High run-time overhead: ~100-200X for capture ➔ Cannot be turned on all the time

http://pinplay.org

Pinball (single-threaded): Initial memory/register + injections

foo.reg



foo.sel / foo.reg

(injections)

foo.text

- **System calls** : skipped by injecting next rip/ memory changed
- **CPUID, RDTSC** : affected registers injected
- **Signals/Callbacks** : New register state injected

Pinball (multi-threaded): Pinball (single-threaded) + Thread-dependencies

foo.reg (per-thread)

Initial registers: T0

Initial registers: T1

Initial registers: T(n-1)

foo.text

Application Memory (common)

Event injection works only if same behavior (same instruction counts) is guaranteed during replay

foo.reg (per-thread)

foo.sel (per-thread)

[T1] 2 T2 2

[T1] 3 T2 3

[T2] 5 T4 1

Thread T2 cannot execute instruction 5 until T4 executes instruction 1

Thread T1 cannot execute instruction 2 until T2 executes instruction 2

foo.race (per-thread)

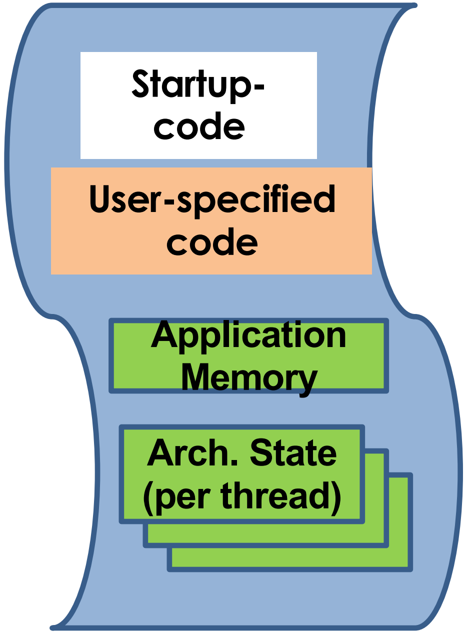
MT Pinball == race-files provide determinism

NUS National University of Singapore

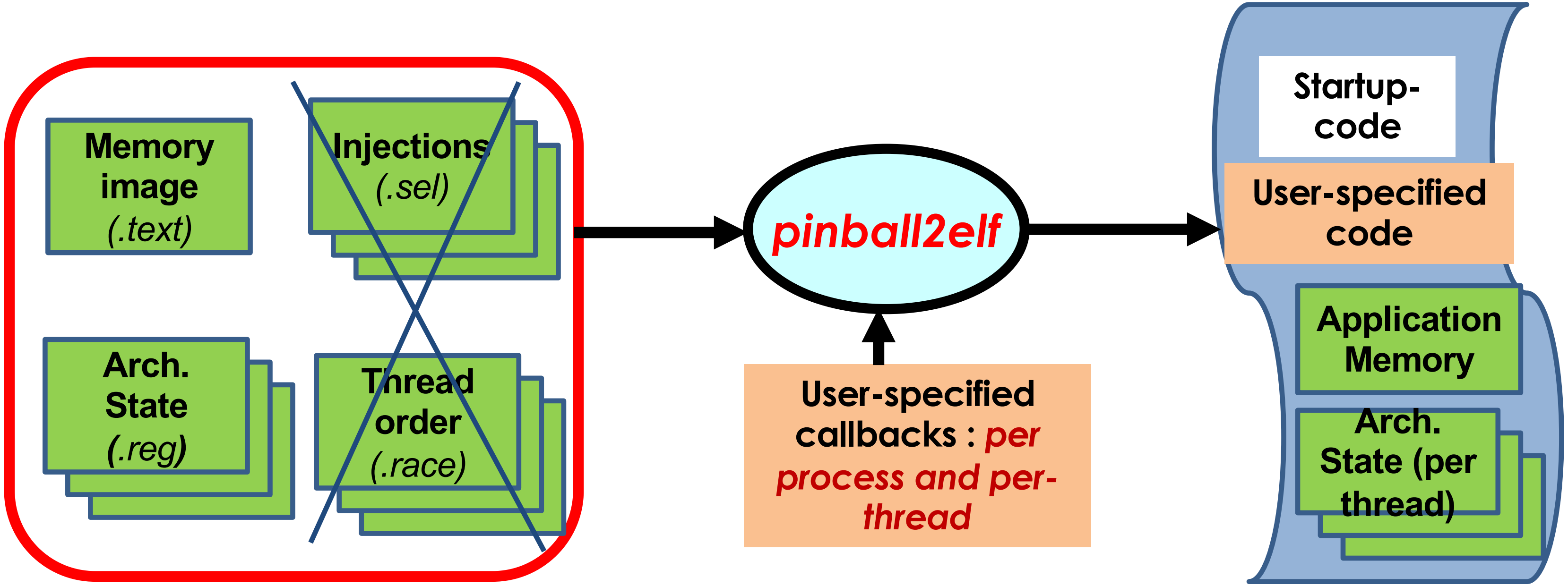
intel

ELFie : An Executable Application Checkpoint

- **Checkpoint:** Memory + Registers
- **Application** : Only program state captured -- no OS or simulator states
- **Executable** : In the Executable Linkage Format commonly used on Linux



pinball2elf: Pinball converter to ELF



http://pinelfie.org

Getting started with *pinball2elf*

Prerequisite: '*perf*' installed on your Linux box (*perf stat /bin/ls* should work)

- Clone pinball2elf repository: *git clone https://github.com/intel/pinball2elf.git*
- *cd pinball2elf/src*
- *make all*
- *cd ../examples/ST*
- *./testST.sh*

Running ../../scripts//pinball2elf.basic.sh pinball.st/log_0

..

Running ../../scripts//pinball2elf.perf.sh pinball.st/log_0 st

export ELFIE_PERFLIST=0:0,0:1,1:1

…

hw_cpu_cycles:47272 hw_instructions:4951 sw_task_clock:224943

Tested : Ubuntu 20.04.4 LTS : gcc/g++ 7.5.0 and 9.4.0

and Ubuntu 18.04.6 LTS: gcc/g++ 7.5.0

ELFie types: basic, sim, perf

	basic	sim	perf
How to create	scripts/pinball2elf.basic.sh pinball	scripts/pinball2elf.sim.sh pinball	scripts/pinball2elf.perf.sh pinball perf.out
Exits gracefully?	NO, either hangs or dumps core	NO, either hangs or dumps core Simulator handles exit	YES, when retired instruction count reaches pinball icount
Environment variables used	NONE	ELFIE_VERBOSE=0/1 ELFIE_COREBASE=X Set affinity : thread 0 → core X, thread 1 → core x+1	"ELFIE_WARMUP" to decide whether to use warmup "ELFIE_PCCONT" to decide how to end warmup/simulation regions ELFIE_PERFLIST, enables performance counting

Optional: Operating system state (SYSSTATE) per pinball: pintools/PinballSYSState [See CGO2021 ELFie paper]

NUS National University of Singapore intel

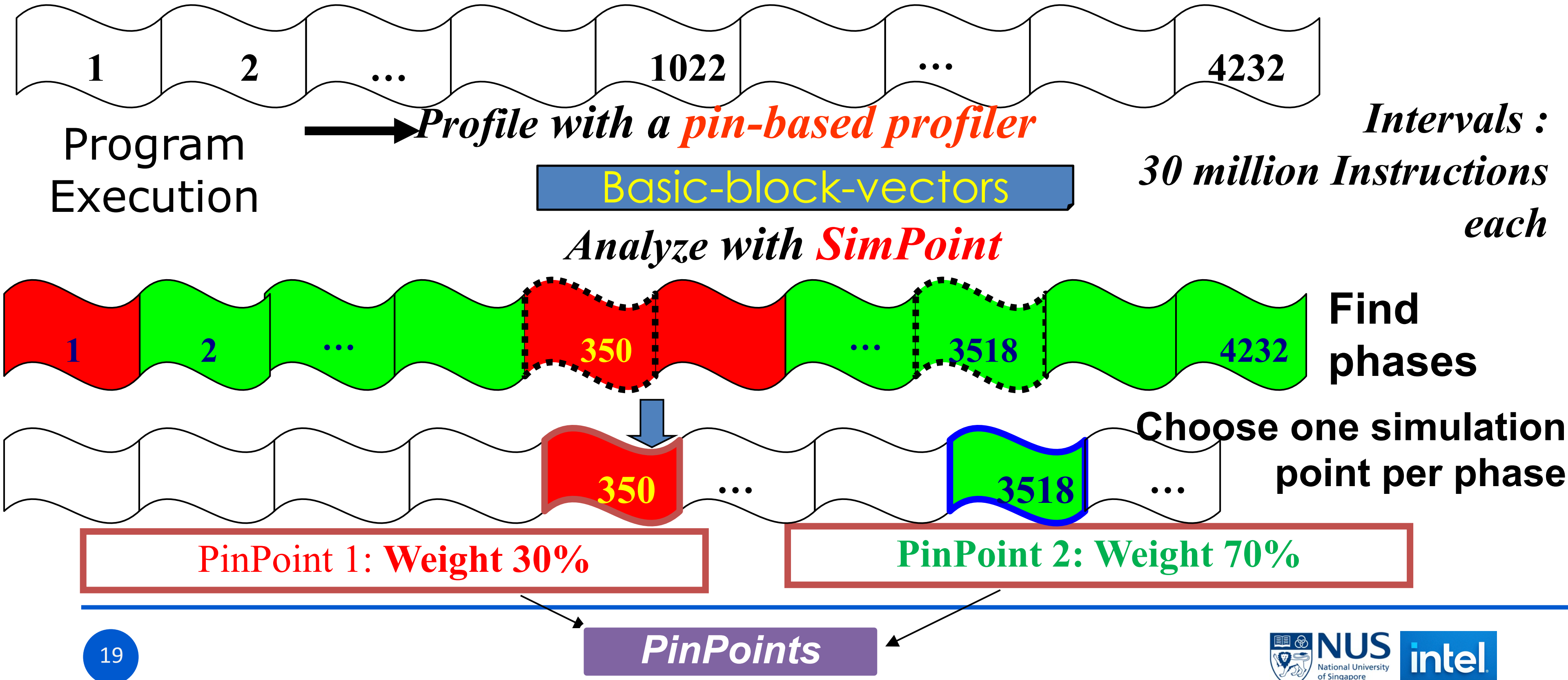
Example: *ELFIE_PERFLIST* with a *perf ELFie*

ELFIE_PERFLIST, enables performance counting
(based on /usr/include/linux/perf_event.h
perftype: 0 --> HW 1 --> SW
HW counter: 0 --> PERF_COUNT_HW_CPU_CYCLES
HW counter: 1 --> PERF_COUNT_HW_CPU_INSTRUCTIONS
SW counter: 0 --> PERF_COUNT_SW_CPU_CLOCK
... <see perf_event.h:'enum perf_hw_ids' and 'enum perf_sw_ids')

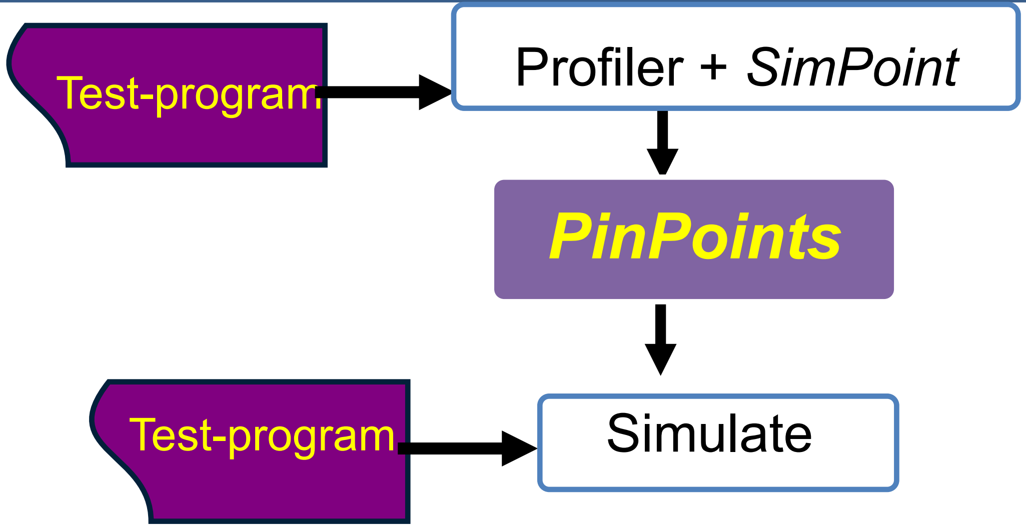
```
% cd examples/MT
% ../../scripts/pinball2elf.perf.sh pinball.mt/log_0 perf.out
% setenv ELFIE_PERFLIST "0:0,0:1,1:1"
% pinball.mt/log_0.perf.elfie
├── perf.out.0.perf.txt
├── perf.out.1.perf.txt
├── perf.out.2.perf.txt
```

```
ROI start: TSC 480511105862177756
Thread start: TSC 480511106238434452
-------------------------------------------------
Simulation end: TSC 480511106250453222
      Sim-end-icount 3436
hw_cpu_cycles:36148 hw_instructions:3476
sw_task_clock:141901
-------------------------------------------------
Thread end: TSC 480511106253665022
ROI end: TSC 480511106259593644
hw_cpu_cycles:40097 hw_instructions:4455
sw_task_clock:188637
```

PinPoints == Pin + SimPoint



PinPoints : The repeatability challenge

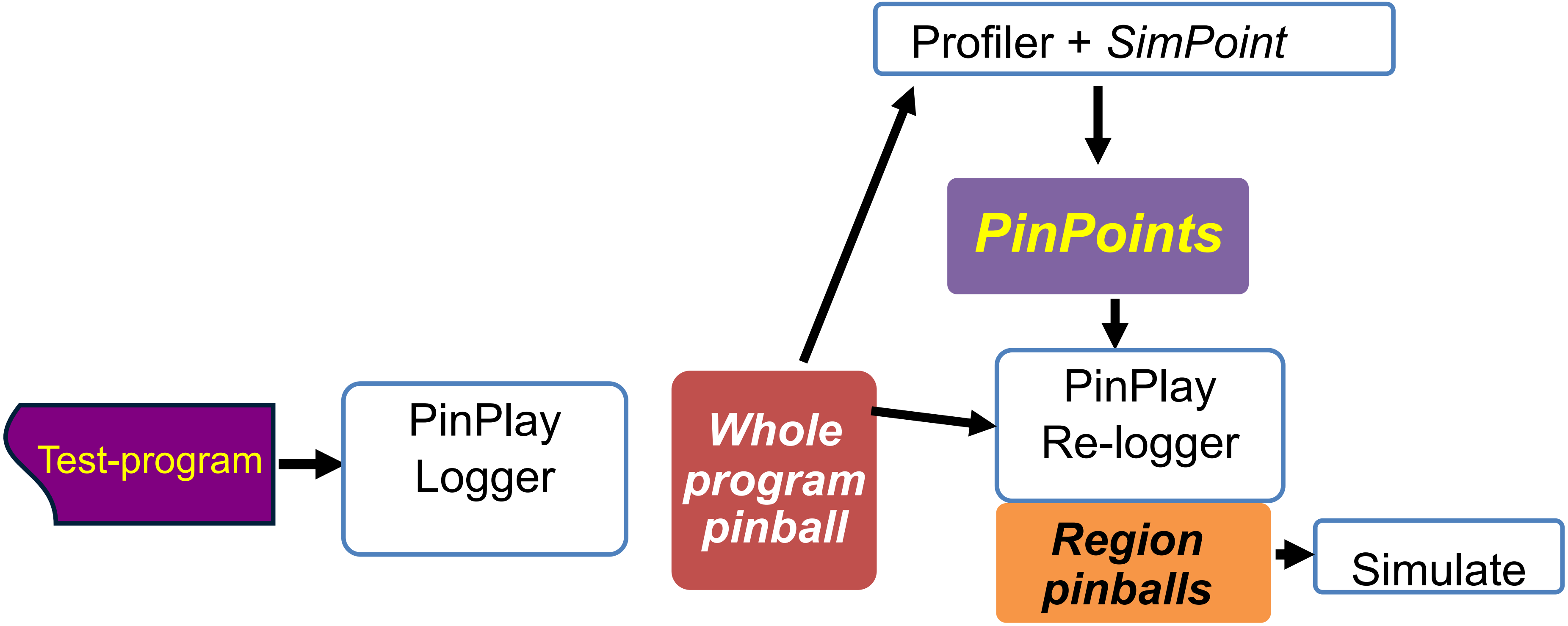


Problem: Two runs are not exactly same → PinPoints missed (PC marker based)

[*"PinPoints out of order" "PinPoint End seen before Start"*]

Found this for 25/54 SPEC2006 runs!

PinPlay provides repeatability



Single-threaded *PinPoints* → SPEC2006/2017 pinballs publicly available

1. University of California (San Diego), Intel Corporation, and Ghent University
   https://www.spec.org/cpu2006/research/simpoint.html
2. University of Texas at Austin
   https://www.spec.org/cpu2017/research/simpoint.html
3. Northwestern University

   Public Release and Validation of SPEC CPU2017 PinPoints

Simulation of multi-threaded Programs: The non-determinism challenge

- Runs across different configurations are non-deterministic [Alameldeen'03]
  - Locks are acquired in different order
  - Unprotected shared-memory accesses
- One can't compare two runs/simulations of the same benchmark directly

  →***Change in micro-architecture present/simulated or execution path taken?***

1.Alameldeen'03 Variability in Architectural Simulations of Multi-threaded Workloads (HPCA2003)

Dealing with non-determinism

1. Run multiple simulations for each studied configuration [Alameldeen’03]
   - Needs random perturbation for each run
   - Average behavior per configuration
   - Cost: multiple runs

- 2. Force deterministic behavior so that one run in each configuration is performed [Pereira’08 @ Intel]
   - Same execution paths
   - Cost: loss in fidelity, thread behavior tied to tracing machine

- 3. Simulate the same “amount of work” [Alameldeen’06] : *LoopPoint* approach

A. Pereira’08: Reproducible Simulation of Multi-Threaded Workloads for Architecture Design Exploration, International Symposium on Workload Characterization (IISWC'08)

B. Alameldeen’06 IPC Considered Harmful for Multi-processors Workloads (IEEE-Micro-2006)

LoopPoint: Key idea 1: Filtering Synchronization Code during profiling

Why: Profiling should look only at 'real work'

What: Skip profiling of synchronization code

How?

- Automatically with Loop Analysis: Very hard

  "Spin Detection Hardware for Improved Management of Multithreaded Systems"
  Transactions on Parallel and Distributed Systems, 2006

  - **Look for loops that do not update architectural state**
  - Was implemented in Sniper(Pin-2) but many OpenMP spin loops maintain stats hence do update architecture state

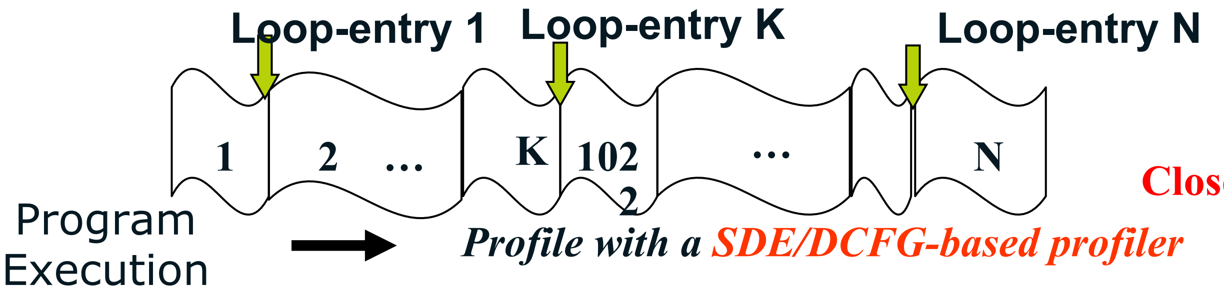
✓ **Heuristic**

  - Filter synchronization library code: e.g. libiomp5.so, libpthread.so

LoopPoint: Key idea 2: Loops as 'Units of work'

Why: Property of program/binary : independent of architecture

Profiling



Variable length intervals

Close to desired length : *-sliceSize S*

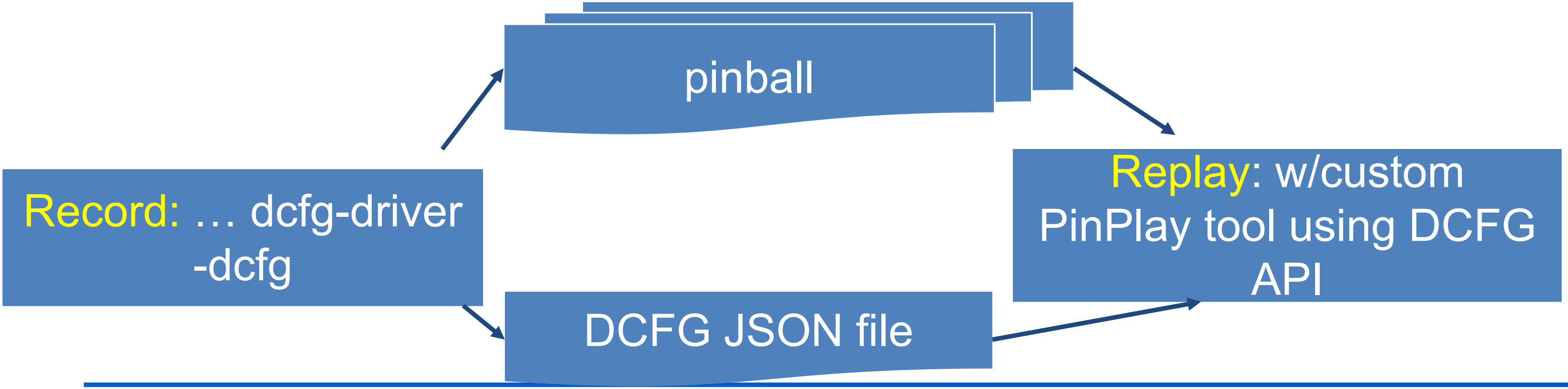
Profile with a SDE/DCFG-based profiler

- Global counting of loop-entries
- Region start/stop : only in the main image
  - Stop when 'desired global instruction count' (SliceSize) is reached
  - Do not count instructions in synchronization library

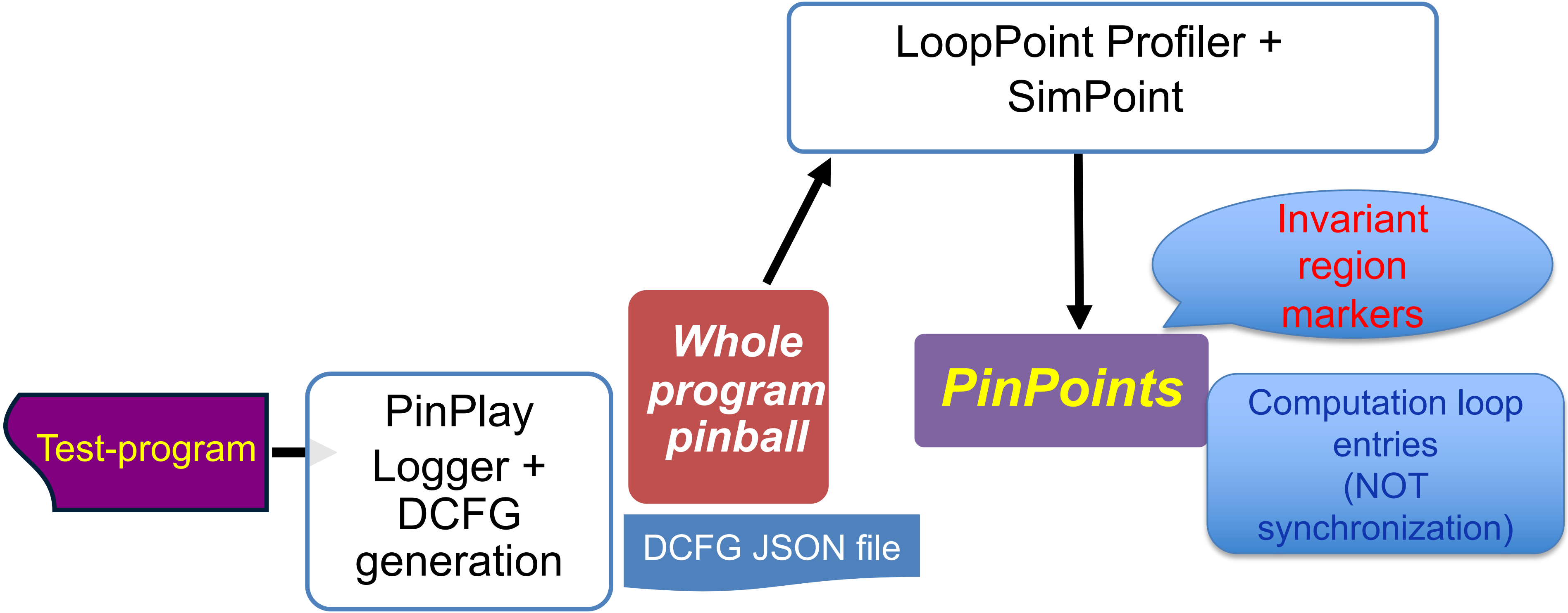
DCFG Generation with *PinPlay*

Dynamic Control-Flow Graph (DCFG)

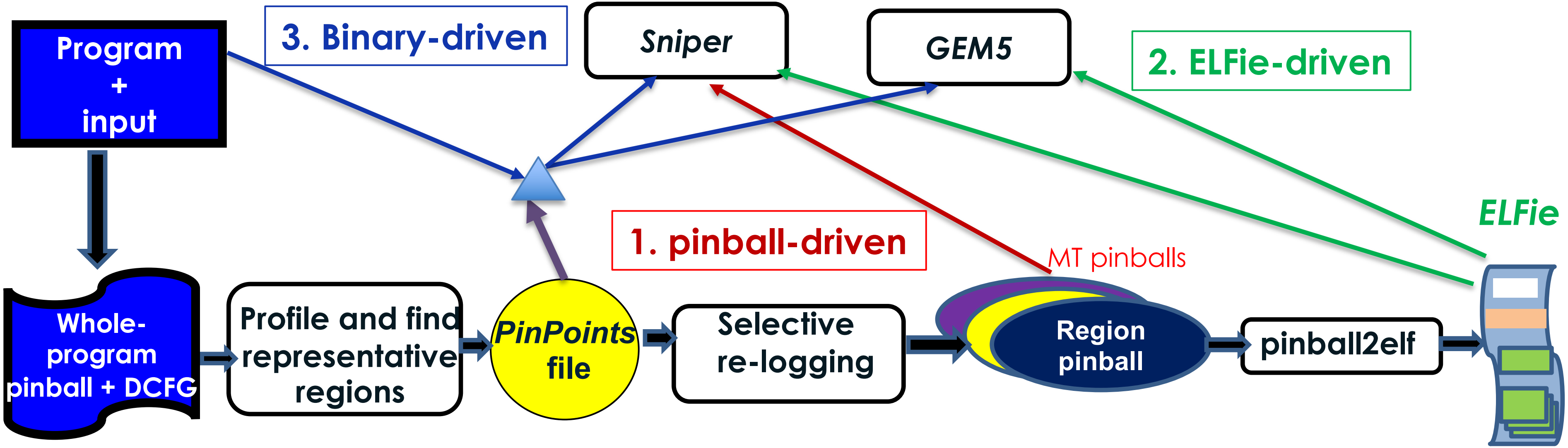
Directed graph extracted for a specific execution:
Nodes ➔ basic blocks
Edges ➔control-flow : augmented with per-thread execution counts



PinPlay + DCFG : Stronger Repeatability



LoopPoint: Simulation alternatives



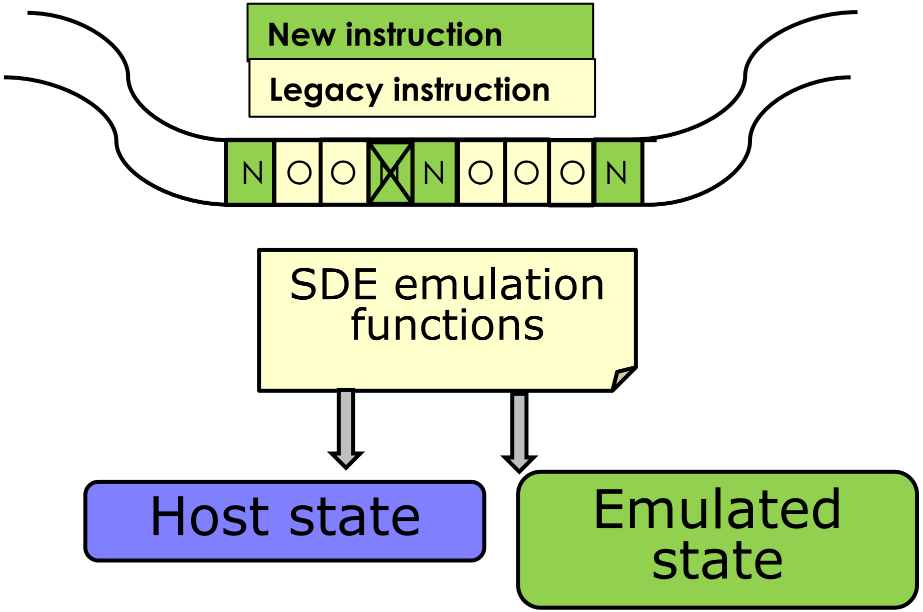
Requirement: Execution invariant region specification
(PC+count for compute loop entries)

Intel Software Development Emulator (*Intel SDE*)

- The Intel® Software Development Emulator is **a functional user-level (ring 3) emulator** for x86 (32b and 64b) new instructions built upon Pin and XED (X86 encoder/decoder)
- <u>**Goal**</u>: New instruction/register emulation between the time when they are designed and when the hardware is available.
- Used for compiler development, architecture and workload analysis, and tracing for architecture simulators
- No special compilation required
- Supported on Windows/Linux/Mac OS
- Runs only in user space (ring 3)

How *SDE* Works

- Based on Pin (http://pintool.intel.com) and XED decoder/encoder (https://github.com/intelxed/xed)
- Instrument new instructions
  - Add call to emulation routine
  - Delete original instruction
- Emulation routine:
  - Update native state with emulated state



http://www.intel.com/software/sde

Using *SDE* for *PinPoints* and *LoopPoint*

Prerequisites:

1. SDE build kit (version 9.0 or higher) from Intel
   http://www.intel.com/software/sde
2. pinplay-tools from Intel
   https://github.com/intel/pinplay-tools
3. SimPoint sources from UCSD
   https://cseweb.ucsd.edu/~calder/simpoint/
4. Pinball2elf sources from Intel
   http://pinelfie.org → https://github.com/intel/pinball2elf

Getting ready for *LoopPoint* …

1. Expand SDE build-kit : *setenv SDE_BUILD_KIT<path to SDE kit>*
2. *cp –r pinplay-tools/pinplay-scripts $ SDE_BUILD_KIT*
3. Build simpoint (see pinplay-tools/pinplay-scripts/README.simpoint)
   - *cp <path>/SimPoint.3.2/bin/simpoint $ SDE_BUILD_KIT/pinplay-scripts/PinPointsHome/Linux/bin/*
4. Build global looppoint tools
   - *setenv PINBALL2ELF <path to pinball2lef repo>*
   - *cd pinplay-tools/GlobalLoopPoint*
   - *./sde-build-GlobalLoopPoint.sh*

SDE kit expanded for *LoopPoint*

```
sde-external-9.0.0-2021-11-07-lin
├── ...
├── intel64
│         ├── sde-global-event-icounter.so
│         ├── sde-global-looppoint.so
├── ...
└── pinplay-scripts

        PinPointsHome/
         └── Linux
         └── bin
         ├── LICENSE.simpoint
         ├── simpoint
```

Running *LoopPoint* for an *OpenMP* program

- *cd pinplay-tools/dotproduct-omp* # see README there
- *make* # builds dotproduct-omp → base.exe
- *./sde-run.looppoint.global_looppoint.concat.filter.flowcontrol.sh*

```
~/pinplay-tools/dotproduct-omp
    ├── dotproduct.1_282016.Data
    ├── dotproduct.1_282016.pp
    └── whole_program.1
```

bbv files (.bb), PinPoints file(*.csv, *.CSV)*

Region pinballs

Whole-program pinball + DCFG

Create LoopPoint region pinballs and replay them

Summary: Simulation of Multi-threaded Programs: Tools & Methodologies

Where to simulate?

SDE + *LoopPoint*
Compute-loop iterations as
`Unit of work'

How to simulate?

1. **Pinball-driven**
2. **ELFie-driven**
3. **Binary-driven**

Are the regions representative?

1. Simulation (Sniper) -based
2.ELFie-based / Binary+*ROIPerf* (*not covered*)
Whole-program performance vs
Region-predicted performance

Agenda

Time	Speaker	Topic
13.20 to 13.30	Alen Sabu	Overview of the tutorial
13.30 to 14.30	Harish Patil	Tools & Methodologies: Pin, PinPlay, SDE, ELFies
14.30 to 15.00	Break	
15.00 to 15.50	Wim Heirman	Simulation with Sniper / Sniper 8.0 GitHub release
15.50 to 16.45	Alen Sabu	Single-threaded and Multi-threaded Sampling, LoopPoint
16.45 to 17.30	Alen Sabu	Running LoopPoint Tools

LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Computer Architecture, June 19th 2022, USA

Session 2

Simulation with Sniper / Sniper 8.0 GitHub release

WIM HEIRMAN, PRINCIPAL ENGINEER (EXTREME SCALE COMPUTING)

INTEL CORPORATION

Architectural Trends in Processor Design

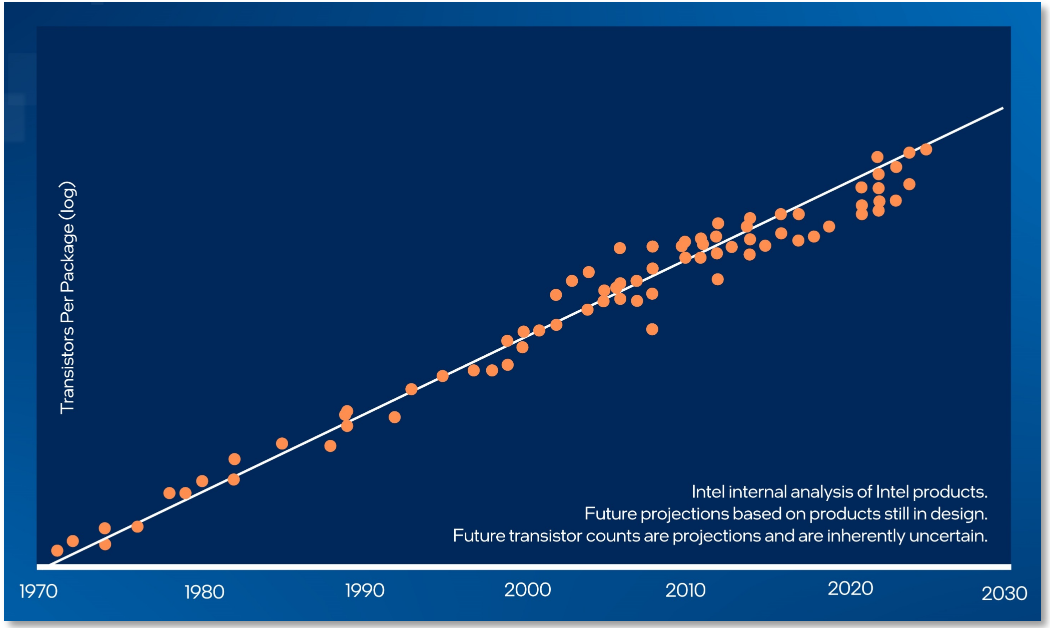


Fig. 1: Moore Law number of transistor per device: past, present, future [Intel]

- Moore's Law predicts that the number of transistors per device will double every two years.
- First microprocessor had 2200 transistors – Intel aspiring to have 1 trillion transistors by 2030.

Source: https://www.intel.com

Architectural Trends in Processor Design

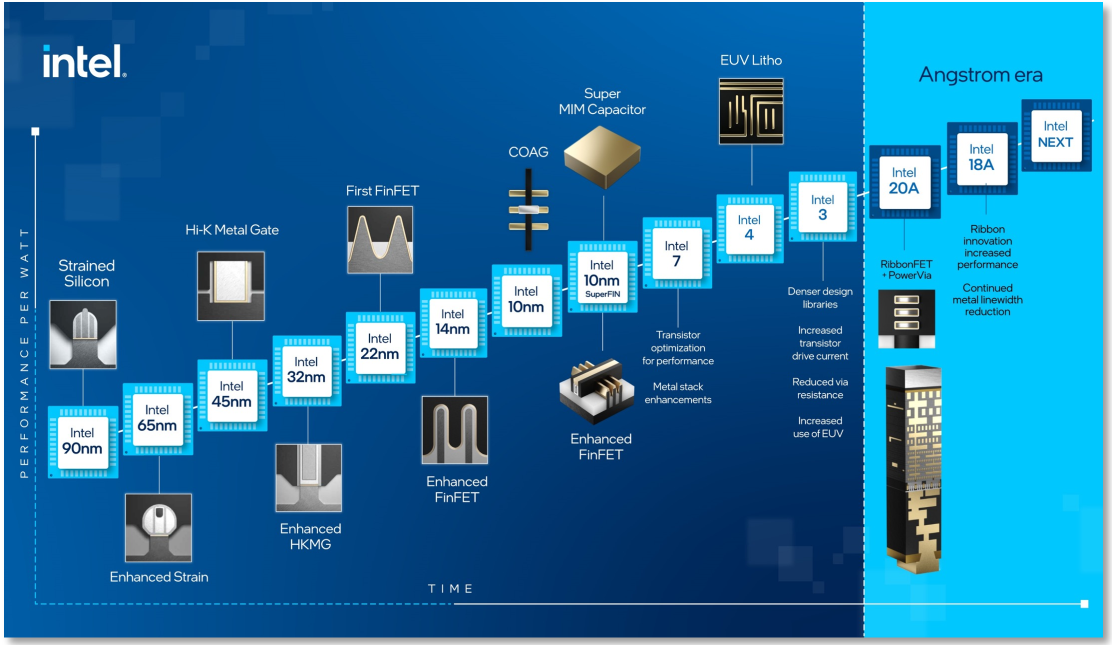


Fig. 2: Transistor innovations over time

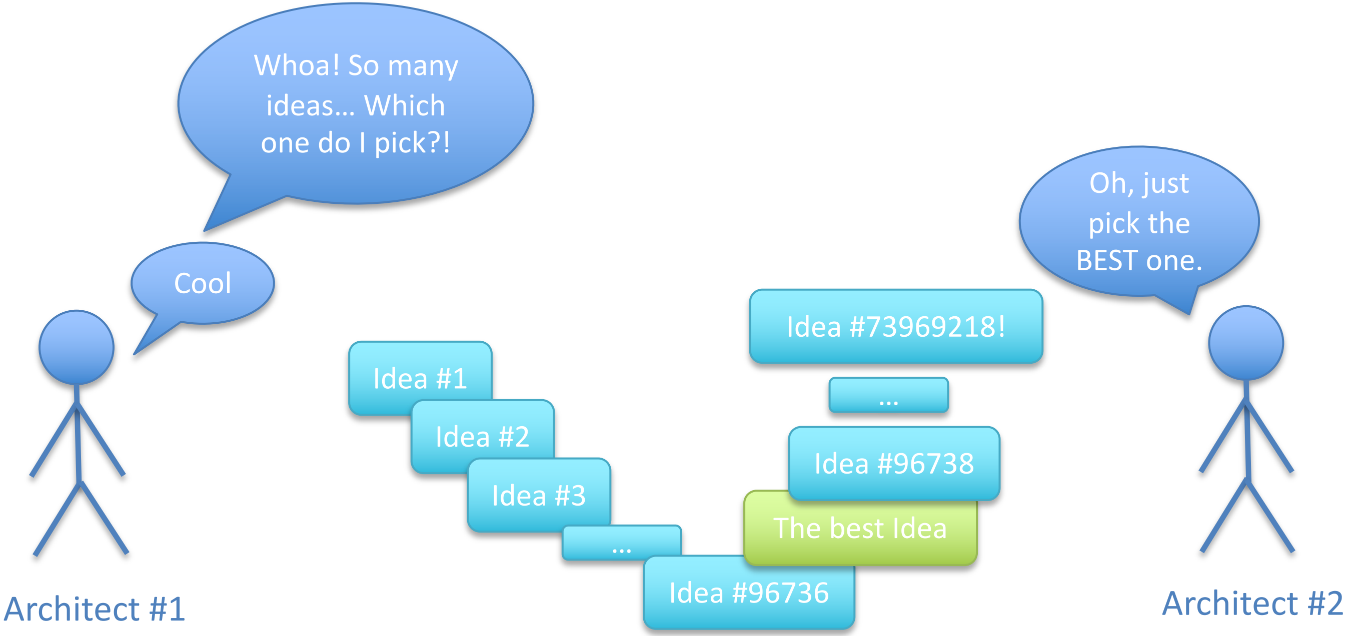
Main Goal: Meeting the ever-increasing computational demands *while* adhering to stringent non-functional requirements (ex: size, power)!

Source: https://www.intel.com/

Exploration and Evaluation of New Ideas

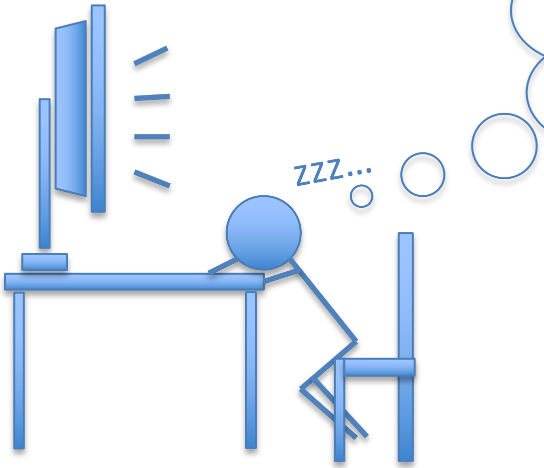
- Architecture is rapidly evolving domain with a lot of new research directions.
- A plethora of design choices are available:
  - Ranging from the choice of components, the choice of operating modes of each component, the choice of interconnects used, the choice of algorithms employed, etc.
- The process of exploration and evaluation of new ideas is often complex and time-consuming.

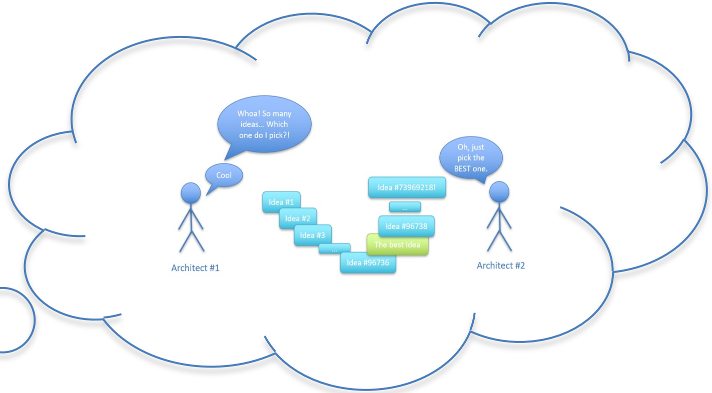
Exploration and Evaluation of New Ideas



Exploration and Evaluation of New Ideas

The Architect IRL



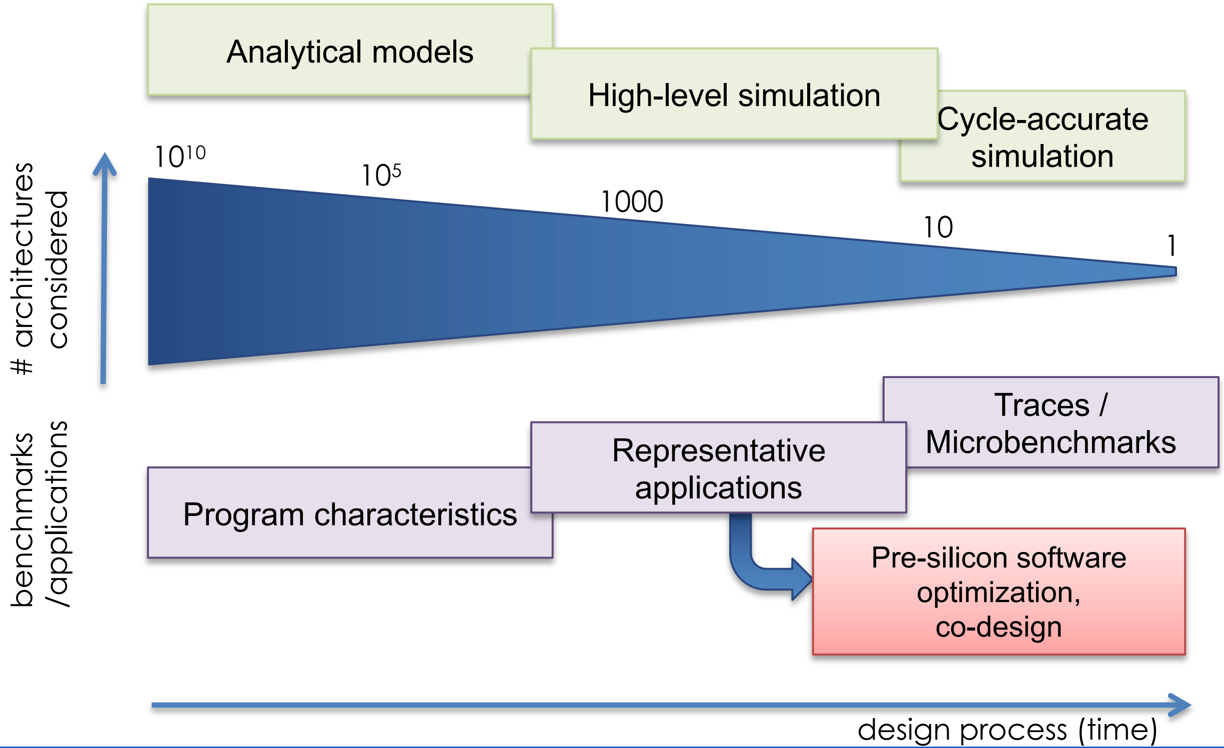


<u>The Important Question:</u>

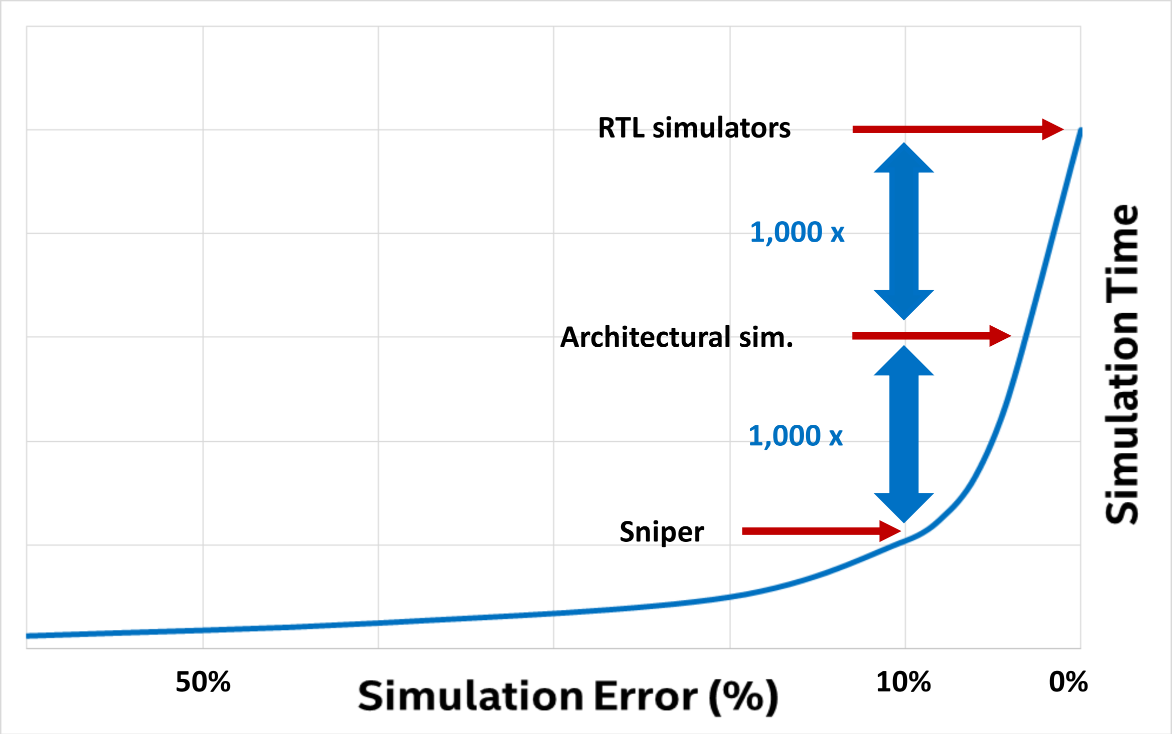
So how do we then explore new ideas quickly and evaluate them accurately to find the *BEST* idea?

NUS National University of Singapore intel

The Architect's Tools – Design Waterfall

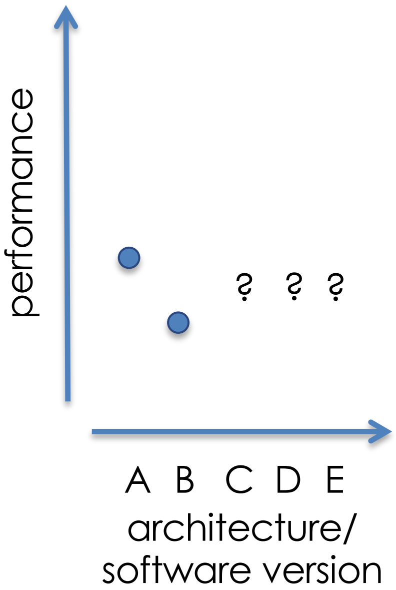


Fast or accurate?

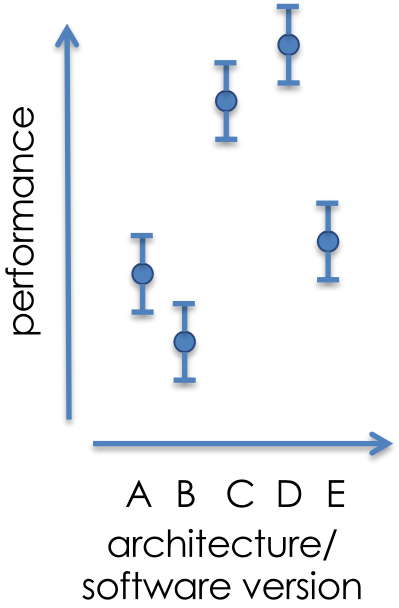


Fast or Accurate Simulation?

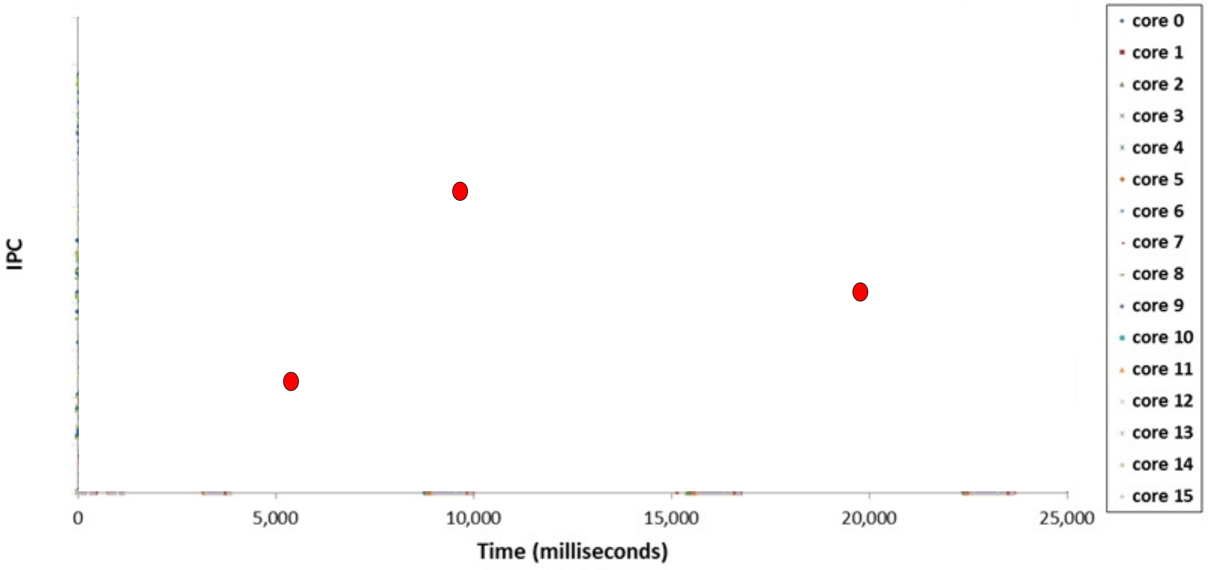
Cycle-accurate simulator



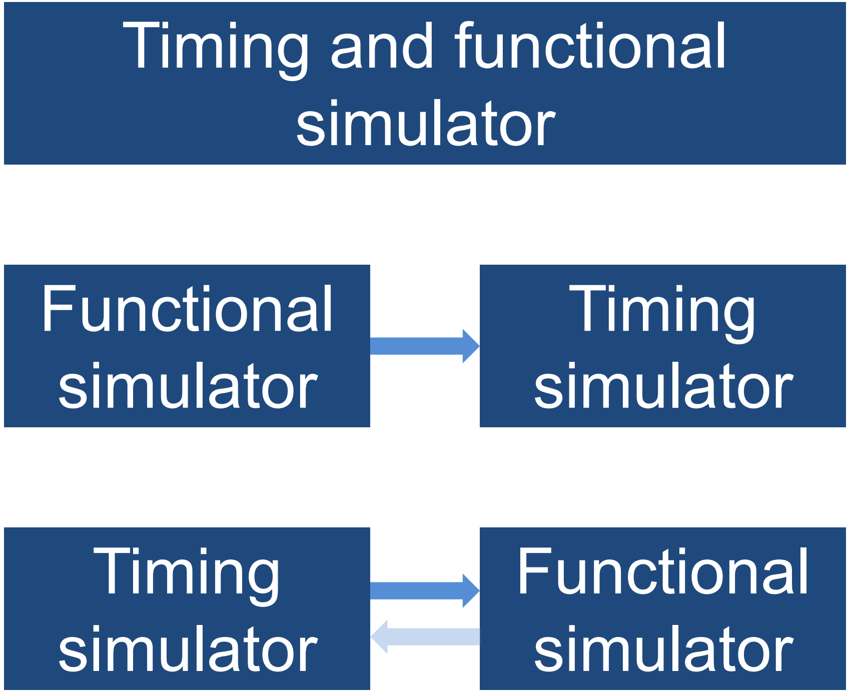
Higher-abstraction level simulator



Fast or Accurate Simulation?



Simulator taxonomy



- Integrated
  - Complex, incl. wrong-path, races
- Functional-first
  - Trace-driven, or timing feedback
- Timing-directed, timing-first
  - Step & verify

Mauer, Hill & Wood. Full-System Timing-First Simulation. SIGMETRICS 2002

Sniper History

- August 2010: Sniper forked from MIT Graphite
- November 2011: SC'11 paper, first public release
- Today:
  - Interval and Instruction-window-centric core models
  - 7000+ downloads from 100+ countries
  - Active mailing list
  - 1200+ citations (SC'11 & TACO'12 papers)

snipersim.org downloads by quarter

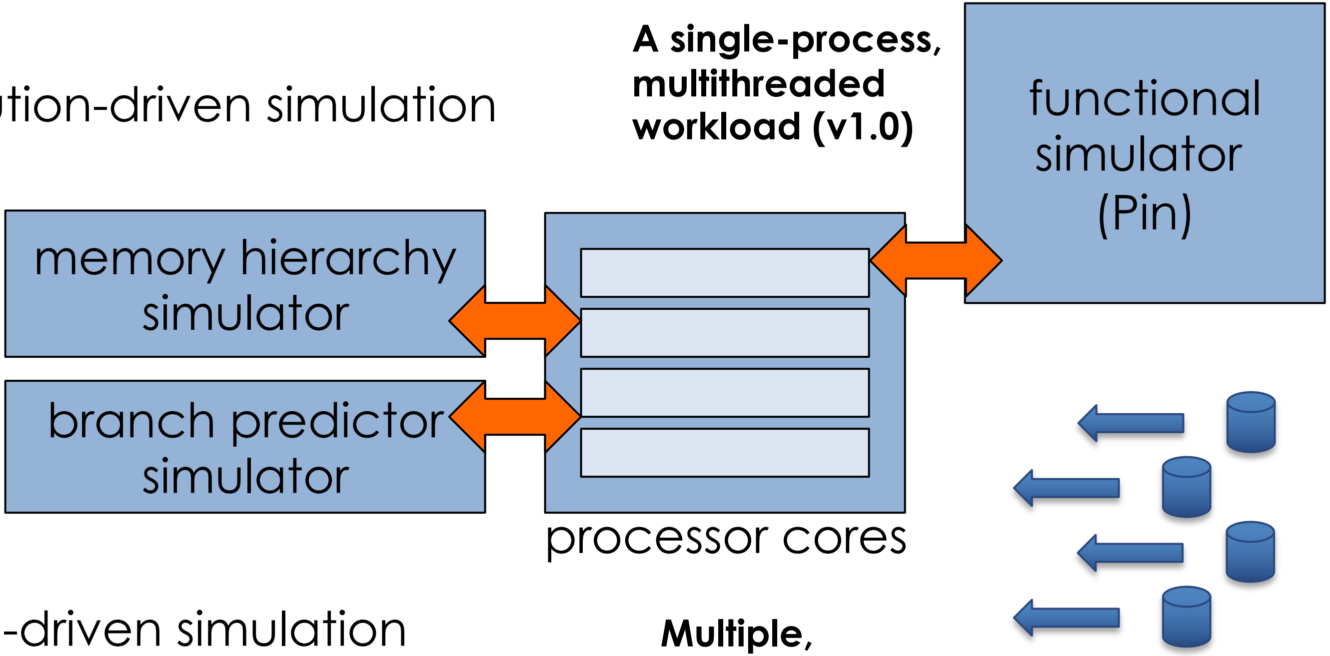
Functional-first with timing feedback

- Functional-first
  - Build on production-quality functional simulator / instrumentation tool
    - Pin/SDE, Simics, SAE [x86], Spike, rv8 [RISC-V]
    - 99/1 rule: 99% of instructions must be correct to get failure rate <1%
  - Extensible timing model
    - 1/99 rule: modeling 1% of the ISA is enough to capture 99% of performance trends
    - Easy to defeature / sweep accuracy
      - From 1-IPC (fast, just counting instructions)…
      - …to near-cycle-accurate
      - Perfect / oracle simulation (perfect caches, perfect branches, etc.)
- Timing feedback
  - Multi-core, relative progress must be sync'd back to functional for e.g. load balancing

Simulation in Sniper

Execution-driven simulation

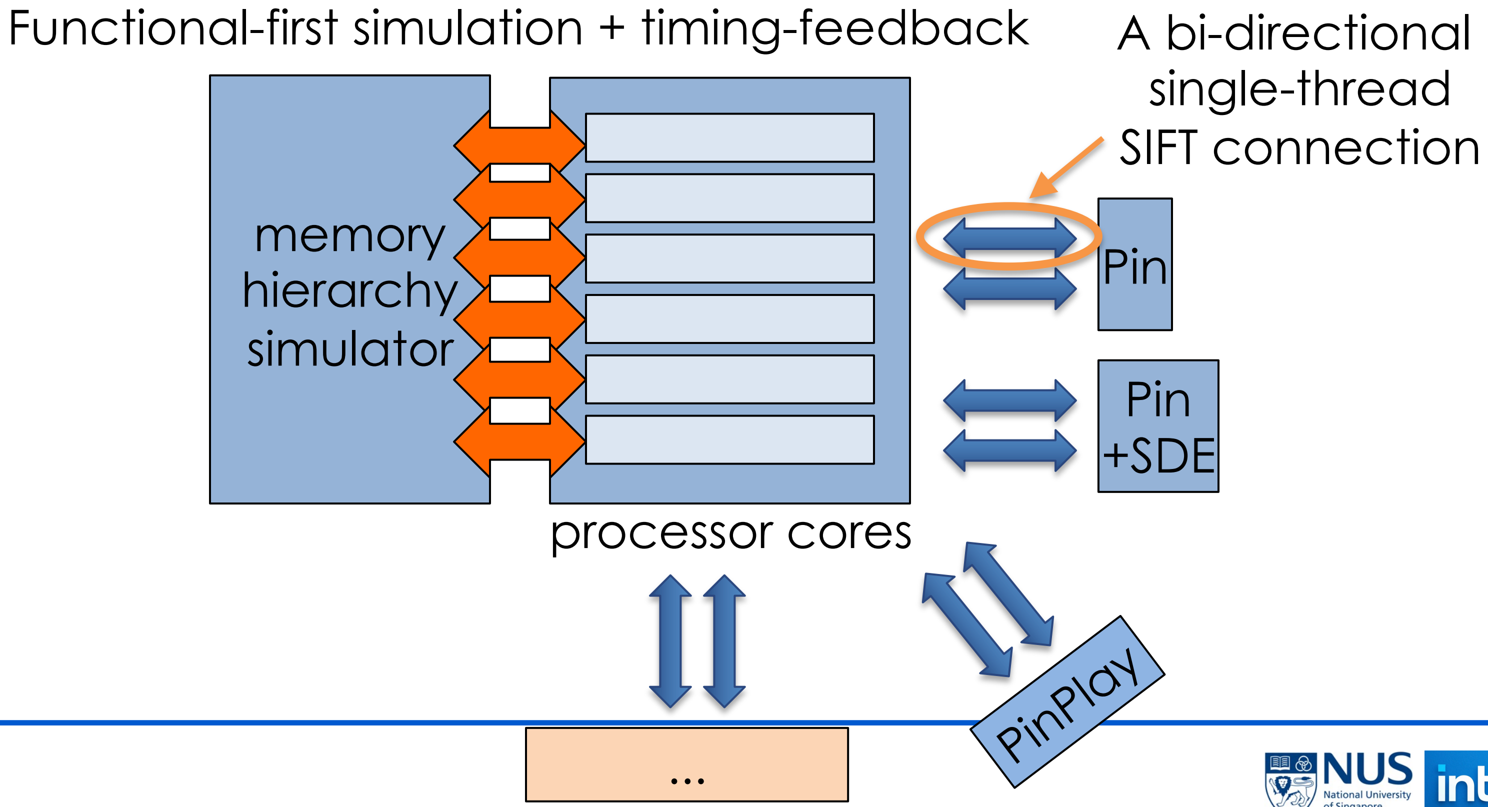
A single-process, multithreaded workload (v1.0)



Trace-driven simulation

Multiple, single-threaded workloads (v2.0)

Simulation in Sniper with SIFT



Running Sniper

Configuration Region of interest markers in code Workload command line

```
$ run-sniper -c gainestown --roi -- ./test/fft/fft -p2
[SNIPER] Start
[SNIPER] --------------------------------------------------------------------------------
[SNIPER] Sniper using Pin frontend
[SNIPER] Running pre-ROI region in  CACHE_ONLY mode
[SNIPER] Running application ROI in DETAILED mode
[SNIPER] Running post-ROI region in FAST_FORWARD mode
[SNIPER] --------------------------------------------------------------------------------

FFT with Blocking Transpose
   1024 Complex Doubles
   2 Processors

[SNIPER] Enabling performance models
[SNIPER] Setting instrumentation mode to DETAILED
[SNIPER] Disabling performance models
[SNIPER] Leaving ROI after 2.08 seconds
[SNIPER] Simulated 1.1M instructions, 0.9M cycles, 1.22 IPC
[SNIPER] Simulation speed 545.5 KIPS (272.8 KIPS / target core - 3666.2ns/instr)
[SNIPER] Setting instrumentation mode to FAST_FORWARD

                 PROCESS STATISTICS
...
[SNIPER] End
[SNIPER] Elapsed time: 5.97 seconds
```

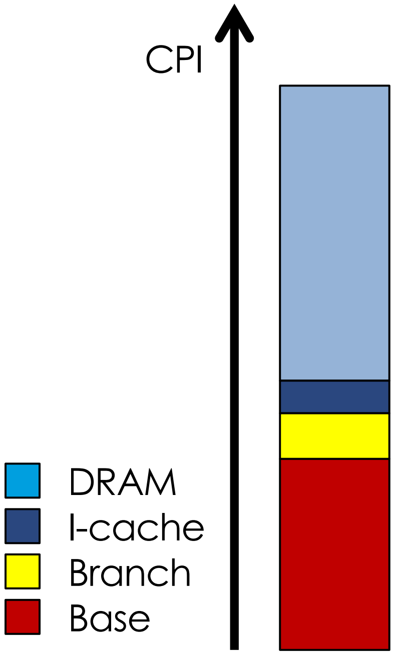
Simulation results

`sim.out`: Quick overview of basic performance results

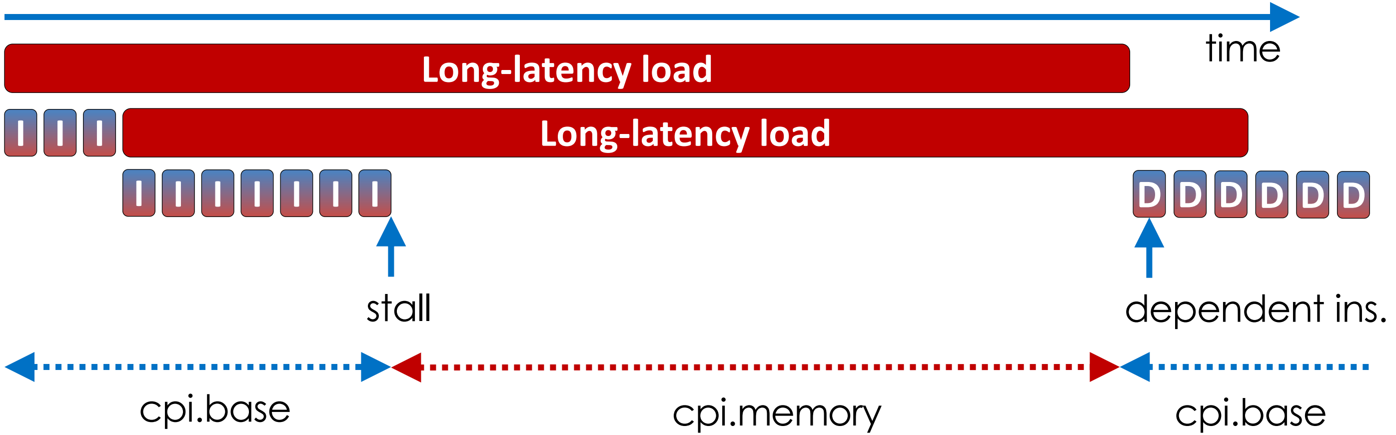
```
                                     | Core 0    | Core 1
  Instructions                       |    506505 |    505562
  Cycles                             |    469101 |    468620
  Time (ns)                          |    176354 |    176173
Branch predictor stats               |           |
  num incorrect                      |      1280 |      1218
  misprediction rate                 |     7.70% |     7.42%
  mpki                               |      2.53 |      2.41
Cache Summary                        |           |
  Cache L1-I                         |           |
    num cache accesses               |     46642 |     46555
    num cache misses                 |       217 |       178
    miss rate                        |     0.47% |     0.38%
    mpki                             |      0.43 |      0.35
  Cache L1-D                         |           |
    num cache accesses               |    332771 |    332412
    num cache misses                 |       517 |       720
    miss rate                        |     0.16% |     0.22%
    mpki                             |      1.02 |      1.42
  Cache L2                           |           |
    num cache accesses               |       984 |      1090
    num cache misses                 |       459 |       853
```

Cycle stacks

- Where did my cycles go?
  - Cycles/time per instruction
  - Broken up in components
    - Base: ideal execution, no bottlenecks
    - Add “lost” cycles do to each HW structure
  - Normalize by either
    - Number of instructions (CPI stack)
    - Execution time (time stack)
- *Different from miss rates*:
  cycle stacks directly quantify the effect on performance
- (Also: top-down analysis in VTune)



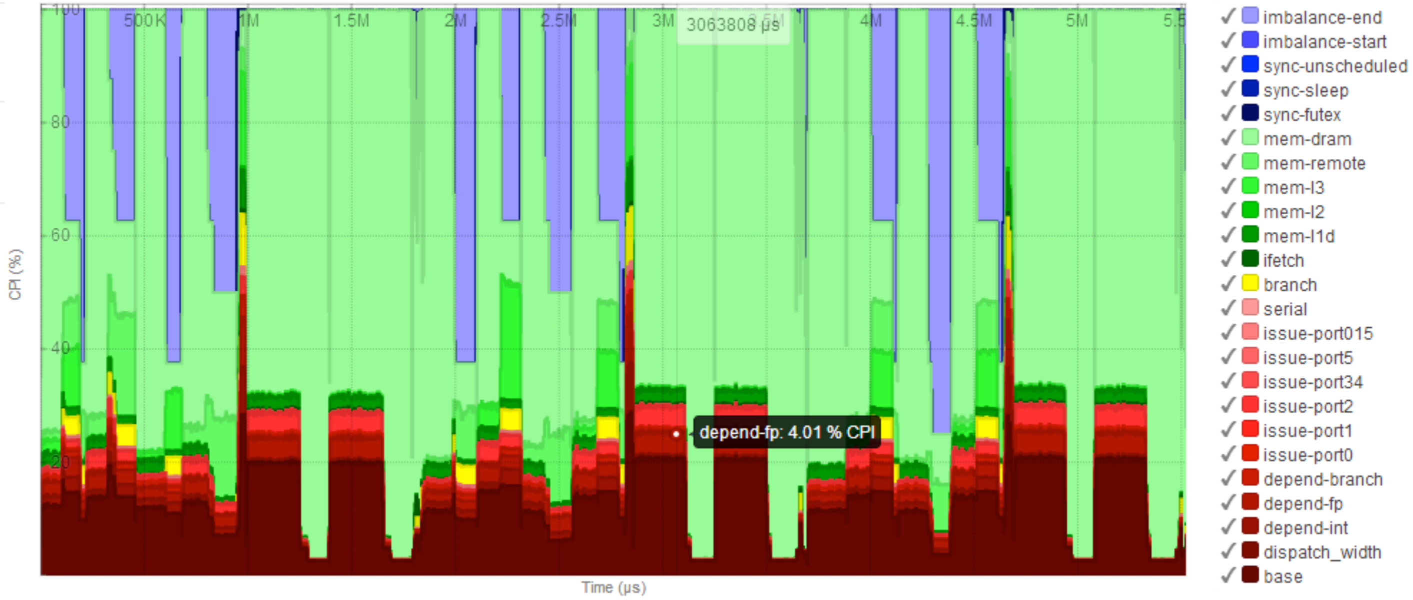
Miss rates vs. CPI stacks



- Miss rate x latency overestimates penalty
  - Ignores overlap with compute, indep. memory accesses
  - Can lead to wrong conclusions / useless optimization
- CPI stack takes overlap into account

Advanced visualization

- Cycle stacks through time



Improved visibility vs. hardware

Hardware

- 2011: Ask architects for a new FLOPS performance counter
- 2014: Haswell: broken…
- 2017: Skylake: success!

Simulator

```
$ git diff
void Core::init()
+   registerMetric(“core”, _id, “flops”, &flops);
void Core::doCommit(MicroOp &uop)
+   flops += uop.fp_operations();
$ make
$ run-sniper -- ./my_app
$ dumpstats | grep flops
core.0.flops 123456
core.1.flops 234567
```

Sniper 8.0 release on GitHub

- New in Sniper 8.0 release:
  - Support for Intel SDE in addition to Intel Pin (emulation)
  - License now allows for redistribution of Sniper (also Pin, SDE) in Docker containers, artifacts, …
  - Available on GitHub: https://github.com/snipersim/snipersim

Agenda

Time	Speaker	Topic
13.20 to 13.30	Alen Sabu	Overview of the tutorial
13.30 to 14.30	Harish Patil	Tools & Methodologies: Pin, PinPlay, SDE, ELFies
14.30 to 15.00	Break	
15.00 to 15.50	Wim Heirman	Simulation with Sniper / Sniper 8.0 GitHub release
15.50 to 16.45	Alen Sabu	Single-threaded and Multi-threaded Sampling, LoopPoint
16.45 to 17.30	Alen Sabu	Running LoopPoint Tools

LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Computer Architecture, June 19th 2022, USA

Session 3

Sampled Simulation and LoopPoint

ALEN SABU, PHD CANDIDATE
NATIONAL UNIVERSITY OF SINGAPORE

Techniques to Simulate Faster

- Partial simulation and extrapolation
  - Simulating the first 1 billion instructions in detail.

  - Fast-forwarding to skip the initialization phase and then simulating 1 billion instructions in detail.

  - Fast-forwarding to skip the initialization phase, microarchitectural state warming, and then simulating the 1 billion instructions in detail

Techniques to Simulate Faster

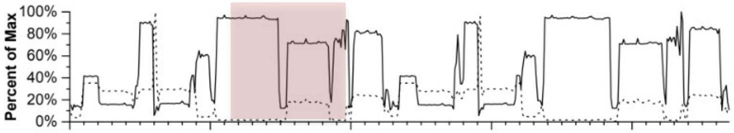
- Workload reduction
  - Simulating for reduced input sets
  - Simulating for reduced loop counts in workloads

Techniques to Simulate Faster

- Workload reduction
  - Simulating for reduced input sets
  - Simulating for reduced loop counts in workloads
- Problems with these techniques:

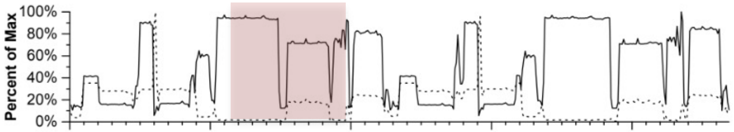
Techniques to Simulate Faster

- Workload reduction
  - Simulating for reduced input sets
  - Simulating for reduced loop counts in workloads
- Problems with these techniques:
  - [Partial simulation + extrapolation] → fail to capture global variations in program behavior and performance.



Techniques to Simulate Faster

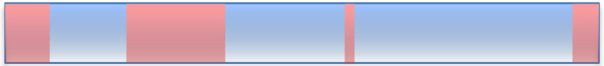
- Workload reduction
  - Simulating for reduced input sets
  - Simulating for reduced loop counts in workloads
- Problems with these techniques:
  - [Partial simulation + extrapolation] → fail to capture global variations in program behavior and performance.

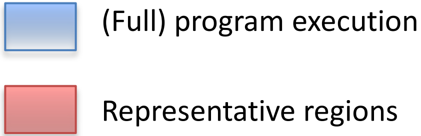


  - [Workload reduction] → benchmark behavior varies significantly across several inputs → do not reflect the actual performance.

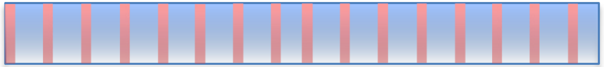
Sampled Simulation to the Rescue!

- Sampling enables the simulation of selective representative regions
  - *Representative regions*: subset of regions in the application that reflect the behavior of the entire system when extrapolated
- How to select these "representative regions"?
  - Targeted sampling (like in SimPoint)



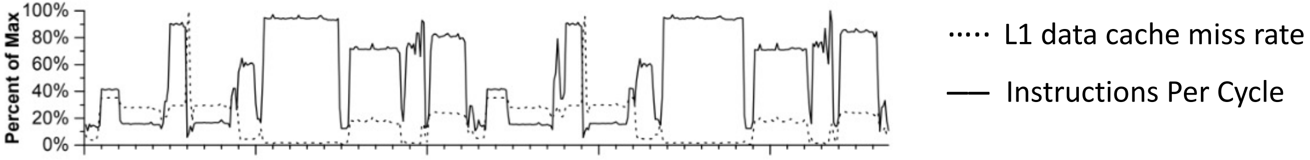


  - Statistical sampling (like in SMARTS)



Sampled Simulation Techniques: SimPoint

- Large-scale program behaviors vary significantly over their run times.
  - Difficult to estimate performance using previously discussed techniques.



- Main idea behind SimPoint:
  - Automatically & efficiently analyzing program behavior over different phases of execution.
- SimPoint uses Basic Block Vectors (BBV) as a hardware-independent metric for characterizing the program behavior in different phases.

Sampled Simulation Techniques: SimPoint

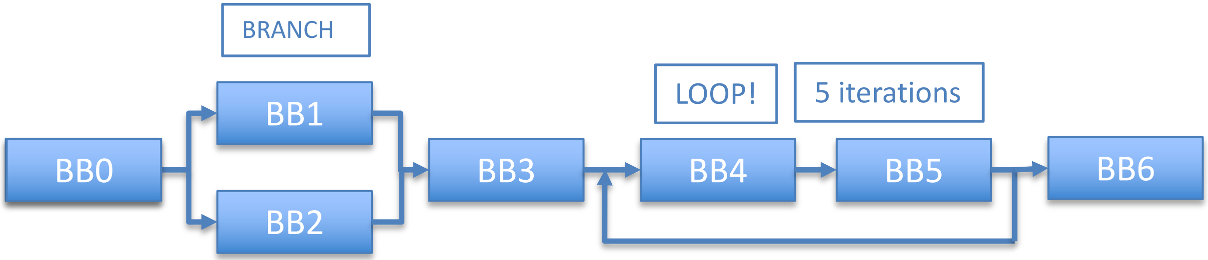
- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

Sampled Simulation Techniques: SimPoint

A Basic Block Vector (BBV) is a single-dimensional array that maintains a count of how many times each basic block was executed in each interval



Basic Block: A section of code that has a single point of entry and a single point of exit.

	0	1	2	3	4	5	6
Basic Block Vector:	1	1	0	1	5	5	1

← Indexed by Basic Block IDs

← Maintains the execution count for each Basic Block

Sampled Simulation Techniques: SimPoint

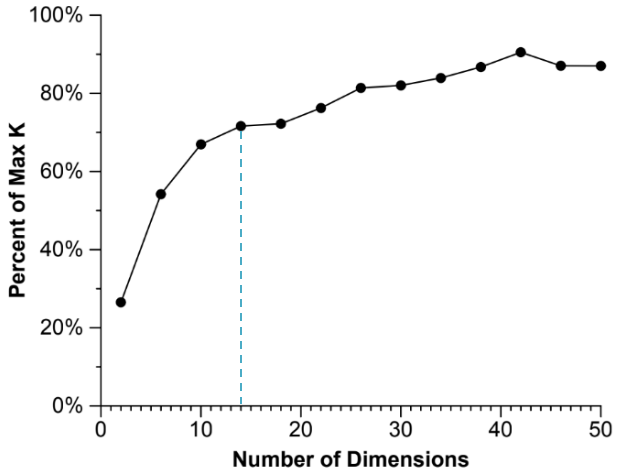
- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

Sampled Simulation Techniques: SimPoint

- The Basic Block Vectors obtained from the basic block profiling step have a very large number of dimensions! (in the range of 2,000 -- 100,000)
- “Curse of dimensionality”:
  - Hard to cluster data as the number of dimensions increases.
  - Clustering time increases significantly wrt as the number of dimensions increases.
- Solution: Reduce the number of dimensions to 15 using Random Linear Projections.



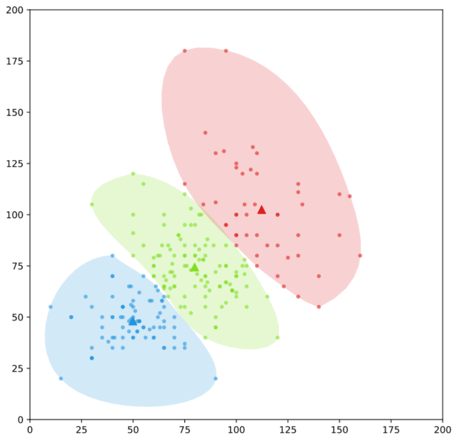
Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

Sampled Simulation Techniques: SimPoint

K-means clustering:

- Initialize k cluster centers by randomly choosing k points from the data.
- Repeat until convergence:
  - Do for all data points:
    - Compare the distance from all k cluster centers.
    - Assign it to the cluster with the closest center.
  - Update cluster center to the centroid of the newly assigned memberships.



Choosing k: The clustering that achieves a BIC[1] score that is at least 90% of the spread between the largest and smallest BIC score is chosen.

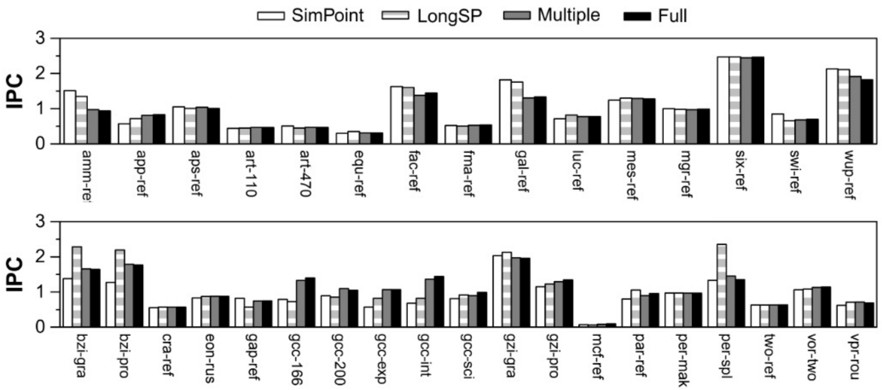
[1]Bayesian Information Criterion

Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

Sampled Simulation Techniques: SimPoint

- Representative region → single simulation point
  - BBV with the lowest distance from the centroid of all cluster centers.
- Representative regions → multiple simulation points
  - For each cluster, choose the BBV that is closest to the centroid of the cluster.



Sampled Simulation Techniques: SMARTS

- Main idea behind SMARTS:
  - Using systematic sampling:
    - To identify a minimal but representative sample from the population for microarchitecture simulation
    - To establish a confidence level for the error on sample estimates
  - Simulating using two modes :
    - Detailed simulation of sampled instructions → accounting for all the microarchitectural details.
    - Functional simulation of remaining instructions → accounting only for the programmer-visible architectural states (ex: registers, memory).

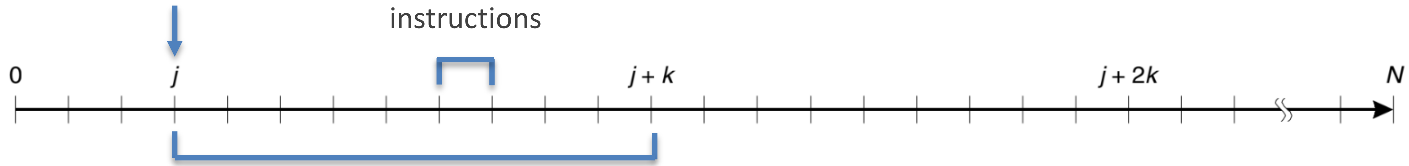
Sampled Simulation Techniques: SMARTS

- SMARTS uses Systematic Sampling:

Total sample size:
$n = N/k$ units
OR
$n \times U = n \times N/k$ instructions

Start sampling
at offset j

Each unit
consists of U
instructions

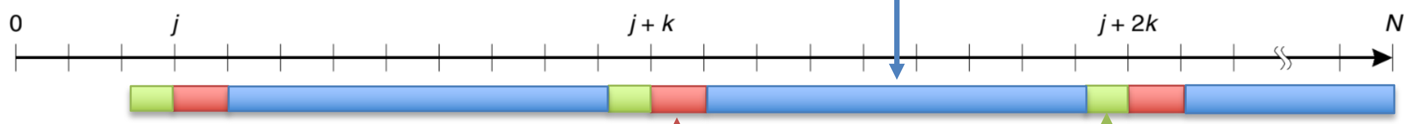


Sample at a fixed interval
of length k units or $k \times U$
instructions, where
$k = N / n$

Sampled Simulation Techniques: SMARTS

- Simulation:

U(k -1) – W instructions are functionally simulated and large structures may be warmed



U instructions are measured as a sampling unit using detailed simulation

W instructions of detailed simulation warm state before each sampling unit

Values of U and W depend on workload characteristics

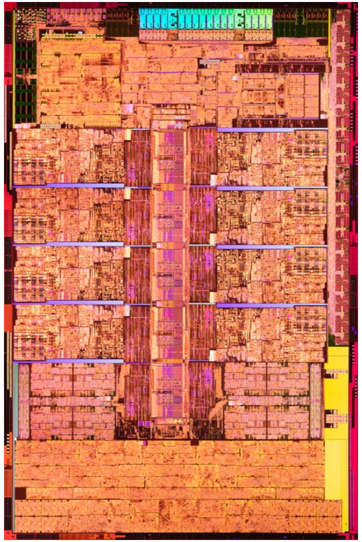
Sampled Simulation Techniques: SMARTS

- Evaluation results:
  - Average error:
    - 0.64% for CPI
    - 0.59% for EPI

    By simulating fewer than 50 million instructions in detail per benchmark.
  - Speedup over full-stream simulation:
    - 35x for 8-way out-of-order processors
    - 60x for 16-way out-of-order processors

Simulation in the Post-Dennard Era

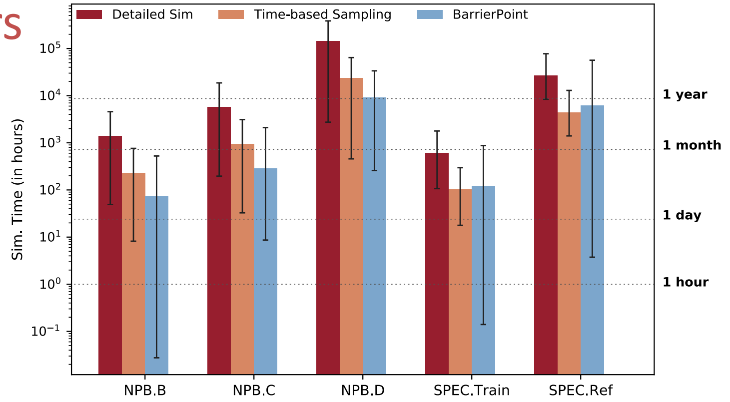
- Modern architectures require smarter simulators
- Microarchitectural simulation is slow
  - NPB (D), SPEC CPU2017 (ref) can take years
  - Solution – Simulate representative sample



Intel's Alder Lake die shot.
Image source: WikiChip

Simulation in the Post-Dennard Era

- Modern architectures require smarter simulators
- Microarchitectural simulation is slow
  - NPB (D), SPEC CPU2017 (ref) can take years
  - Solution – Simulate representative sample

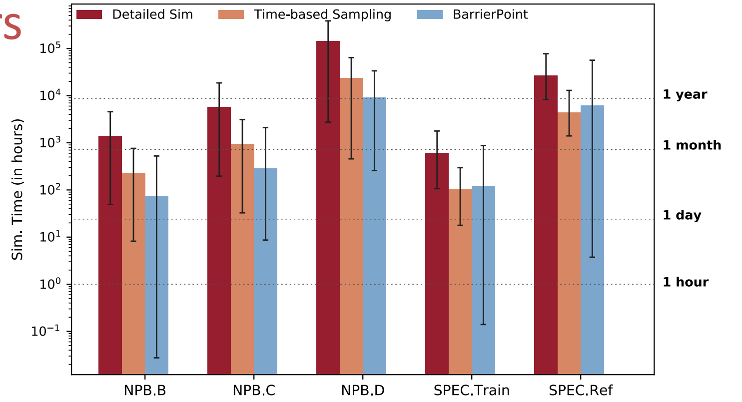


Benchmarks with *8* threads, *static* schedule, *passive* wait-policy, simulated at *100 KIPS*.

Simulation in the Post-Dennard Era

- Modern architectures require smarter simulators
- Microarchitectural simulation is slow
  - NPB (D), SPEC CPU2017 (ref) can take years
  - Solution – Simulate representative sample

? Can we further bring down simulation time



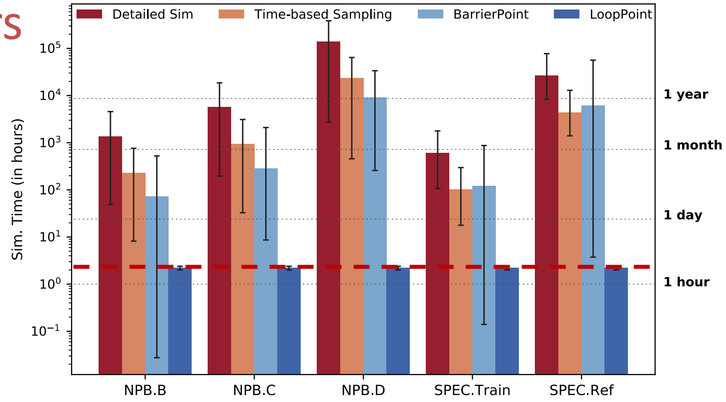
Benchmarks with *8* threads, *static* schedule, *passive* wait-policy, simulated at *100 KIPS*.

NUS National University of Singapore
intel

Simulation in the Post-Dennard Era

- Modern architectures require smarter simulators
- Microarchitectural simulation is slow
  - NPB (D), SPEC CPU2017 (ref) can take years
  - Solution – Simulate representative sample

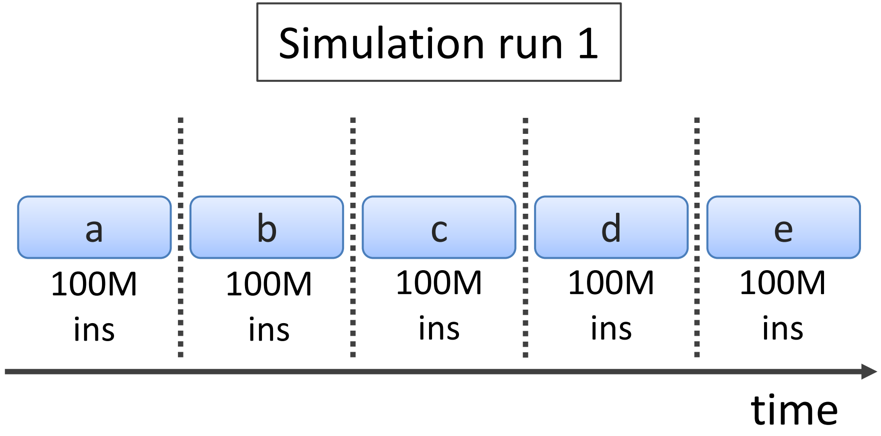
? Can we further bring down simulation time



Benchmarks with *8* threads, *static* schedule, *passive* wait-policy, simulated at *100 KIPS*.

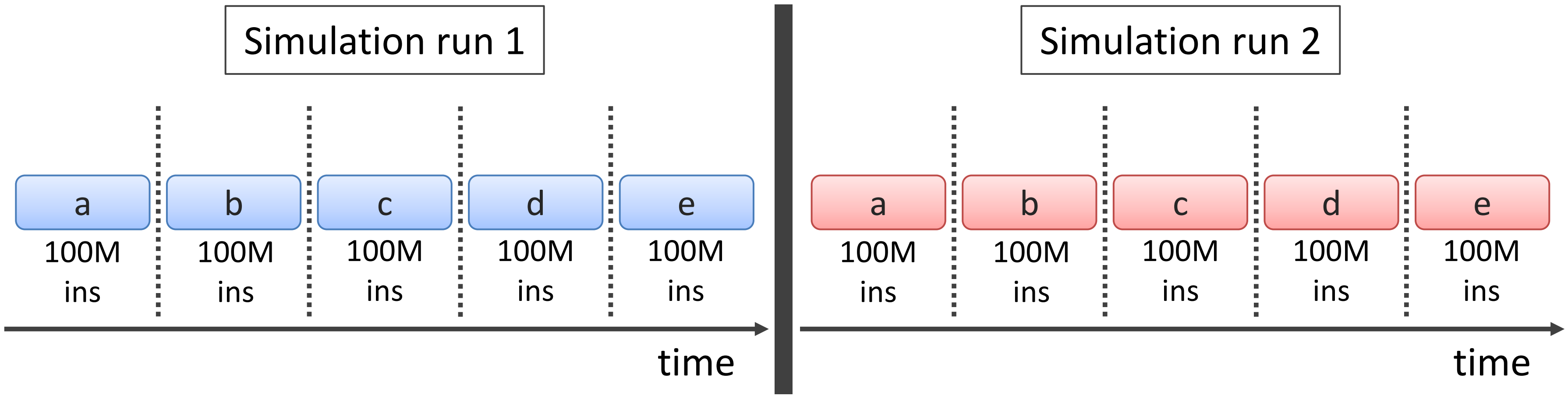
Extending Single-threaded Techniques

- SimPoint or SMARTS ➢ Instruction count-based techniques
  - Works well for single-threaded applications



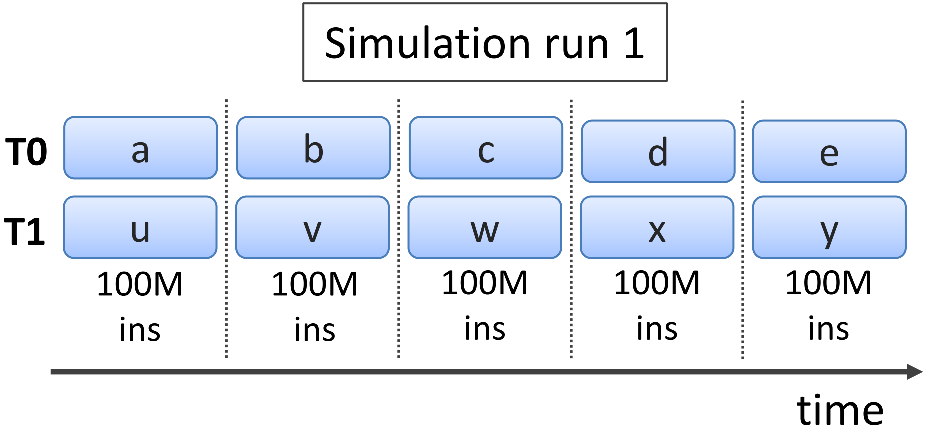
Extending Single-threaded Techniques

- SimPoint or SMARTS ➤ Instruction count-based techniques
  - Works well for single-threaded applications



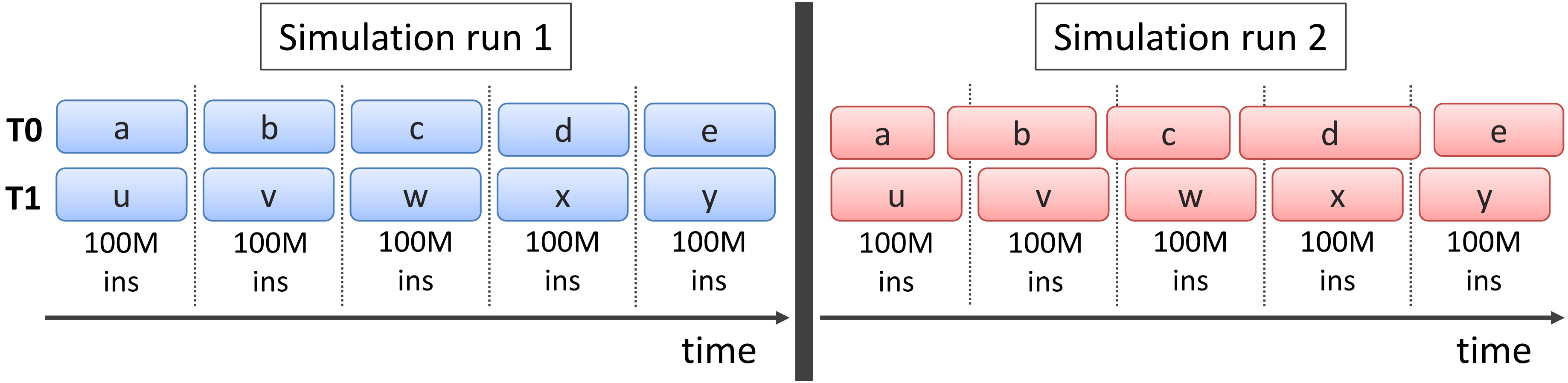
Extending Single-threaded Techniques

- SimPoint or SMARTS ➢ Instruction count-based techniques
  - Inconsistent regions for multi-threaded applications



Extending Single-threaded Techniques

- SimPoint or SMARTS ➢ Instruction count-based techniques
  - Inconsistent regions for multi-threaded applications



Multi-threaded Sampling is Complex

Instruction count-based techniques are unsuitable[1]

Threads progress differently due to load imbalance

Representing parallelism among threads

Differentiating thread waiting from real work

[1]Alameldeen et al., “IPC Considered Harmful for Multiprocessor Workloads”, IEEE Micro 2006

Multi-threaded Sampling is Complex

Instruction count-based techniques are unsuitable[1]

Threads progress differently due to load imbalance

Identify a *unit of work* that is *invariant* across executions

Representing parallelism among threads

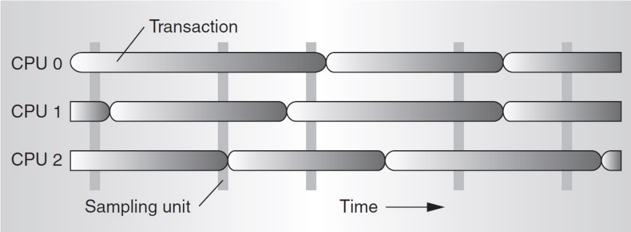
Differentiating thread waiting from real work

[1]Alameldeen et al., "IPC Considered Harmful for Multiprocessor Workloads", IEEE Micro 2006

Multi-threaded Sampling: Prior works

FlexPoints

- Designed for non-synchronizing throughput workloads
- Instruction count-based sampling
- Assumes no thread interaction
- Requires simulation of the full application

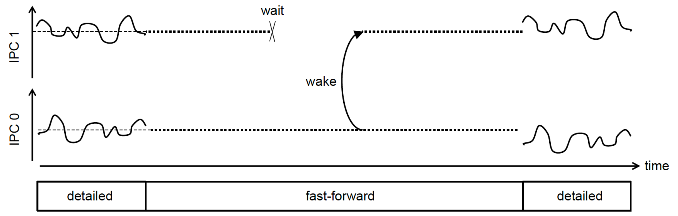


Wenisch et al., "SimFlex: statistical sampling of computer system simulation", IEEE Micro'06

Multi-threaded Sampling: Prior works

Time-based Sampling

- Designed for synchronizing generic multi-threaded workloads
- Applies to generic multi-threaded workloads
- Extremely slow
- Requires simulation of the full application

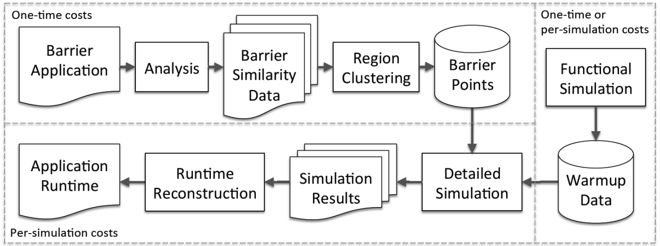


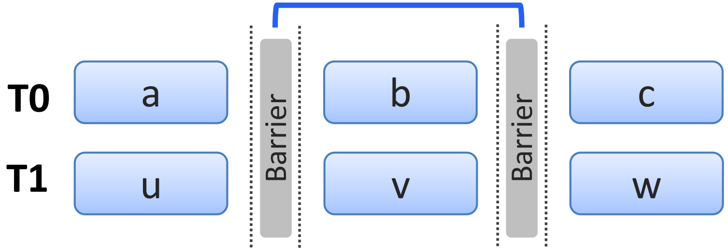
Carlson et al., “Sampled Simulation of Multithreaded Applications”, ISPASS’13
Ardestani et al., "ESESC: A fast multicore simulator using time-based sampling." HPCA, 2013

Multi-threaded Sampling: Prior works

BarrierPoint

- Designed for barrier-synchronized multi-threaded workloads
- Scales well with number of barriers
- Slow when *inter-barrier regions* are large





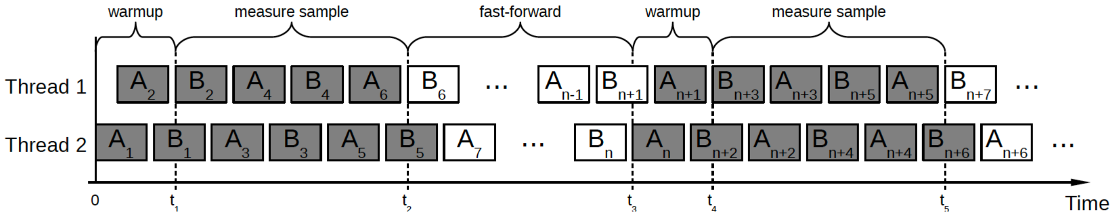
Carlson et al., "BarrierPoint: Sampled simulation of multi-threaded applications", ISPASS'14

Multi-threaded Sampling: Prior works

TaskPoint

- Designed for task-based workloads
- Uses analytical models to improve accuracy
- Works only for the particular workload type

```
#pragma omp task
    label(task type 1)
do_something();
```



Grass et al., “TaskPoint: Sampled simulation of task-based programs”, ISPASS’16

The Unit of Work

Single-threaded Sampling

SimPoint[1]
SMARTS[2] → Instruction count

Multiprocessor Sampling

Flex Points[3] → Instruction count

Multi-threaded Sampling

Time-based sampling[4] → Time

Application-specific Sampling

BarrierPoint[5] → Inter-barrier regions
TaskPoint[6] → Task instances

[1]Sherwood et al., "Automatically Characterizing Large Scale Program Behavior", ASPLOS'02
[2]Wunderlich et al., "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling", ISCA'03
[3]Wenisch et al., "SimFlex: statistical sampling of computer system simulation", IEEE Micro'06
[4]Carlson et al., "Sampled Simulation of Multithreaded Applications", ISPASS'13
[5]Carlson et al., "BarrierPoint: Sampled simulation of multi-threaded applications", ISPASS'14
[6]Grass et al., "TaskPoint: Sampled simulation of task-based programs", ISPASS'16

The Unit of Work

Single-threaded Sampling		
SimPoint[1] SMARTS[2]	→	Instruction count
Time-based sampling[4]	→	Time

Multiprocessor Sampling		
Flex Points[3]	→	Instruction count
BarrierPoint[5]	→	Inter-barrier regions
TaskPoint[6]	→	Task instances

We consider generic loop iterations as the unit of work

[1]Sherwood et al., "Automatically Characterizing Large Scale Program Behavior", ASPLOS'02
[2]Wunderlich et al., "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling", ISCA'03
[3]Wenisch et al., "SimFlex: statistical sampling of computer system simulation", IEEE Micro'06

[4]Carlson et al., "Sampled Simulation of Multithreaded Applications", ISPASS'13
[5]Carlson et al., "BarrierPoint: Sampled simulation of multi-threaded applications", ISPASS'14
[6]Grass et al., "TaskPoint: Sampled simulation of task-based programs", ISPASS'16

Overall Methodology

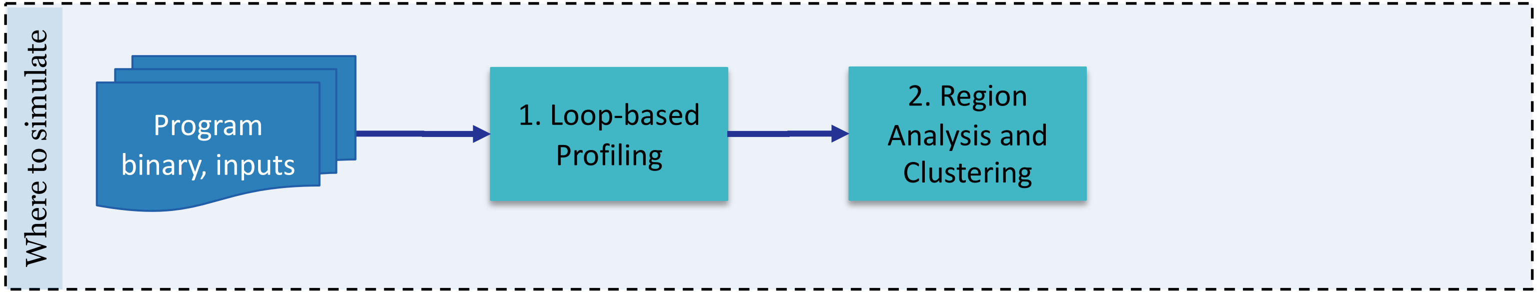
Where to simulate

How to simulate

Overall Methodology

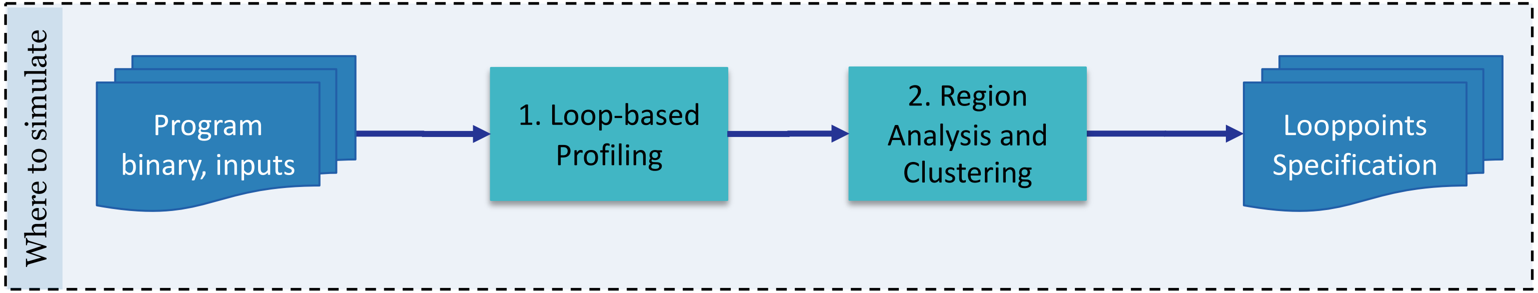
How to simulate

Overall Methodology



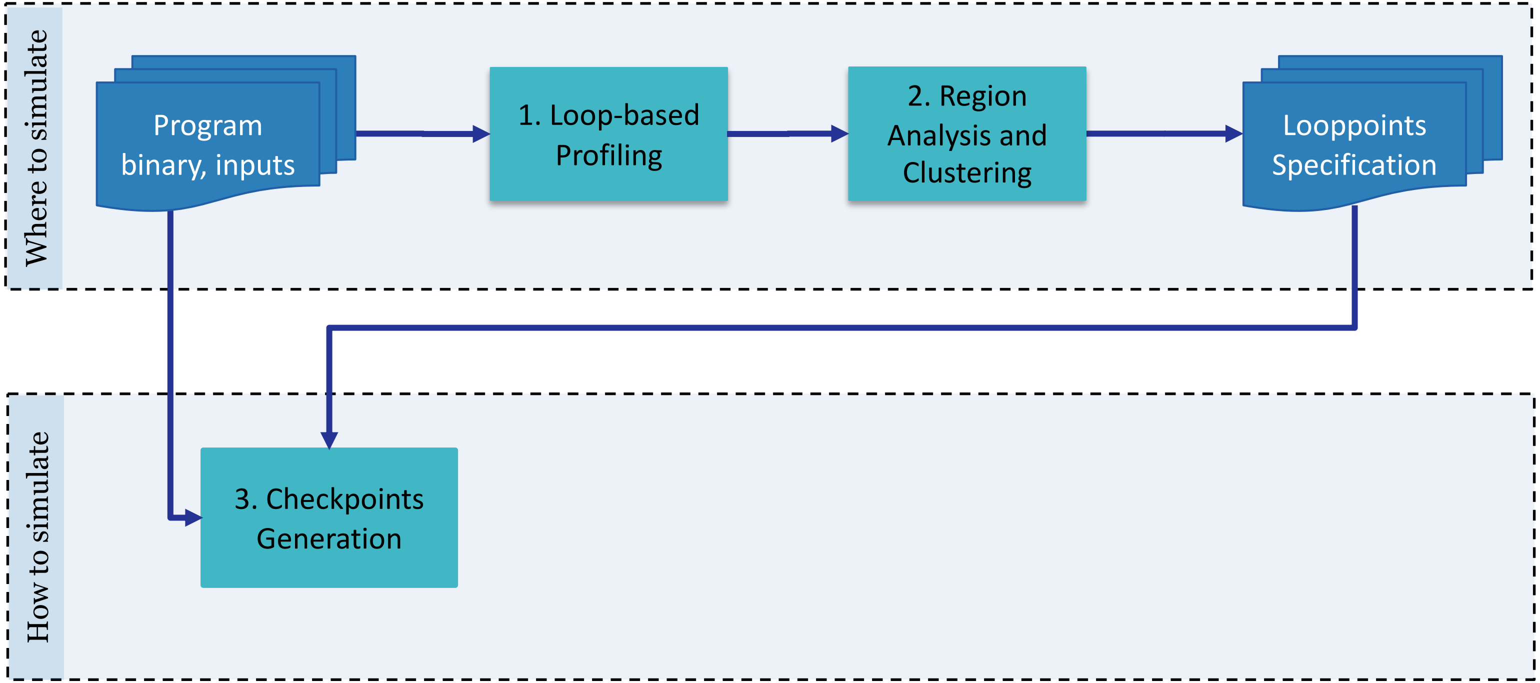
How to simulate

Overall Methodology



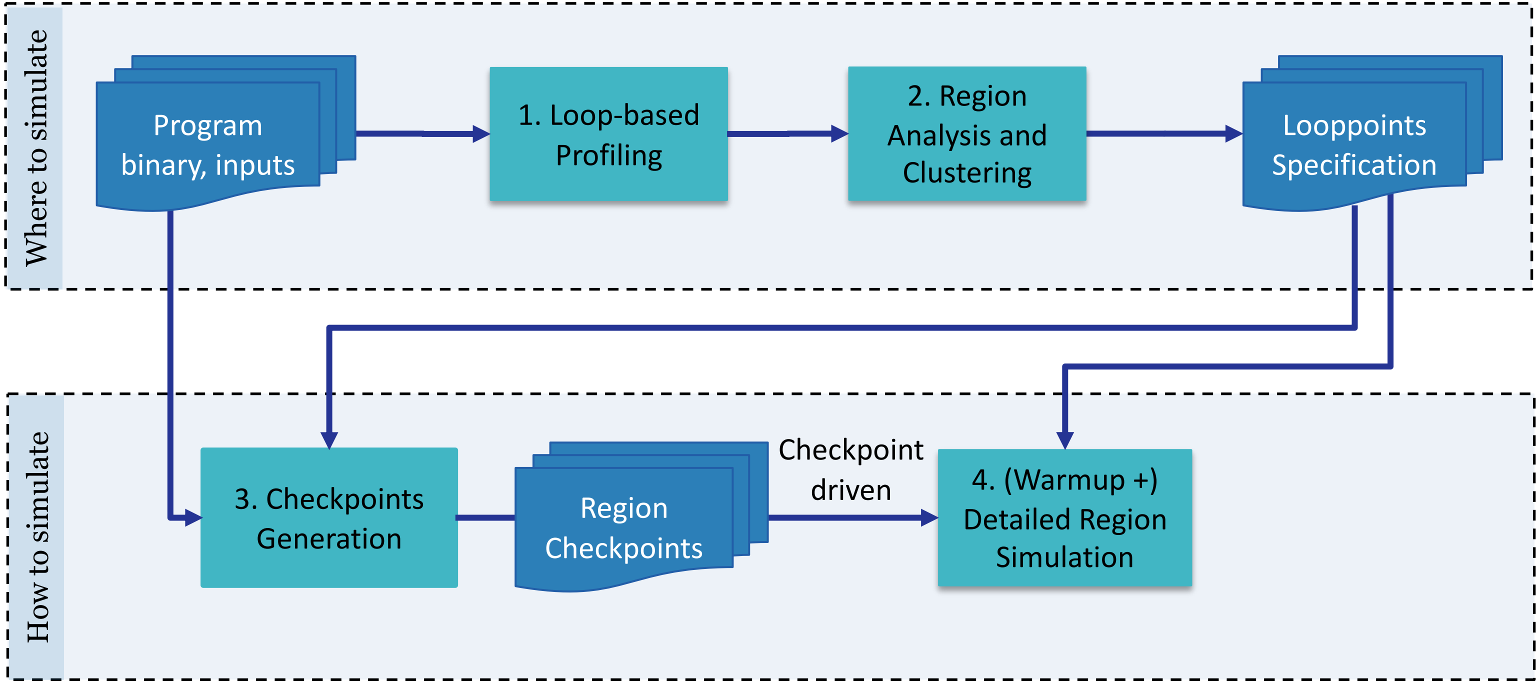
How to simulate

Overall Methodology

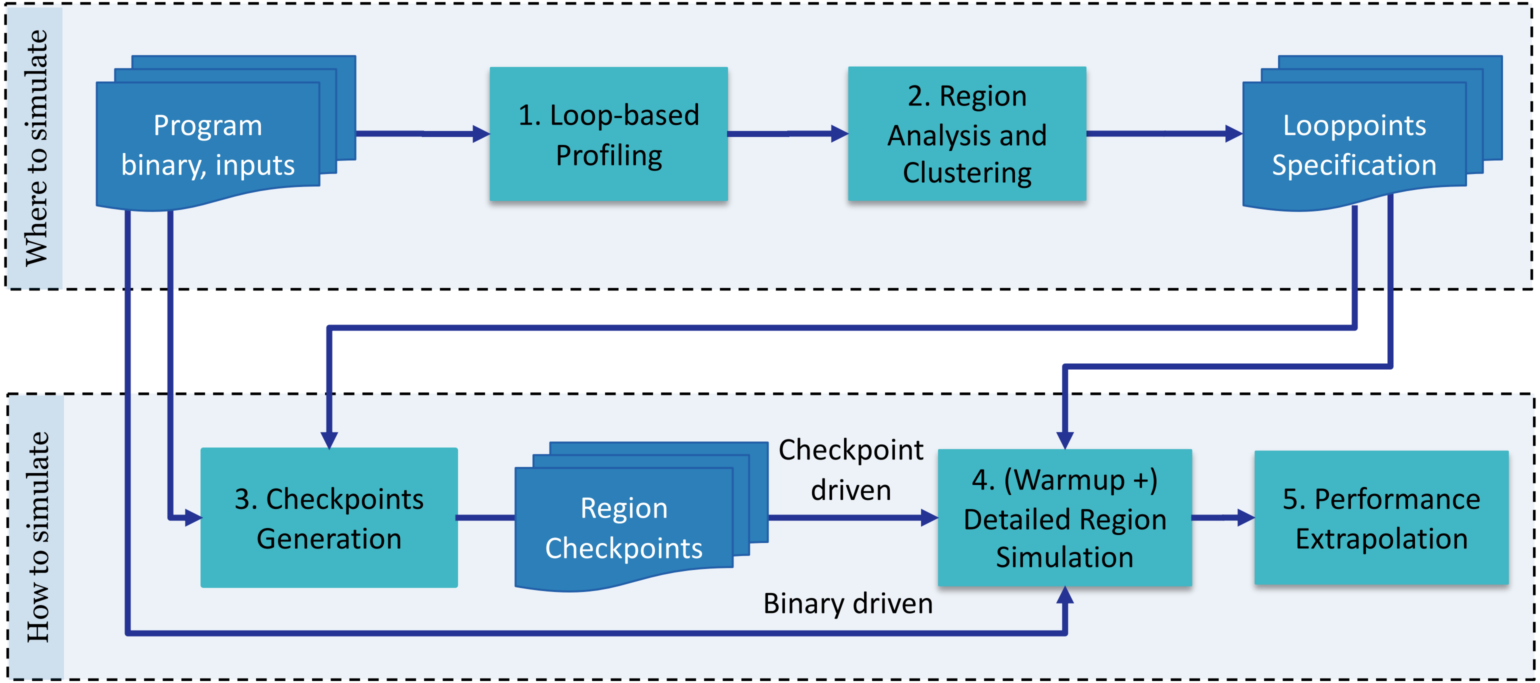


NUS National University of Singapore intel

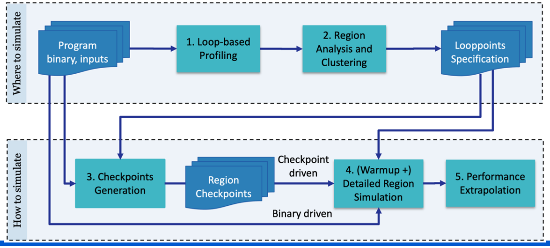
Overall Methodology



Overall Methodology

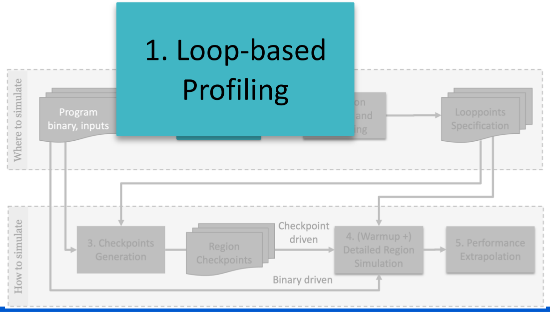


Loop-based Profiling

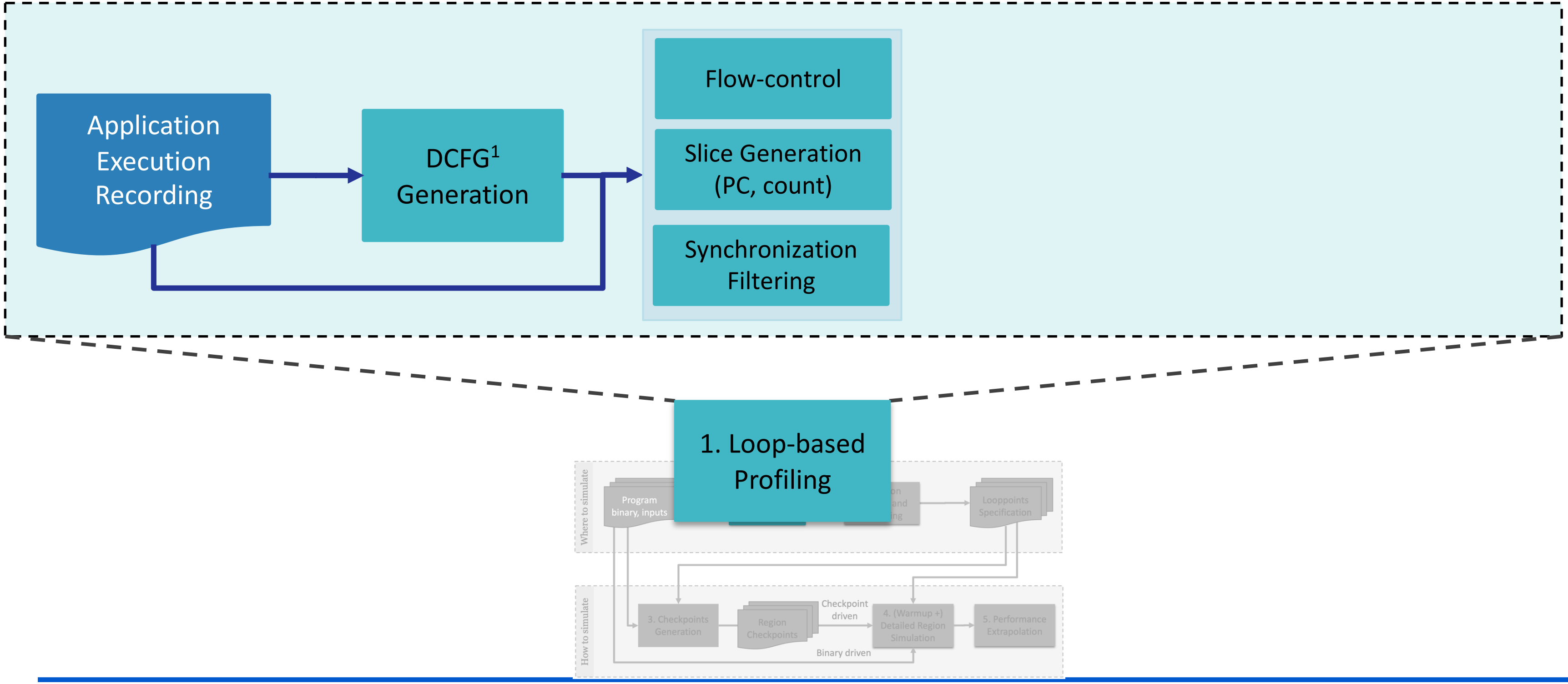


DCFG: Dynamic Control-Flow Graph

Loop-based Profiling

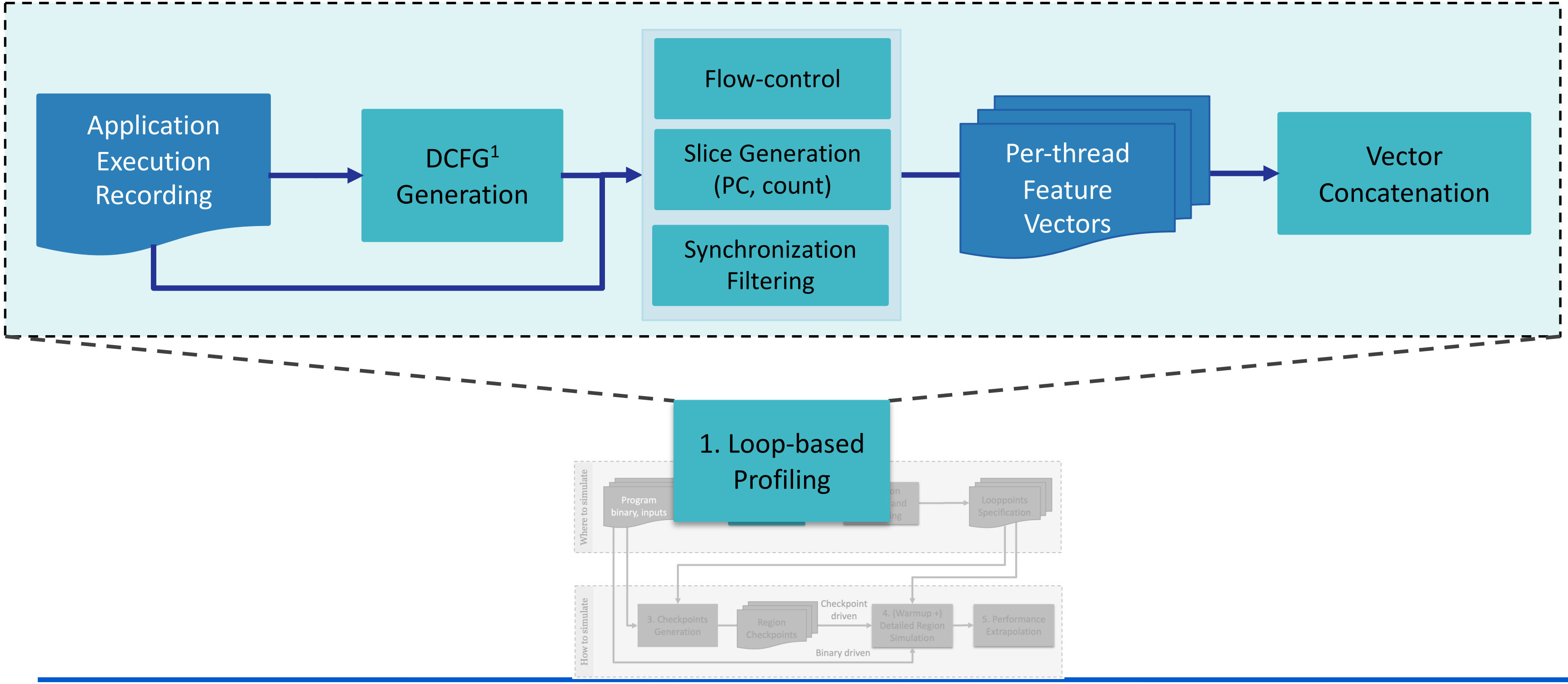


Loop-based Profiling



DCFG: Dynamic Control-Flow Graph

Loop-based Profiling



[1] DCFG: Dynamic Control-Flow Graph

Loop-based Profiling

Flow-control

Slice Generation (PC, count)

Synchronization Filtering

Loop-based Profiling: Flow-control

- Load Imbalance can affect profiling
  - Make sure threads make equal forward progress
- Implementation: Control the forward progress of threads
  - Synchronize threads (barriers) externally at regular intervals
  - Make sure all threads execute similar number of instructions

Flow-control

Slice Generation (PC, count)

Synchronization Filtering

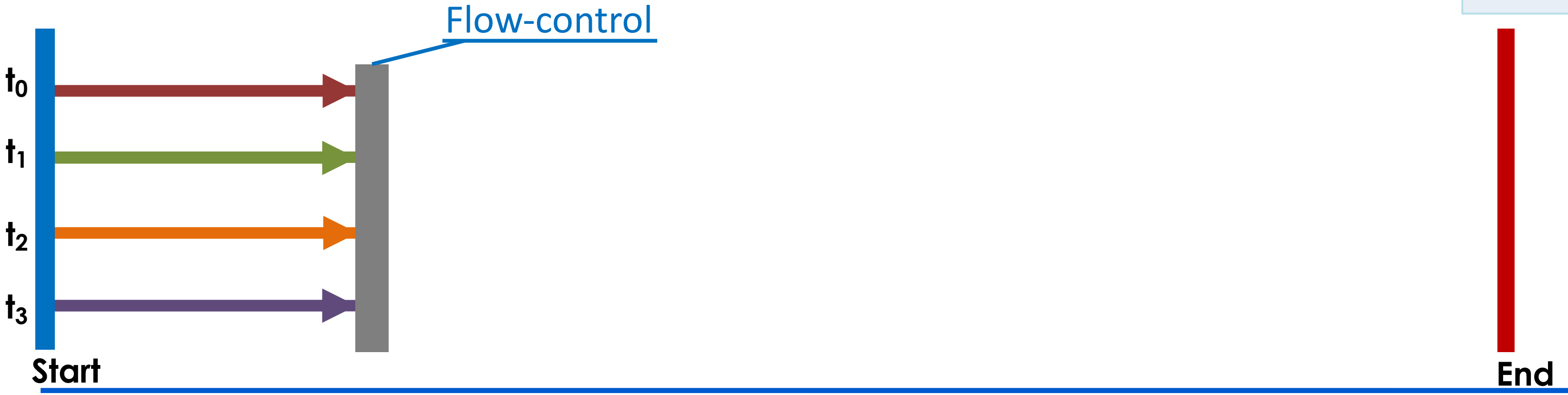
Loop-based Profiling: Flow-control

- Load Imbalance can affect profiling
  - Make sure threads make equal forward progress
- Implementation: Control the forward progress of threads
  - Synchronize threads (barriers) externally at regular intervals
  - Make sure all threads execute similar number of instructions

Flow-control

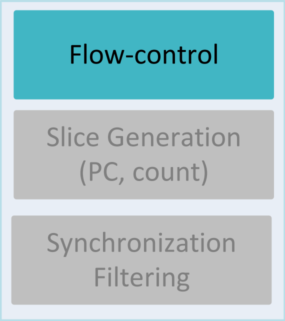
Slice Generation (PC, count)

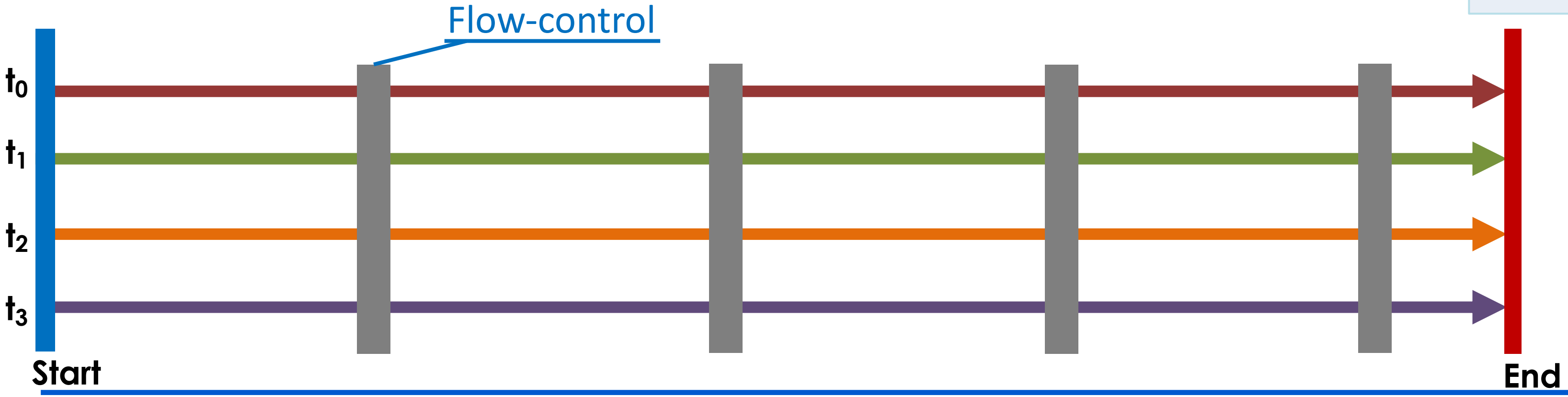
Synchronization Filtering



Loop-based Profiling: Flow-control

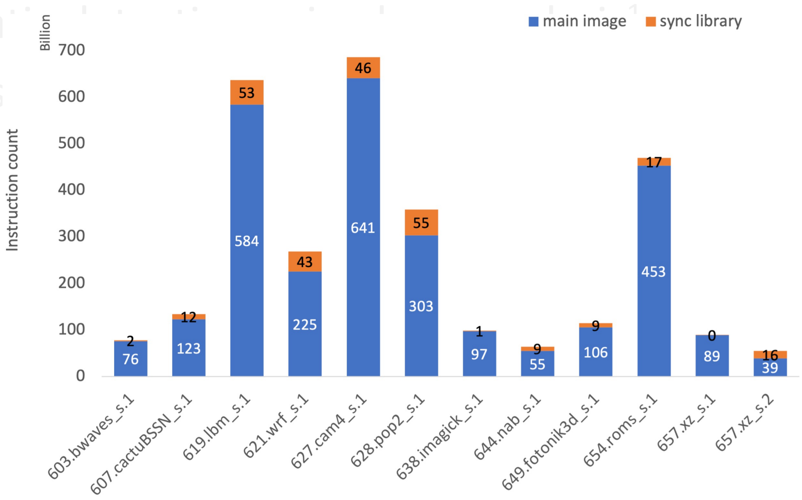
- Load Imbalance can affect profiling
  - Make sure threads make equal forward progress
- Implementation: Control the forward progress of threads
  - Synchronize threads (barriers) externally at regular intervals
  - Make sure all threads execute similar number of instructions





Loop-based Profiling: Sync Filtering

- Goal: Filter out synchronization during profiling
  - Profiling data should contain only ***real*** work
- Solutions



Flow-control

Slice Generation (PC, count)

Synchronization Filtering

Loop-based Profiling: Sync Filtering

- Goal: Filter out synchronization during profiling
  - Profiling data should contain only ***real*** work
- Solutions
  - Automatic detection using loop analysis[1]
  - Ignore sync library code (Ex. `libiomp5.so`, `libpthread.so`)

Flow-control

Slice Generation (PC, count)

Synchronization Filtering

[1]Li et al., "Spin detection hardware for improved management of multithreaded systems," TPDS, 2006

NUS National University of Singapore intel

Loop-based Profiling: Sync Filtering

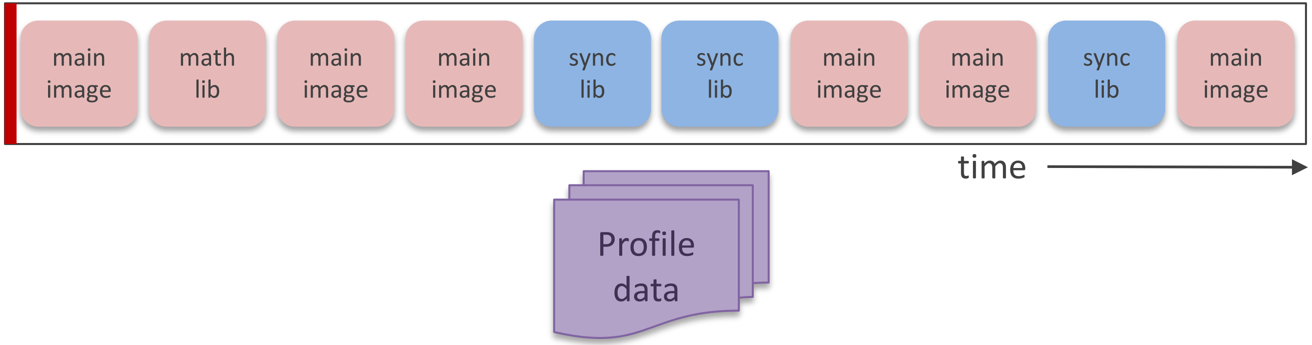
Ignore sync library code (Ex. `libiomp5.so`, `libpthread.so`)

Flow-control

Slice Generation (PC, count)

Synchronization Filtering

Application execution



[1]Li et al., "Spin detection hardware for improved management of multithreaded systems," TPDS, 2006

Loop-based Profiling: Sync Filtering

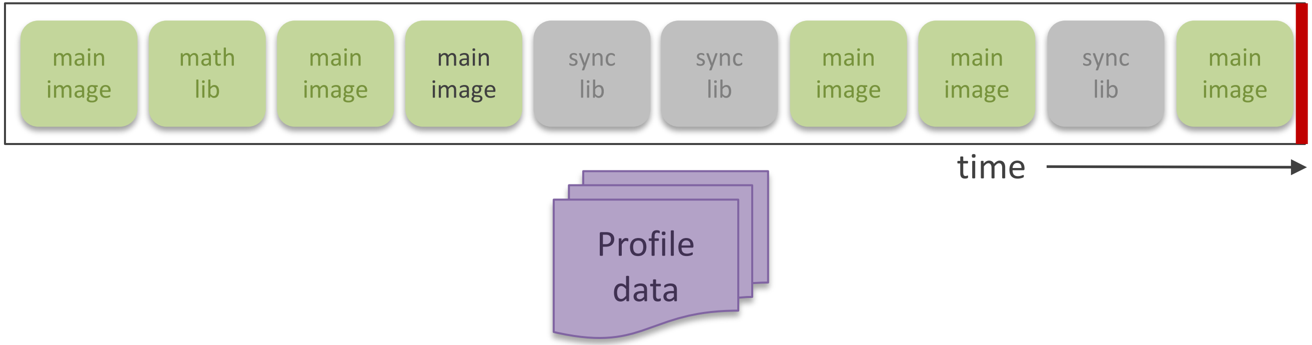
Ignore sync library code (Ex. `libiomp5.so`, `libpthread.so`)

Flow-control

Slice Generation (PC, count)

Synchronization Filtering

Application execution



[1]Li et al., "Spin detection hardware for improved management of multithreaded systems," TPDS, 2006

Loop-based Profiling: Slice Generation

- Region start/stop
  - Global instruction count reaches threshold (*#threads* × 100 M)
  - Region boundary at a loop entry/exit – use DCFG analysis
- Looppoint region markers (*PC, count*$_{PC}$)
  - Global count of loop entries: invariant across executions
  - Simulate the same ***amount of work***

Flow-control

Slice Generation (PC, count)

Synchronization Filtering

Loop-based Profiling: Slice Generation

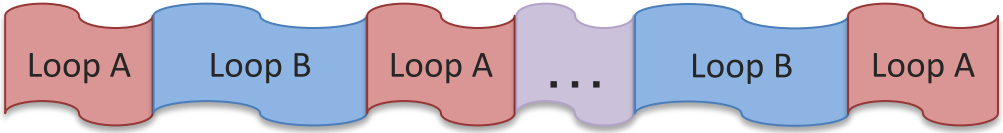
- Region start/stop
  - Global instruction count reaches threshold (*#threads* × 100 M)
  - Region boundary at a loop entry/exit – use DCFG analysis
- Looppoint region markers ***(PC, count$_{PC}$)***
  - Global count of loop entries: invariant across executions
  - Simulate the same ***amount of work***

Flow-control

Slice Generation (PC, count)

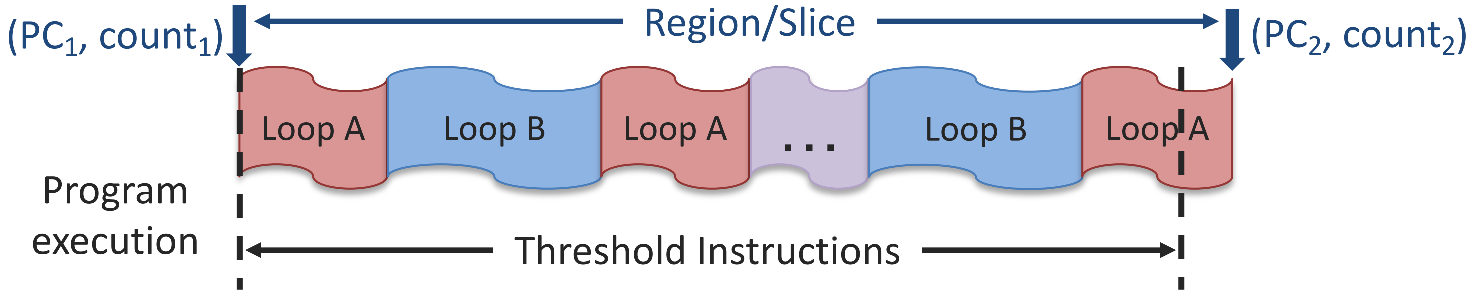
Synchronization Filtering

Program execution



Loop-based Profiling: Slice Generation

- Region start/stop
  - Global instruction count reaches threshold (*#threads* × 100 M)
  - Region boundary at a loop entry/exit – use DCFG analysis
- Looppoint region markers ***(PC, count$_{PC}$)***
  - Global count of loop entries: invariant across executions
  - Simulate the same ***amount of work***



Flow-control

Slice Generation (PC, count)

Synchronization Filtering

Loop-based Profiling: Slice Generation

- Basic Block (BB)
  - Section of code with single entry and exit
- Basic Block Vector (BBV)
  - Execution fingerprint of an application interval
  - Vector with one element for each basic block
  - Exec Wt = *entry count × number of instructions*

BB	Example Assembly Code

A	srl a2, 0x8, t4
	and a2, 0xff, t12
	addl zero, t12, s6
	subl t7, 0x1, t7
	cmpeq s6, 0x25, v0
	cmpeq s6, 0, t0
	bis v0, t0, v0
	bne v0, 0x120018c48

B	subl t7, 0x1, t7
	cmple t7, 0x3, t2
	beq t2, 0x120018b04

C	ble t7, 0x120018bb4
…	…

ID: A B C

Image source: Sherwood et.al., “Automatically Characterizing Large Scale Program Behavior”, ASPLOS 2002

Loop-based Profiling: Slice Generation

- Basic Block (BB)
  - Section of code with single entry and exit
- Basic Block Vector (BBV)
  - Execution fingerprint of an application interval
  - Vector with one element for each basic block
  - Exec Wt = *entry count × number of instructions*

BB	Example Assembly Code	
A	srl	a2, 0x8, t4
	and	a2, 0xff, t12
	addl	zero, t12, s6
	subl	t7, 0x1, t7
	cmpeq	s6, 0x25, v0
	cmpeq	s6, 0, t0
	bis	v0, t0, v0
	bne	v0, 0x120018c48
B	subl	t7, 0x1, t7
	cmple	t7, 0x3, t2
	beq	t2, 0x120018b04
C	ble	t7, 0x120018bb4
…	…	

```
               ID:   A   B   C
    BB Exec Count: < 1, 20, 0, …>
weigh by Block Size: < 8,  3, 1, …>
```


Image source: Sherwood et.al., “Automatically Characterizing Large Scale Program Behavior”, ASPLOS 2002

Loop-based Profiling: Slice Generation

- Basic Block (BB)
  - Section of code with single entry and exit
- Basic Block Vector (BBV)
  - Execution fingerprint of an application interval
  - Vector with one element for each basic block
  - Exec Wt = *entry count* × *number of instructions*

BB	Example Assembly Code

A	srl a2, 0x8, t4
	and a2, 0xff, t12
	addl zero, t12, s6
	subl t7, 0x1, t7
	cmpeq s6, 0x25, v0
	cmpeq s6, 0, t0
	bis v0, t0, v0
	bne v0, 0x120018c48

B	subl t7, 0x1, t7
	cmple t7, 0x3, t2
	beq t2, 0x120018b04

C	ble t7, 0x120018bb4
…	…

```
                 ID:   A   B   C
      BB Exec Count: < 1, 20, 0, …>
weigh by Block Size: < 8,  3, 1, …>
        BB Exec Wt: < 8, 60, 0, …>
```


Image source: Sherwood et.al., “Automatically Characterizing Large Scale Program Behavior”, ASPLOS 2002

Loop-based Profiling: Slice Generation

- Basic Block (BB)
  - Section of code with single entry and exit
- Basic Block Vector (BBV)
  - Execu
  - Vecto
  - Exec Wt = *entry count × number of instructions*

```
[ A:8, B:60, C:0, …]
```

BBV

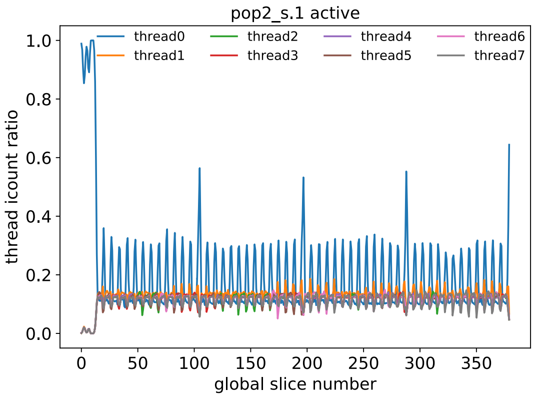
BB	Example Assembly Code	
A	srl	a2, 0x8, t4
	and	a2, 0xff, t12
	addl	zero, t12, s6
	subl	t7, 0x1, t7
	cmpeq	s6, 0x25, v0
	cmpeq	s6, 0, t0
	bis	v0, t0, v0
	bne	v0, 0x120018c48
B	subl	t7, 0x1, t7
	cmple	t7, 0x3, t2
	beq	t2, 0x120018b04
C	ble	t7, 0x120018bb4
…	…	

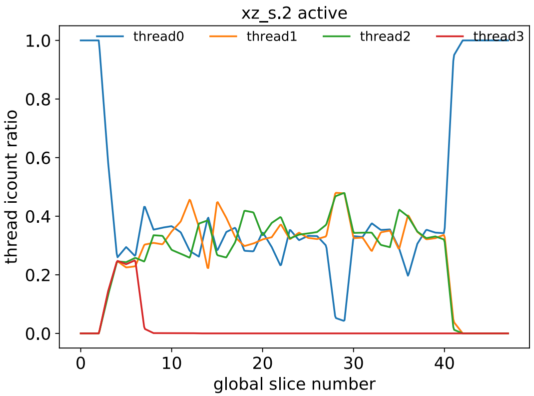
```
                ID:   A   B   C
     BB Exec Count: < 1, 20, 0, …>
weigh by Block Size: < 8,  3, 1, …>
       BB Exec Wt: < 8, 60, 0, …>
```

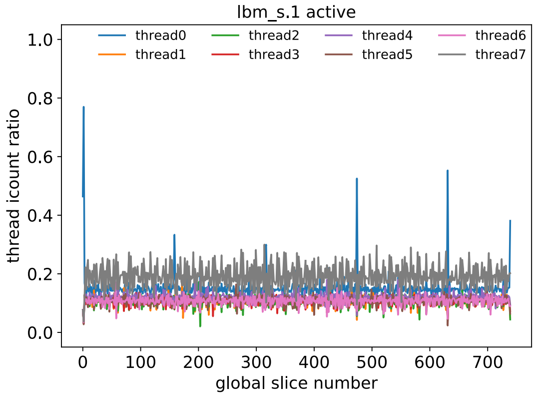

Image source: Sherwood et.al., “Automatically Characterizing Large Scale Program Behavior”, ASPLOS 2002

Loop-based Profiling: Vector Concatenation

- Ratio of instructions per thread may differ
- *Global-BBVs*: Concatenate per-thread BBVs to larger Global BBV







Loop-based Profiling: Vector Concatenation

- Ratio of instructions per thread may differ
- ***Global-BBV***s: Concatenate per-thread BBVs to larger Global BBV

BB	Example Assembly Code

A	srl a2, 0x8, t4
	and a2, 0xff, t12
	addl zero, t12, s6
	subl t7, 0x1, t7
	cmpeq s6, 0x25, v0
	cmpeq s6, 0, t0
	bis v0, t0, v0
	bne v0, 0x120018c48

B	subl t7, 0x1, t7
	cmple t7, 0x3, t2
	beq t2, 0x120018b04

C	ble t7, 0x120018bb4
…	…
M	subl t7, 0x1, t7 gt t7, 0x120018b90

BB	Example Assembly Code

A	srl a2, 0x8, t4
	and a2, 0xff, t12
	addl zero, t12, s6
	subl t7, 0x1, t7
	cmpeq s6, 0x25, v0
	cmpeq s6, 0, t0
	bis v0, t0, v0
	bne v0, 0x120018c48

B	subl t7, 0x1, t7
	cmple t7, 0x3, t2
	beq t2, 0x120018b04

C	ble t7, 0x120018bb4
…	…
M	subl t7, 0x1, t7 gt t7, 0x120018b90

Loop-based Profiling: Vector Concatenation

- Ratio of instructions per thread may differ
- *Global-BBVs*: Concatenate per-thre

Thread 1

Thread 0

BB	Example Assembly Code	
A	srl	a2, 0x8, t4
	and	a2, 0xff, t12
	addl	zero, t12, s6
	subl	t7, 0x1, t7
	cmpeq	s6, 0x25, v0
	cmpeq	s6, 0, t0
	bis	v0, t0, v0
	bne	v0, 0x120018c48
B	subl	t7, 0x1, t7
	cmple	t7, 0x3, t2
	beq	t2, 0x120018b04
C	ble	t7, 0x120018bb4
…	…	
M	subl	t7, 0x1, t7
	gt	t7, 0x120018b90

Loop-based Profiling: Vector Concatenation

- Ratio of instructions per thread may differ

BB ID: A B C …
BB Exec Wt: < 8, 60, 0, … >

BB ID: N O P …
BB Exec Wt: < 5, 90, 3, … >

Thread 1

BB	Example Assembly Code
N	
O	
P	
…	
Z	

Thread 0

BB	Example Assembly Code

A	srl a2, 0x8, t4
	and a2, 0xff, t12
	addl zero, t12, s6
	subl t7, 0x1, t7
	cmpeq s6, 0x25, v0
	cmpeq s6, 0, t0
	bis v0, t0, v0
	bne v0, 0x120018c48

B	subl t7, 0x1, t7
	cmple t7, 0x3, t2
	beq t2, 0x120018b04

C	ble t7, 0x120018bb4
…	…
M	subl t7, 0x1, t7 gt t7, 0x120018b90

Loop-based Profiling: Vector Concatenation

- Ratio of instructions per thread may differ
- *Global-BBVs*: Concatenate per-thre

BB ID:	A	B	C	…
BB Exec Wt:	< 8,	60,	0,	… >

[A:8, B:60, C:0, …, N:5, O:90, P:3, …]

Global-BBV

BB ID:	N	O	P	…
BB Exec Wt:	< 5,	90,	3,	… >

Thread 1

BB	Example Assembly Code
N	srl a2, 0x8, t4
P	
…	
Z	

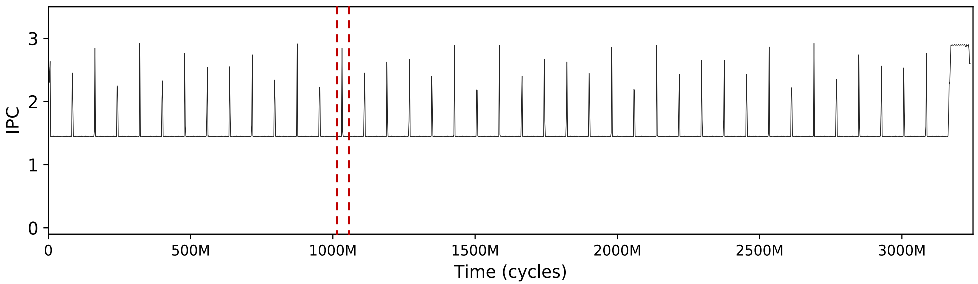
Thread 0

BB	Example Assembly Code	
A	srl	a2, 0x8, t4
	and	a2, 0xff, t12
	addl	zero, t12, s6
	subl	t7, 0x1, t7
		0x25, v0
		0, t0
		t0, v0
		0x120018c48
B	subl	t7, 0x1, t7
	cmple	t7, 0x3, t2
	beq	t2, 0x120018b04
C	ble	t7, 0x120018bb4
…	…	
M	subl	t7, 0x1, t7
	gt	t7, 0x120018b90

A LoopPoint Region

638.imagick_s/magick/morphology.c

```
#if defined(MAGICKCORE_OPENMP_SUPPORT)
  #pragma omp parallel for schedule(static,4) shared(progress,status) \
    magick_threads(image,result_image,image->rows,1)
#endif
  for (y=0; y < (ssize_t) image->rows; y++)
  {
     ……
     for (x=0; x < (ssize_t) image->columns; x++)
     {
        for (v=0; v < (ssize_t) kernel->height; v++) {
          for (u=0; u < (ssize_t) kernel->width; u++, k--) {
     ……
          } /* u */
     ……
        }  /* v */
     } /* x */
  } /* y */
       ……
```

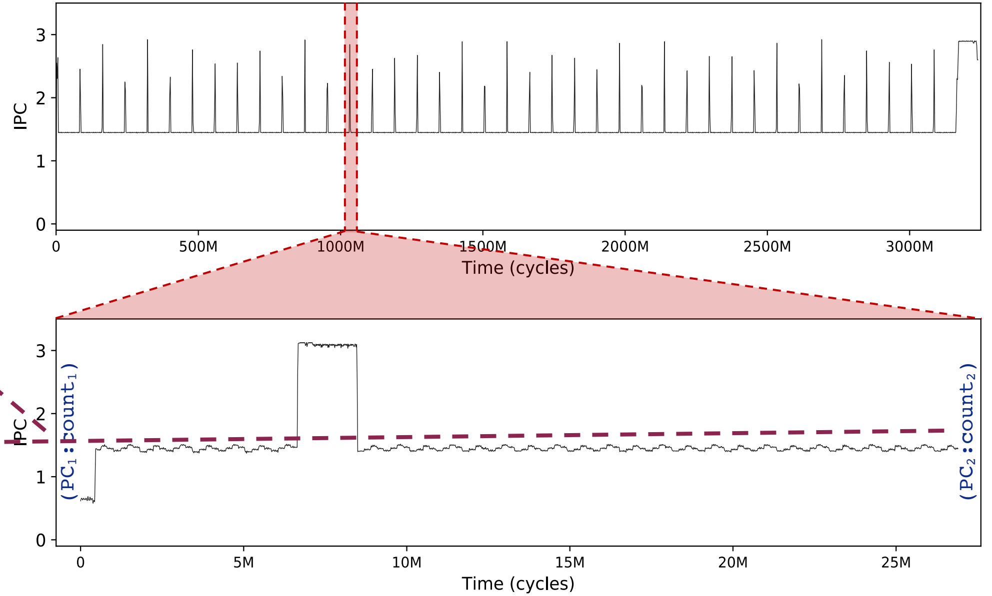


638.imagick_s, train input, 8 threads

A LoopPoint Region

638.imagick_s/magick/morphology.c

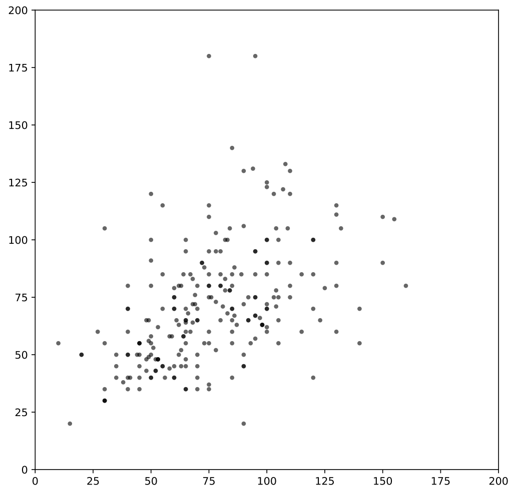
```
2842 #if defined(MAGICKCORE_OPENMP_SUPPORT)
2843   #pragma omp parallel for schedule(static,4) shared(progress,status) \
2844     magick_threads(image,result_image,image->rows,1)
2845 #endif
2846   for (y=0; y < (ssize_t) image->rows; y++)
2847   {
        ……
2886     for (x=0; x < (ssize_t) image->columns; x++)
2887     {
3021       for (v=0; v < (ssize_t) kernel->height; v++) {
3022         for (u=0; u < (ssize_t) kernel->width; u++, k--) {
        ……
3034         } /* u */
        ……
3037       }  /* v */
3342     } /* x */
3357   } /* y */
          ……
```



638.imagick_s, train input, 8 threads

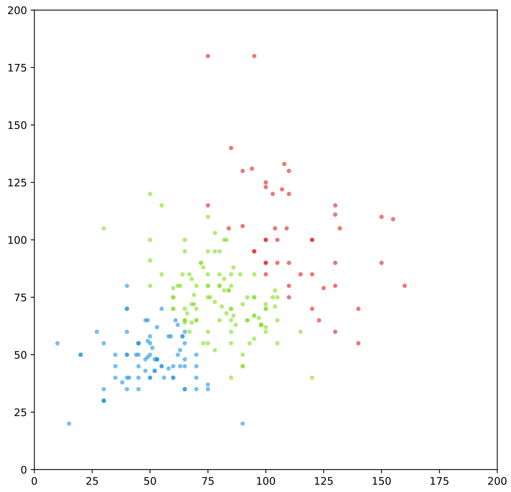
Identifying Simulation Regions

- Group similar Global-BBVs
  - K-means algorithm: Centroid-based clustering
- Vector closest to centroid is the representative
- Simulation regions (looppoints)
  - Checkpoints generated from the application
  - Use (PC, $count_{PC}$) information of representatives



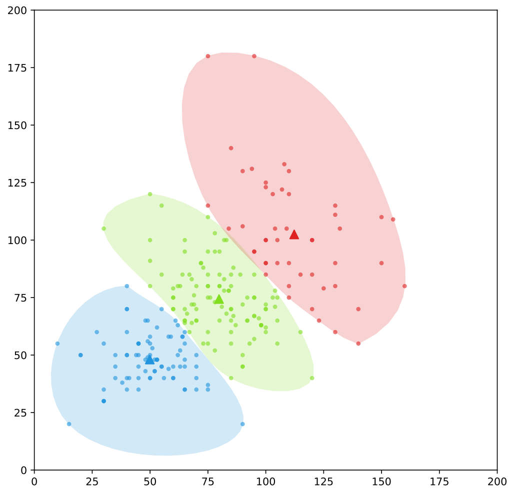
Identifying Simulation Regions

- Group similar Global-BBVs
  - K-means algorithm: Centroid-based clustering
- Vector closest to centroid is the representative
- Simulation regions (looppoints)
  - Checkpoints generated from the application
  - Use (PC, $count_{PC}$) information of representatives



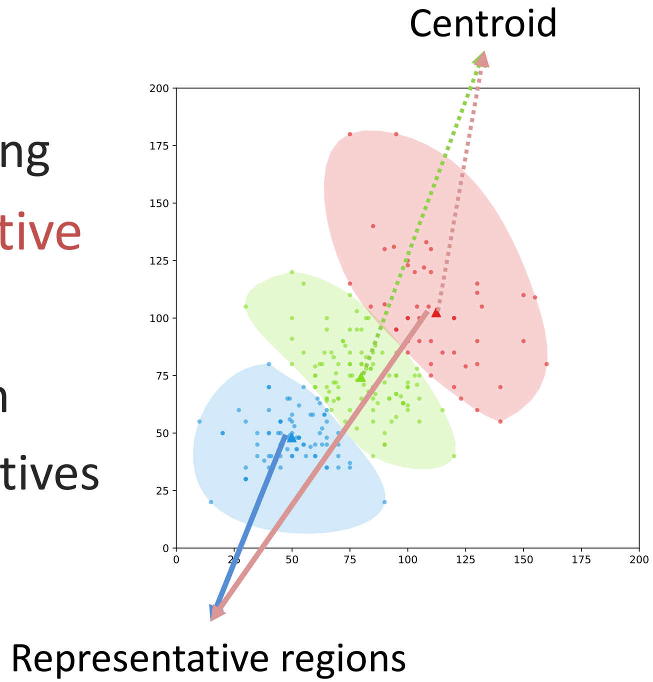
Identifying Simulation Regions

- Group similar Global-BBVs
  - K-means algorithm: Centroid-based clustering
- Vector closest to centroid is the representative
- Simulation regions (looppoints)
  - Checkpoints generated from the application
  - Use (PC, $count_{PC}$) information of representatives



Identifying Simulation Regions

- Group similar Global-BBVs
  - K-means algorithm: Centroid-based clustering
- Vector closest to centroid is the representative
- Simulation regions (looppoints)
  - Checkpoints generated from the application
  - Use (PC, $count_{PC}$) information of representatives



Application Reconstruction

- Representative regions (looppoints) are simulated in parallel
- Warmup handling
  - Simulate a large enough warmup region before simulation region
- Application performance
  - The weighted average of the performance of simulation regions

Application Reconstruction

- Representative regions (looppoints) are simulated in parallel
- Warmup handling
  - Simulate a large enough warmup region before simulation region
- Application performance
  - The weighted average of the performance of simulation regions

$$\boldsymbol{total\,runtime} = \sum_{i=rep_1}^{rep_N} \boldsymbol{runtime_i \times multiplier_i}$$

Application Reconstruction

- Representative regions (looppoints) are simulated in parallel
- Warmup handling
  - Simulate a large enough warmup region before simulation region
- Application performance
  - The weighted average of the perform

$$multiplier_j = \frac{\sum_{i=0}^{m} inscount_i}{inscount_j}$$

m regions represented by ***j***th looppoint

$$\boldsymbol{total\ runtime = \sum_{i=rep_1}^{rep_N} runtime_i \times multiplier_i}$$

Experimental Setup

- Simulation Infrastructure
  - Sniper[1] 7.4
    - Mimics Intel Gainestown 8/16 core
- Benchmarks and OpenMP settings
  - SPEC CPU2017 speed benchmarks
    - Input: train; Threads: 8; Wait policy: Active, Passive
  - NAS Parallel Benchmarks (NPB)
    - Input: Class C; Threads: 8, 16; Wait policy: Passive
  - OpenMP scheduling policy: *static*

[1]Carlson et.al., "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation", SC 2011

Image source: https://www.openmp.org/; https://www.spec.org/cpu2017/ ; https://www.nas.nasa.gov/

SPEC CPU2017 Analysis

Application (speed version)	Parallel	static for	dynamic for	barrier (explicit)	master	single	reduction (nowait)	atomic (float8_add)	atomic (float8_max)	atomic (fixed4_add)	lock
603.bwaves	Yes	Yes					Yes	Yes	Yes		
607.cactuBSSN	Yes	Yes	Yes	Yes			Yes	Yes			
619.lbm	Yes	Yes									
621.wrf	Yes		Yes		Yes						
627.cam4	Yes	Yes	Yes	Yes	Yes						
628.pop2	Yes	Yes		Yes	Yes						
638.imagick	Yes	Yes		Yes	Yes	Yes					Yes
644.nab	Yes		Yes	Yes			Yes	Yes		Yes	
649.fotonik3d	Yes	Yes									
654.roms	Yes	Yes									

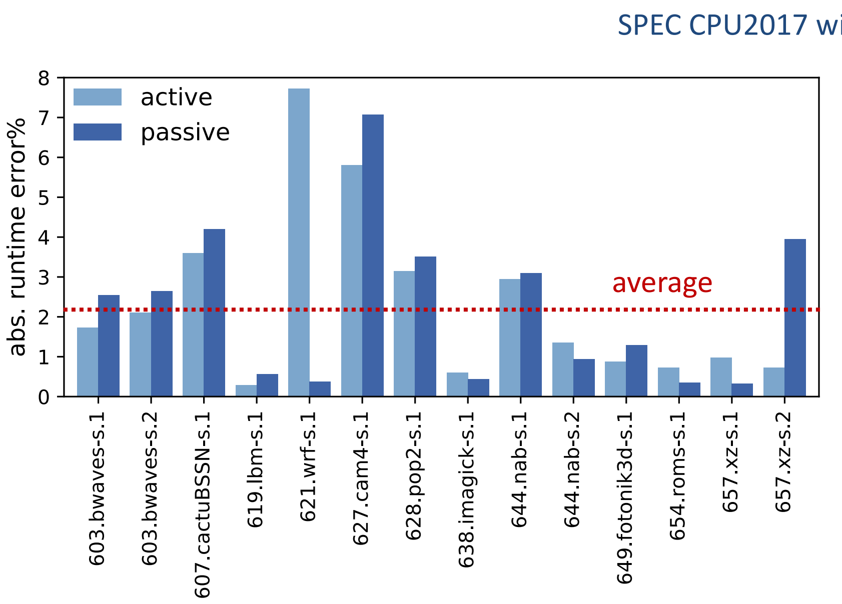
Source: SPEC CPU®2017 documentation index

Workload Type Supported

- Software
  - Static OpenMP scheduling (`OMP_WAIT_POLICY=STATIC`)
  - Homogeneous parallel threads doing similar amount of work
- Hardware
  - Simulated hardware needs to be homogeneous
  - No dynamic hardware events supported

Accuracy Results

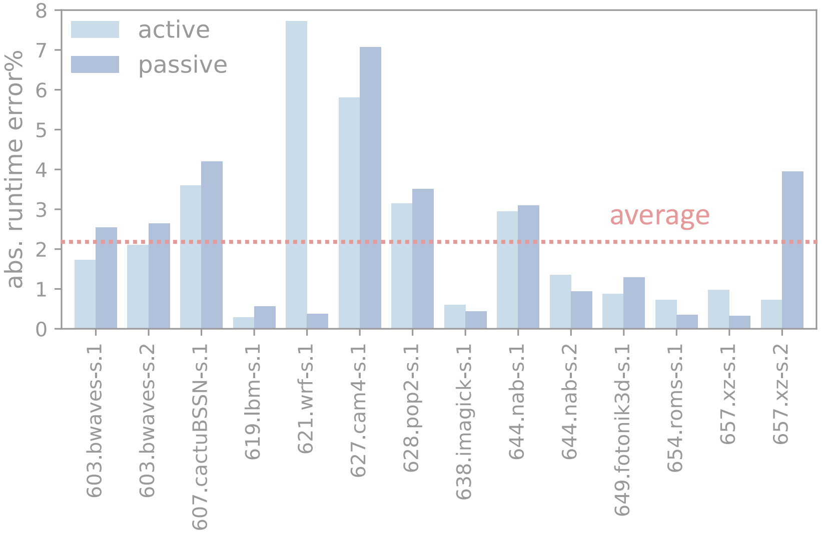
Prediction error wrt. performance of whole application

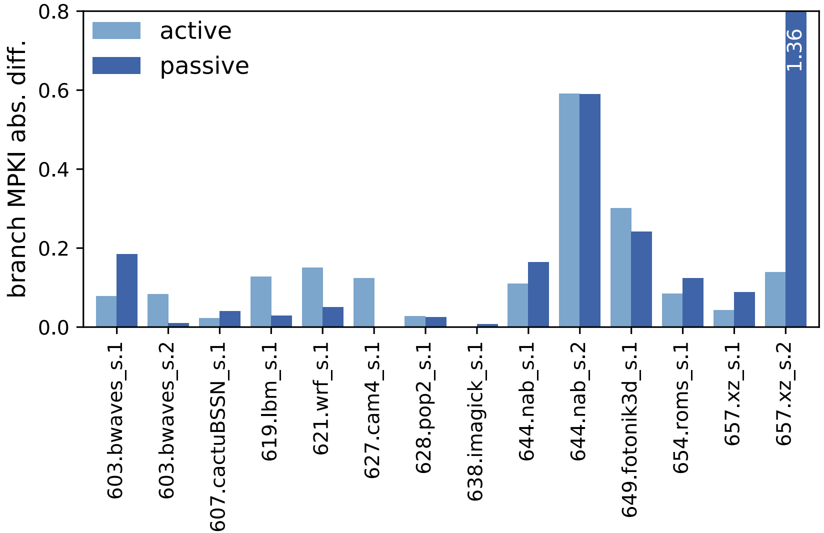


Accuracy Results

Prediction error wrt. performance of whole application

SPEC CPU2017 with train inputs, *8* threads

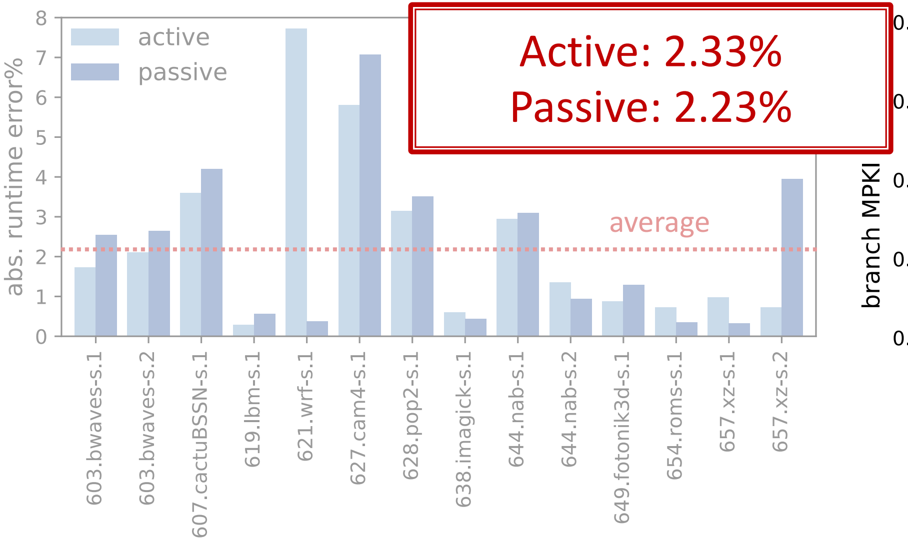


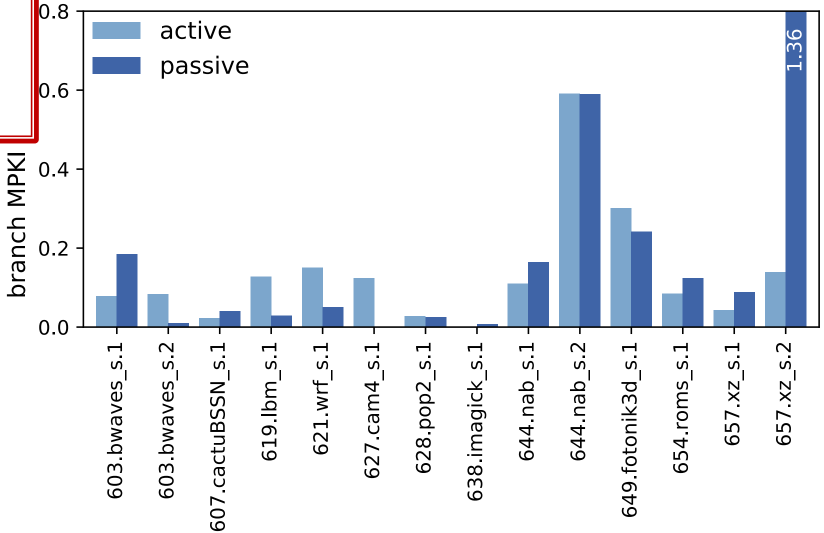


Accuracy Results

Prediction error wrt. performance of whole application

SPEC CPU2017 with train inputs, *8* threads

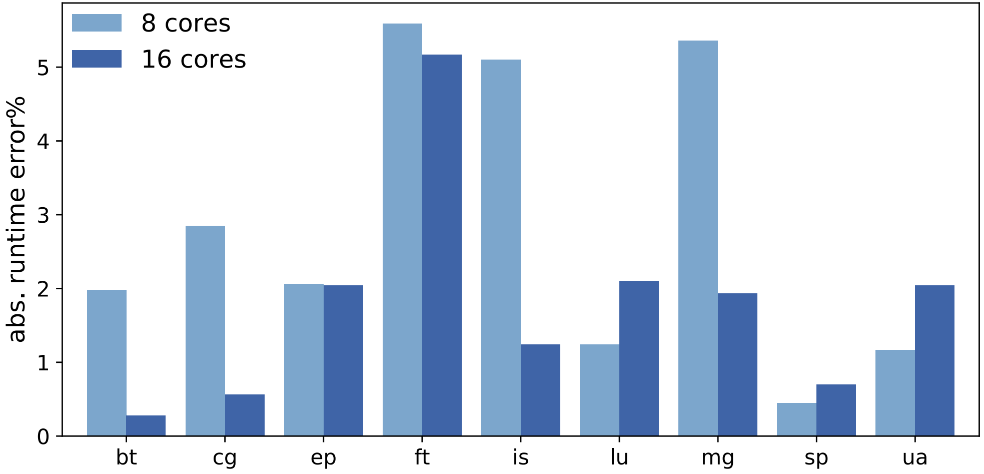




Changing Thread Count

Runtime prediction error wrt. whole application runtime

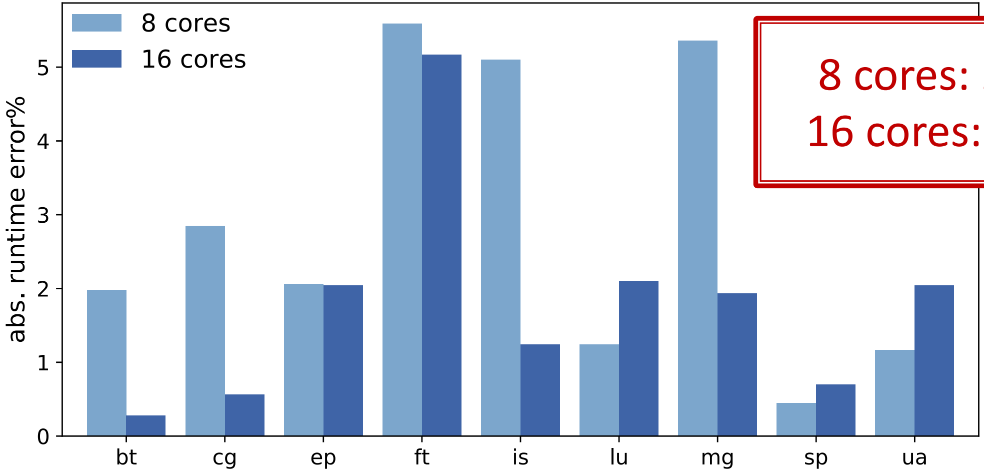
NPB 3.3 with *Class C* inputs, *8* and 16 threads, *passive* wait-policy



Changing Thread Count

Runtime prediction error wrt. whole application runtime

NPB 3.3 with *Class C* inputs, *8* and 16 threads, *passive* wait-policy

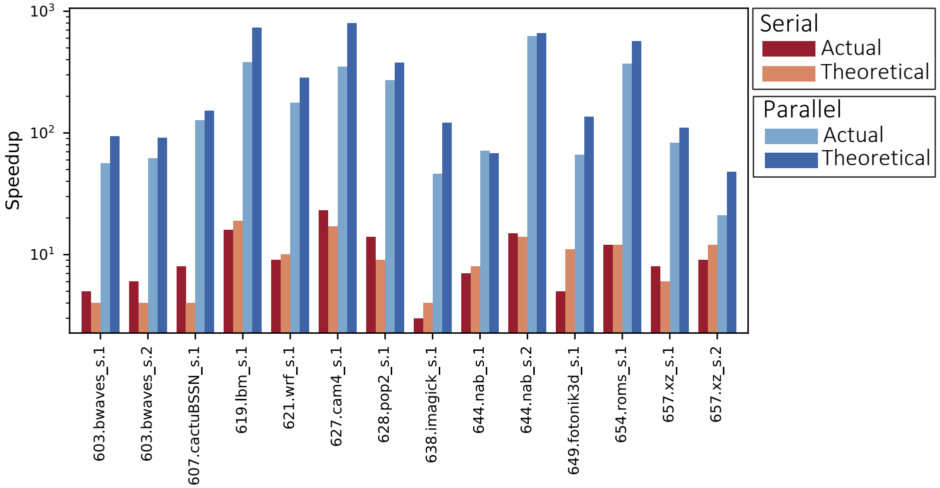


8 cores: 2.87%
16 cores: 1.78%

Speedup

Parallel and serial speedup achieved for LoopPoint

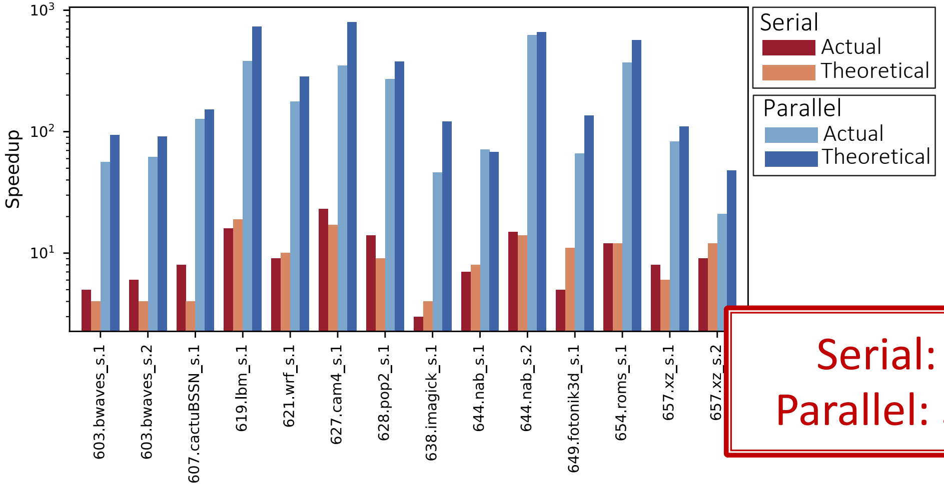
SPEC CPU2017 with *train* inputs, *8* threads, *active* wait-policy



Speedup

Parallel and serial speedup achieved for LoopPoint

SPEC CPU2017 with *train* inputs, *8* threads, *active* wait-policy

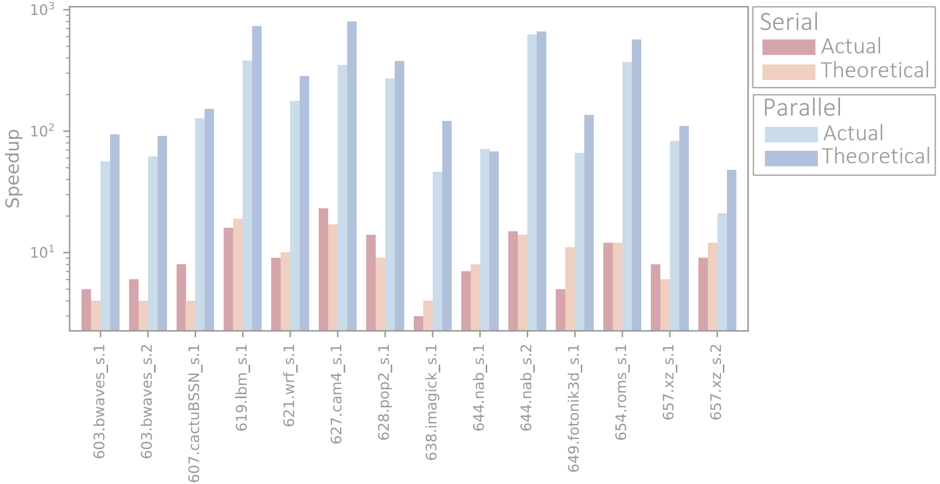


Serial: 9×
Parallel: 303×

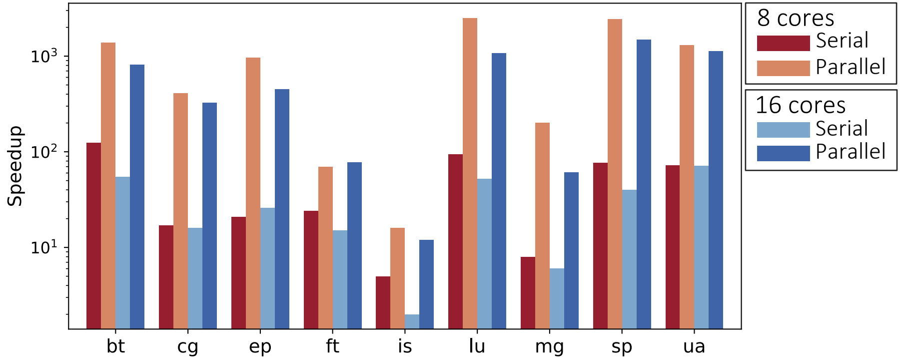
Speedup

Parallel and serial speedup achieved for LoopPoint

SPEC CPU2017 with *train* inputs, *8* threads, *active* wait-policy



NPB with *Class C* inputs, *8 and 16* threads, *passive* wait-policy

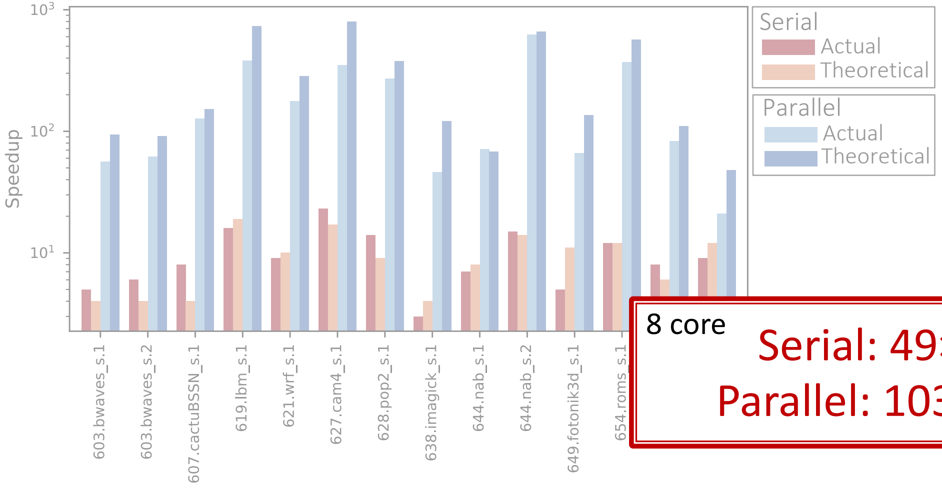


NUS National University of Singapore
intel

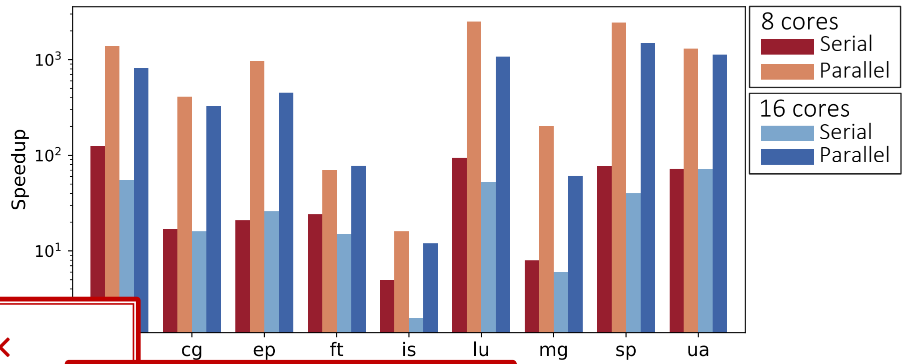
Speedup

Parallel and serial speedup achieved for LoopPoint

SPEC CPU2017 with *train* inputs, *8* threads, *active* wait-policy



NPB with *Class C* inputs, *8 and 16* threads, *passive* wait-policy

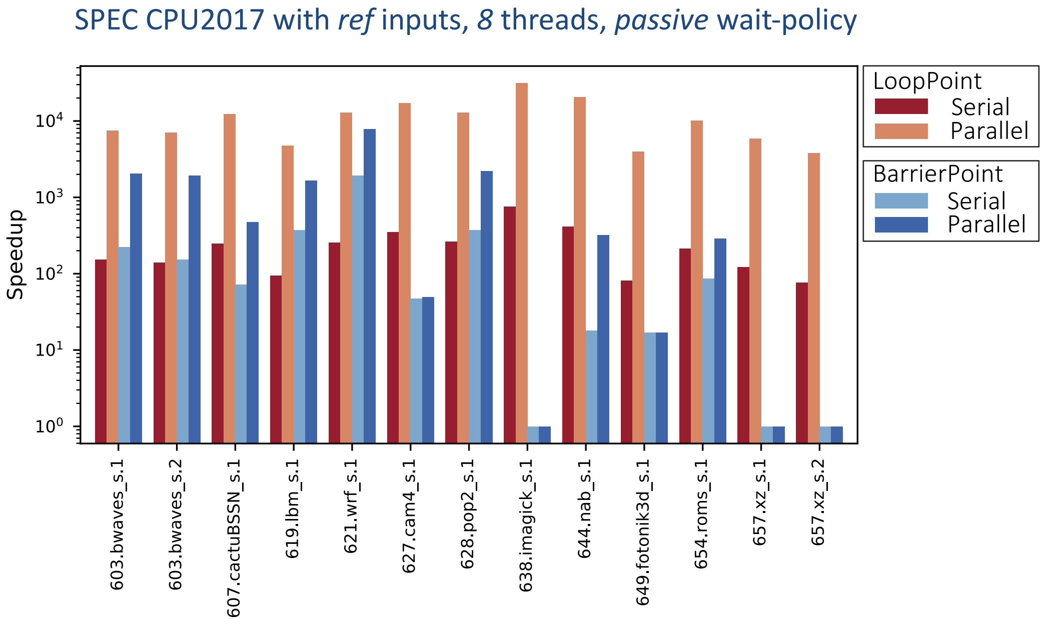


8 core
Serial: 49×
Parallel: 1031×

16 core
Serial: 31×
Parallel: 606×

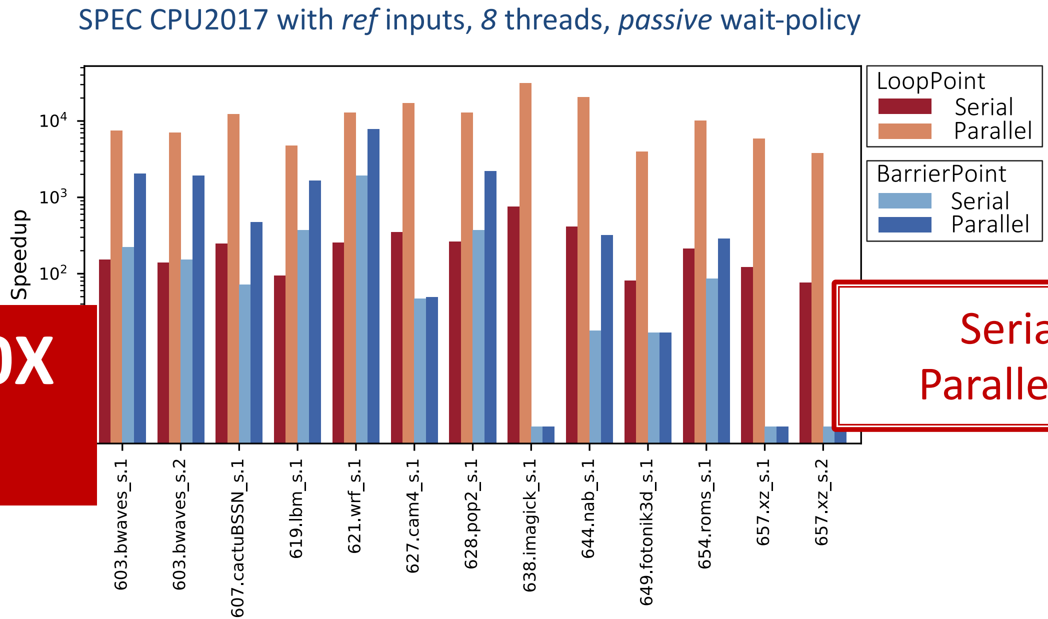
Speedup

Theoretical Speedup comparison with BarrierPoint



Speedup

Theoretical Speedup comparison with BarrierPoint



Up to 31000X speedup!

Serial: 244×
Parallel: 11587×

Summary

- Contributions
  - Methodology to sample generic multi-threaded workloads
  - Uses application loops (barring spinloops) as the unit of work
  - Flexible to be used for checkpoint-based simulation
- Accurate results in minimal time
  - Average absolute error of 2.3% across applications
  - Parallel speedup going up to 31,000 ×
  - Reduces simulation time from a few years to a few hours

More Information

- Links
  - Artifact: https://github.com/nus-comparch/looppoint
  - Page: https://looppoint.github.io
  - Short talk: https://youtu.be/Tr6O9MkT42g
  - Questions: alens@comp.nus.edu.sg, tcarlson@comp.nus.edu.sg

We can share our SPEC binaries and LoopPoint specifications if you have the SPEC user license

Agenda

Time	Speaker	Topic
13.20 to 13.30	Alen Sabu	Overview of the tutorial
13.30 to 14.30	Harish Patil	Tools & Methodologies: Pin, PinPlay, SDE, ELFies
14.30 to 15.00	Break	
15.00 to 15.50	Wim Heirman	Simulation with Sniper / Sniper 8.0 GitHub release
15.50 to 16.45	Alen Sabu	Single-threaded and Multi-threaded Sampling, LoopPoint
16.45 to 17.30	Alen Sabu	Running LoopPoint Tools

LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Computer Architecture, June 19th 2022, USA

Session 4

LoopPoint Demo

ALEN SABU, PHD CANDIDATE
NATIONAL UNIVERSITY OF SINGAPORE

Downloading Sniper 8.0

- Clone from https://github.com/snipersim/snipersim
- `export CC=gcc-9; export CXX=g++-9`
- `make` or `make USE_PINPLAY=1`
- Set `SNIPER_ROOT` to point to the Sniper base directory
- All set to use Sniper 8.0!
- Testing:
  - `make -C test/fft`

Downloading LoopPoint

- Prerequisites
  - x86-based Linux machine
  - Require GCC 9
  - Python
  - Docker

Downloading LoopPoint

- Opensource code
  - https://github.com/nus-comparch/looppoint.git
  - Clone the repo

```
[redacted]/isca2022 $ git clone https://github.com/nus-comparch/looppoint.git
Cloning into 'looppoint'...
remote: Enumerating objects: 320, done.
remote: Counting objects: 100% (168/168), done.
remote: Compressing objects: 100% (141/141), done.
remote: Total 320 (delta 27), reused 148 (delta 21), pack-reused 152
Receiving objects: 100% (320/320), 15.74 MiB | 13.79 MiB/s, done.
Resolving deltas: 100% (56/56), done.
Checking connectivity... done.
[redacted]/isca2022 $ ls
looppoint
```

Building LoopPoint

- `make build`
  - Build docker image

```
  Created wheel for tabulate: filename=tabulate-0.8.9-py2-none-any.whl size=33171 sha256=c170d0c5148145e2deb57b20db0b76d241909980d4dcea24
278faa8f3e0a3136
  Stored in directory: /tmp/pip-ephem-wheel-cache-5zZe7v/wheels/0a/4b/e1/d0e504a346ed0882b93f971fe1122b9de64fabebd9b1d81b9f
Successfully built tabulate
Installing collected packages: tabulate
Successfully installed tabulate-0.8.9
Removing intermediate container f962cd7c7f48
 ---> fdccc13883e7
Step 11/11 : RUN pip3 install --no-cache-dir  --upgrade pip &&     pip3 install --no-cache-dir numpy
 ---> Running in 89fa1a2a269a
Collecting pip
  Downloading https://files.pythonhosted.org/packages/a4/6d/6463d49a933f547439d6b5b98b46af8742cc03ae83543e4d7688c2420f8b/pip-21.3.1-py3-n
one-any.whl (1.7MB)
Installing collected packages: pip
  Found existing installation: pip 9.0.1
    Not uninstalling pip at /usr/lib/python3/dist-packages, outside environment /usr
Successfully installed pip-21.3.1
WARNING: pip is being invoked by an old script wrapper. This will fail in a future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip directly.
Collecting numpy
  Downloading numpy-1.19.5-cp36-cp36m-manylinux2010_x86_64.whl (14.8 MB)
Installing collected packages: numpy
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is
 recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Successfully installed numpy-1.19.5
Removing intermediate container 89fa1a2a269a
 ---> b006ee297a64
[Warning] One or more build-args [TZ_ARG] were not consumed
Successfully built b006ee297a64
Successfully tagged ubuntu:18.04-looppoint
```

Building LoopPoint

- `make build`
  - Build docker image

```
  Created wheel for tabulate: filename=tabulate-0.8.9-py2-none-any.whl size=33171 sha256=c170d0c5148145e2deb57b20db0b76d241909980d4dcea24
278faa8f3e0a3136
  Stored in directory: /tmp/pip-ephem-wheel-cache-5zZe7v/wheels/0a/4b/e1/d0e504a346ed0882b93f971fe1122b9de64fabebd9b1d81b9f
Successfully built tabulate
Installing collected packages: tabulate
Successfully installed tabulate-0.8.9
Removing intermediate container f962cd7c7f48
 ---> fdccc13883e7
Step 11/11 : RUN pip3 install --no-cache-dir  --upgrade pip &&     pip3 install --no-cache-dir numpy
 ---> Running in 89fa1a2a269a
Collecting pip
  Downloading https://files.pythonhosted.org/packages/a4/6d/6463d49a933f547439d6b5b98b46af8742cc03ae83543e4d7688c2420f8b/pip-21.3.1-py3-n
one-any.whl (1.7MB)
Installing collected packages: pip
```

```
Successfully built b006ee297a64
Successfully tagged ubuntu:18.04-looppoint
```

```
Installing collected packages: numpy
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is
 recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Successfully installed numpy-1.19.5
Removing intermediate container 89fa1a2a269a
 ---> b006ee297a64
[Warning] One or more build-args [TZ_ARG] were not consumed
Successfully built b006ee297a64
Successfully tagged ubuntu:18.04-looppoint
```

Building LoopPoint

- `make build`
- `make`
  - Run the docker image

```
[redacted]/isca2022/looppoint (main)$ make
docker run --rm -it -v "[redacted]/isca2022/looppoint:[redacted]/isca2022/looppoint"
--user 2014:100 -w "[redacted]/isca2022/looppoint" ubuntu:18.04-looppoint
I have no name!@9b31dd16ef4e:[redacted]/isca2022/looppoint$ ls
Dockerfile-ubuntu-18.04  README.md  lplib.py    run-looppoint.py  tools
Makefile                 apps       preprocess  suites.py
I have no name!@9b31dd16ef4e:[redacted]/isca2022/looppoint$
```

Building LoopPoint

- `make build`
- `make`
- `make apps`
  - Build the demo applications
  - Source code of the apps
    - `apps/demo/matrix-omp`
    - `apps/demo/dotproduct-omp`

```
I have no name!@9b31dd16ef4e:[redacted]/isca2022/looppoint$ make apps
make -C apps/demo/matrix-omp
make[1]: Entering directory '[redacted]/isca2022/looppoint/apps/demo/matrix-omp'
g++ -g -O3 -fopenmp -o matrix-omp matrix-omp-init.cpp matrix-omp.cpp -static
/usr/lib/gcc/x86_64-linux-gnu/9/libgomp.a(target.o): In function `gomp_target_init':
(.text+0x358): warning: Using 'dlopen' in statically linked applications requires at
runtime the shared libraries from the glibc version used for linking
ln -s matrix-omp base.exe
make[1]: Leaving directory '[redacted]/isca2022/looppoint/apps/demo/matrix-omp'
make -C apps/demo/dotproduct-omp
make[1]: Entering directory '[redacted]/isca2022/looppoint/apps/demo/dotproduct-omp'
g++ -g -O3 -fopenmp -o dotproduct-omp dot_product_openmp.cpp
ln -s dotproduct-omp base.exe
make[1]: Leaving directory '[redacted]/isca2022/looppoint/apps/demo/dotproduct-omp'
I have no name!@9b31dd16ef4e:[redacted]/isca2022/looppoint$
```

Building LoopPoint

- `make build`
- `make`
- `make apps`
- `make tools`
  - Build Sniper and LoopPoint tools

```
I have no name!@f3f87f6c10eb:[redacted]/isca2022/looppoint$ make tools
Downloading SDE kit
--2022-06-19 09:04:36--  https://downloadmirror.intel.com/684899/sde-external-9.0.0-2021-11-07-lin.tar.xz
Resolving downloadmirror.intel.com (downloadmirror.intel.com)... 13.33.88.124, 13.33.88.27, 13.33.88.68, ...
Connecting to downloadmirror.intel.com (downloadmirror.intel.com)|13.33.88.124|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26240092 (25M) [binary/octet-stream]
Saving to: 'STDOUT'

-                      100%[=====================>]  25.02M  10.8MB/s    in 2.3s

2022-06-19 09:04:38 (10.8 MB/s) - written to stdout [26240092/26240092]
```

Downloading Intel SDE

Downloading Sniper

```
Downloading Sniper from https://github.com/snipersim/snipersim
make -C tools/sniper
make[1]: Entering directory '[redacted]/isca2022/looppoint/tools/sniper'
Using SDE kit
Building for x86 (intel64)
[DOWNLO] SDE 9.0.0
[DOWNLO] pinplay-scripts
[DOWNLO] Pin 3.18-98332
[DOWNLO] mbuild
[DOWNLO] xed
[INSTAL] xed
[PYTHON VERSION] 2.7.17
[GIT VERSION] v10.0-298-g2be2d28
[GCC VERSION] 9
          .
          .
          .
make[4]: Entering directory '[redacted]/isca2022/looppoint/tools/sniper/sift/recorder'
make[4]: Leaving directory '[redacted]/isca2022/looppoint/tools/sniper/sift/recorder'
make[3]: Leaving directory '[redacted]/isca2022/looppoint/tools/sniper/sift/recorder'
make[2]: Leaving directory '[redacted]/isca2022/looppoint/tools/sniper/sift'
make[2]: Entering directory '[redacted]/isca2022/looppoint/tools/sniper/standalone'
[DEP    ] standalone/standalone.d
[DEP    ] standalone/exceptions.d
[CXX    ] standalone/exceptions.o
[CXX    ] standalone/standalone.o
[LD     ] lib/sniper
```

Sniper build completed

Building LoopPoint

- Opensource code
  - https://github.com/nus-comparch/looppoint.git
  - Clone the repo
- LoopPoint script
  - `make build`
    - Build docker image
  - `make`
    - Run docker image
  - `make apps`
    - Build the demo applications
  - `make tools`
    - Build Sniper and LoopPoint tools

Running LoopPoint

- Use LoopPoint driver script
  - `./run–looppoint.py –h`
  - Provides the information on how to run the tool

```
I have no name!@1fbfad8b73ce:[redacted]/isca2022/looppoint$ ./run-looppoint.py -h
Benchmarks:
  demo:
    dotproduct matrix

The tool helps reproduce some of the major results showed in LoopPoint paper.
Usage:
    run-looppoint.py
    [-h | --help]: Help
    [-n | --ncores=<num of threads> (8)]
    [-i | --input-class=<input class> (test)]
    [-w | --wait-policy=<omp wait policy> (passive)]
    [-p | --program=<suite-application-input> (demo-dotproduct-1)]: Ex. demo-matrix-1,cpu2017-bwaves-1
    [--force]: Start a new set of end-to-end run
    [--reuse-profile]: Reuse the profiling data (used along with --force)
    [--reuse-fullsim]: Reuse the full program simulation (used along with --force)
    [--no-flowcontrol]: Disable thread flowcontrol during profiling
    [--use-pinplay]: Use PinPlay instead of SDE for profiling
    [--native]: Run the application natively
```

Running LoopPoint

- Example run command
  - ./run-looppoint.py -p demo-dotproduct-1 -n 8 --force

```
I have no name!@1fbfad8b73ce:[redacted]/isca2022/looppoint$ ./run-looppoint.py -p demo-dotproduct-1 -n 8 --force
[LOOPPOINT] Generating fat pinball.
[PREPROCESS] dotproduct-omp
[PREPROCESS] apps/demo/dotproduct-omp/dotproduct-omp
[PREPROCESS] [redacted]/isca2022/looppoint/apps/demo/dotproduct-omp/dotproduct-omp
[PREPROCESS] symlinking dotproduct-omp /tmp/tmpcdMI_d/base.exe
[PREPROCESS] apps/demo/dotproduct-omp/test
[PREPROCESS] [redacted]/isca2022/looppoint/apps/demo/dotproduct-omp/test
[PREPROCESS] symlinking [redacted]/isca2022/looppoint/apps/demo/dotproduct-omp/test/dotproduct-omp.1.cfg /tmp/tmp
cdMI_d/dotproduct-omp.1.cfg
[PREPROCESS] Done
***  TRACING: START  ***    June 19, 2022 10:03:27
Script version $Revision:1.128$
Script:                    sde_pinpoints.py
Script args:               --delete --mode mt --sdehome=[redacted]/isca2022/looppoint/tools/sde-external-9.0.0-20
21-11-07-lin --cfg [redacted]/isca2022/looppoint/apps/demo/dotproduct-omp/test/dotproduct-omp.1.cfg --log_options
 -start_address main -log:fat  -log:mp_atomic 0 -log:mp_mode 0 -log:strace -log:basename [redacted]/isca2022/loop
point/results/demo-dotproduct-1-test-passive-8-20220619100327/whole_program.1/dotproduct.1 --replay_options=-repl
ay:strace -l
```

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application
    - `make_mt_pinball` : Generate whole-program pinball
    - `gen_dcfg` : Generate DCFG file to identify loop information
    - `gen_bbv` : Generate feature vector of each region
    - `gen_cluster` : Cluster regions

Fat Pinball

- Makes Pin-based analyses repeatable.
- Command:
  - `$SDE_KIT/pinplay-scripts/sde_pinpoints.py --mode mt --cfg=$CFGFILE --log_options="-start_address main -log:fat -log:basename $WPP_BASE” --replay_options="-replay:strace" –l`
- Generates a whole-program pinball for further profiling steps

DCFG Generation

- A dynamic control-flow graph (DCFG) is a specialized control-flow graph that adds data from a specific execution of a program
- C++ DCFG APIs available for accessing the data
  - `DCFG_LOOP_CONTAINER::get_loop_ids`
    - Get the set of loop IDs
  - `DCFG_LOOP`
    - `get_routine_id` : get the function that the loop belongs to
    - `get_parent_loop_id` : get the parent loop

DCFG Generation

- A dynamic control-flow graph (DCFG) is a specialized control-flow graph that adds data from a specific execution of a program
- C++ DCFG APIs available for accessing the data.
- More APIs can be found in
  - `tools/sde-external-9.0.0-2021-11-07-lin/pinkit/sde-example/include`
    - `dcfg_api.H`
    - `dcfg_pin_api.H`
    - `dcfg_trace_api.H`

DCFG Generation

- Collect Loop Information
- Command:
  - `$SDE_BUILD_KIT/pinplay-scripts/replay.py --pintool=sde-global-looppoint.so --pintool_options "-dcfg -replay:deadlock_timeout 0 -replay:strace -dcfg:out_base_name $DCFG_BASE $WPP_BASE"`
  - -dcfg : enable DCFG generation
  - DCFG_BASE : the basename of DCFG that is generated

BBV Generation

- Profiling the feature vector of each region
- Command:
  - `$SDE_BUILD_KIT/pinplay-scripts/sde_pinpoints.py --pintool="sde-global-looppoint.so" --global_regions --pccount_regions --cfg $CFG --whole_pgm_dir $WPP_DIR --mode mt -S $SLICESIZE -b --replay_options "-replay:deadlock_timeout 0 -global_profile -emit_vectors 0 -filter_exclude_lib libgomp.so.1 -filter_exclude_lib libiomp5.so -looppoint:global_profile -looppoint:dcfg-file $DCFG -looppoint:main_image_only 1 -looppoint:loop_info $PROGRAM.$INPUT.loop_info.txt -flowcontrol:verbose 1 -flowcontrol:quantum 1000000 -flowcontrol:maxthreads $NCORES”`
  - `-pccount_regions` : (PC, count)-based region information
  - `-S $SLICESIZE`: The *global* instruction count for each region
  - `-filter_exclude_lib`: Exclude libraries from profiling information

BBV Generation

- Profiling the feature vector of each region
- Command:
  - $SDE_BUILD_KIT/pinplay-scripts/sde_pinpoints.py --pintool="sde-global-looppoint.so" --global_regions --pccount_regions --cfg $CFG --whole_pgm_dir $WPP_DIR --mode mt -S $SLICESIZE -b --replay_options "-replay:deadlock_timeout 0 -global_profile -emit_vectors 0 -filter_exclude_lib libgomp.so.1 -filter_exclude_lib libiomp5.so -looppoint:global_profile -looppoint:dcfg-file $DCFG -looppoint:main_image_only 1 -looppoint:loop_info $PROGRAM.$INPUT.loop_info.txt -flowcontrol:verbose 1 -flowcontrol:quantum 1000000 -flowcontrol:maxthreads $NCORES”
  - -looppoint:main_image_only: Select only main image for choosing markers
  - -looppoint:loop_info : Utilize loop information as the marker of each region
  - -flowcontrol:quantum : synchronize each thread every 1000000 instructions

Clustering

- Cluster all regions into several groups.
  - SimPoint [1]
  - Utilize feature vectors of all threads
  - kmeans algorithm

[1] Sherwood et al., “Automatically Characterizing Large Scale Program Behavior”, ASPLOS’02

Clustering

- Cluster all regions into several groups.
- Command
  - `$SDE_BUILD_KIT/pinplay-scripts/sde_pinpoints.py --pintool="sde-global-looppoint.so" --cfg $CFG --whole_pgm_dir $WPP_DIR -S $SLICESIZE --warmup_factor=2 --maxk=$MAXK --append_status -s --simpoint_options="-dim $DIM -coveragePct 1.0 -maxK $MAXK”`
  - `DIM` : The reduced dimension of the vector that BBVs are projected to
  - `MAXK` : Maximum number of clusters for kmeans

Running LoopPoint

- The LoopPoint driver script
  - Profiling Results:
    - `dotproduct.1_52.global.pinpoints.csv`
    - (start-pc, start-pc-count), (end-pc, end-pc-count)

```
# comment,thread-id,region-id,start-pc, start-image-name, start-image-offset, start-pc-count,end-pc, end-image-name, end-image-offset, end-pc-count,end-pc-relative-count, region-length, region-weight, region-multiplier, region-type

# RegionId = 1 Slice = 1 Icount = 80000008 Length = 80000066 Weight = 0.10000 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 80000066
#Start: pc : 0x555555554e80 image: dotproduct-omp offset: 0xe80 absolute_count: 1588076  source-info: Unknown:0
#End: pc : 0x5555555553c0 image: dotproduct-omp offset: 0x13c0 absolute_count: 3383564  relative_count: 243103.0 source-info: Unknown:0
cluster 0 from slice 1,global,1,0x555555554e80,dotproduct-omp,0xe80,1588076,0x5555555553c0,dotproduct-omp,0x13c0,3383564,243103,80000066,0.10000,1.000,simulation
```

Running LoopPoint

- The LoopPoint driver script
  - Profiling Results:
    - `dotproduct.1_52.global.pinpoints.csv`
    - (start-pc, start-pc-count), (end-pc, end-pc-count)

```
# comment,thread-id,region-id,start-pc, start-image-name, start-image-offset, start-pc-count,end-pc, end-image-name, end-image-offset, end-pc-count,end-pc-relative-count, region-length, region-weight, region-multiplier, region-type

# RegionId = 1 Slice = 1 Icount = 80000008 Length = 80000066 Weight = 0.10000 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 80000066
#Start: pc : 0x555555554e80 image: dotproduct-omp offset: 0xe80 absolute_count: 1588076  source-info: Unknown:0
#End: pc : 0x5555555553c0 image: dotproduct-omp offset: 0x13c0 absolute_count: 3383564  relative_count: 243103.0 source-info: Unknown:0
cluster 0 from slice 1,global,1,0x555555554e80,dotproduct-omp,0xe80,1588076,0x5555555553c0,dotproduct-omp,0x13c0,3383564,243103,80000066,0.10000,1.000,simulation
```

Running LoopPoint

- The LoopPoint driver script
  - Profiling Results:
    - `dotproduct.1_52.global.pinpoints.csv`
    - `(start-pc, start-pc-count ), (end-pc, end-pc-count)`
    - Cluster group id

```
# comment,thread-id,region-id,start-pc, start-image-name, start-image-offset, start-pc-count,end-pc, end-image-name, end-image-offset, end-pc-count,end-pc-relative-count, region-length, region-weight, region-multiplier, region-type

# RegionId = 1 Slice = 1 Icount = 80000008 Length = 80000066 Weight = 0.10000 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 80000066
#Start: pc : 0x555555554e80 image: dotproduct-omp offset: 0xe80 absolute_count: 1588076  source-info: Unknown:0
#End: pc : 0x5555555553c0 image: dotproduct-omp offset: 0x13c0 absolute_count: 3383564  relative_count: 243103.0 source-info: Unknown:0
cluster 0 from slice 1,global,1,0x555555554e80,dotproduct-omp,0xe80,1588076,0x5555555553c0,dotproduct-omp,0x13c0,3383564,243103,80000066,0.10000,1.000,simulation
```

Running LoopPoint

- The LoopPoint driver script
  - Profiling Results:
    - `dotproduct.1_52.global.pinpoints.csv`
    - `(start-pc, start-pc-count ), (end-pc, end-pc-count)`
    - Cluster group id
    - Cluster multiplier

```
# comment,thread-id,region-id,start-pc, start-image-name, start-image-offset, start-pc-count,end-pc, end-image-name, end-image-offset, end-pc-count,end-pc-relative-count, region-length, region-weight, region-multiplier, region-type

# RegionId = 1 Slice = 1 Icount = 80000008 Length = 80000066 Weight = 0.10000 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 80000066
#Start: pc : 0x555555554e80 image: dotproduct-omp offset: 0xe80 absolute_count: 1588076  source-info: Unknown:0
#End: pc : 0x5555555553c0 image: dotproduct-omp offset: 0x13c0 absolute_count: 3383564  relative_count: 243103.0 source-info: Unknown:0
cluster 0 from slice 1,global,1,0x555555554e80,dotproduct-omp,0xe80,1588076,0x5555555553c0,dotproduct-omp,0x13c0,3383564,243103,80000066,0.10000,1.000,simulation
```

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application
    - `dotproduct.1_52.global.pinpoints.csv`
    - Sampled Simulation : `(start-pc, start-pc-count ), (end-pc, end-pc-count), cluster group id`
    - Extrapolation : `cluster group id, cluster-multiplier`

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application
  - Sampled simulation of selected regions

Simulation using Sniper

- LoopPoint support in Sniper 8.0 (using Intel SDE)
- Handle the beginning and ending of representative regions
    - Using PC-based markers
  - Sniper shifts simulation modes based on signals from Pin/SDE

Simulation using Sniper

- LoopPoint support in Sniper 8.0 (using Intel SDE)
  - Handle the beginning and ending of representative regions
  - ```
    ./run-sniper -n 8 -gscheduler/type=static  -cgainestown  -ssimuserroi  --roi-script  --trace-args=-control start:address:<PC>:count<Count>:global --trace-args=-control stop:address:<PC>:count<Count>:global -- <app cmd>
    ```
  - Region start: -control start:address:<PC>:count<Count>
  - Region end: -control end:address:<PC>:count<Count>
  - *PC*, Count : LoopPoint region boundaries
  - Note: Use -pinplay:control if Pin/Pinplay is used instead of SDE
    ```

Simulation using Sniper

```
./run-sniper        -n 8        -v -sprogresstrace:10000000 -gtraceinput/timeout=2000 -gscheduler/type=static -
cgainestown --trace-args=-sniper:flow 1000 -ssimuserroi --roi-script --trace-args=-control start:address:
0x5555555553c0:count8095299:global --trace-args=-control stop:address:0x5555555553c0:count16984191:global -
gperf_model/fast_forward/oneipc/interval=100 -ggeneral/inst_mode_init=detailed -gperf_model/fast_forward/oneipc/
include_memory_latency=true -- ./base.exe
```

Simulation using Sniper

Start PC and count

```
./run-sniper        -n 8        -v -sprogresstrace:10000000 -gtraceinput/timeout=2000 -gscheduler/type=static -
cgainestown --trace-args=-sniper:flow 1000 -ssimuserroi --roi-script --trace-args=-control start:address:
0x5555555553c0:count8095299:global --trace-args=-control stop:address:0x5555555553c0:count16984191:global -
gperf_model/fast_forward/oneipc/interval=100 -ggeneral/inst_mode_init=detailed -gperf_model/fast_forward/oneipc/
include_memory_latency=true -- ./base.exe
```

Application

End PC and count

Simulation using Sniper

```
[PROGRESS] 700M instructions, 3198 KIPS, 2.37 IPC
[PROGRESS] 710M instructions, 6004 KIPS, 8.00 IPC
[PROGRESS] 720M instructions, 5526 KIPS, 8.00 IPC
[CONTROLLER] tid: 5 ip: 0x00005555555553e2 658579928 Start
[SNIPER] Enabling performance models
[PROGRESS] 730M instructions, 608 KIPS, 1.97 IPC
[PROGRESS] 740M instructions, 469 KIPS, 1.61 IPC
[PROGRESS] 750M instructions, 455 KIPS, 1.61 IPC
[PROGRESS] 760M instructions, 447 KIPS, 1.61 IPC
[PROGRESS] 770M instructions, 447 KIPS, 1.61 IPC
[PROGRESS] 780M instructions, 446 KIPS, 1.61 IPC
[PROGRESS] 790M instructions, 446 KIPS, 1.61 IPC
[PROGRESS] 800M instructions, 448 KIPS, 1.61 IPC
[CONTROLLER] tid: 4 ip: 0x00005555555553e2 669005339 Stop
[SNIPER] Disabling performance models
[SNIPER] Leaving ROI after 176.54 seconds
[SNIPER] Simulated 80.0M instructions, 708.4M cycles, 0.11 IPC
[SNIPER] Simulation speed 453.2 KIPS (56.6 KIPS / target core - 17654.0ns/instr)
[SNIPER] Sampling: executed 7.03% of simulated time in detailed mode
[SNIPER] Setting instrumentation mode to FAST_FORWARD
[PROGRESS] 810M instructions, 1918 KIPS, 4.23 IPC
```

Simulation using Sniper

Warmup ends

Fast-forwarding the rest

```
[PROGRESS] 700M instructions, 3198 KIPS, 2.37 IPC
[PROGRESS] 710M instructions, 6004 KIPS, 8.00 IPC
[PROGRESS] 720M instructions, 5526 KIPS, 8.00 IPC
[CONTROLLER] tid: 5 ip: 0x00005555555553e2 658579928 Start
[SNIPER] Enabling performance models
[PROGRESS] 730M instructions, 608 KIPS, 1.97 IPC
[PROGRESS] 740M instructions, 469 KIPS, 1.61 IPC
[PROGRESS] 750M instructions, 455 KIPS, 1.61 IPC
[PROGRESS] 760M instructions, 447 KIPS, 1.61 IPC
[PROGRESS] 770M instructions, 447 KIPS, 1.61 IPC
[PROGRESS] 780M instructions, 446 KIPS, 1.61 IPC
[PROGRESS] 790M instructions, 446 KIPS, 1.61 IPC
[PROGRESS] 800M instructions, 448 KIPS, 1.61 IPC
[CONTROLLER] tid: 4 ip: 0x00005555555553e2 669005339 Stop
[SNIPER] Disabling performance models
[SNIPER] Leaving ROI after 176.54 seconds
[SNIPER] Simulated 80.0M instructions, 708.4M cycles, 0.11 IPC
[SNIPER] Simulation speed 453.2 KIPS (56.6 KIPS / target core - 17654.0ns/instr)
[SNIPER] Sampling: executed 7.03% of simulated time in detailed mode
[SNIPER] Setting instrumentation mode to FAST_FORWARD
[PROGRESS] 810M instructions, 1918 KIPS, 4.23 IPC
```

Detailed simulation

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application
  - Sampled simulation of selected regions
  - Extrapolation of performance results

Extrapolation of Performance Result

- Runtime of corresponding representative region : region_runtime
- Scaling factor : multiplier

```
for regionid, multiplier in region_mult.iteritems():
  region_runtime = 0
  try:
    region_runtime =  read_simstats(region_stats[regionid], region_config[regionid], 'runtime')
  except:
    print('[LOOPPOINT] Warning: Skipping r%s as the simulation results are not available' % regionid)
    continue
  cov_mult += multiplier
  extrapolated_runtime += region_runtime * multiplier
  if region_runtime > max_rep_runtime:
    max_rep_runtime = region_runtime
  sum_rep_runtime += region_runtime
```

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application
  - Sampled simulation of selected regions
  - Extrapolation of performance results
    - Predicted runtime using sampled simulation

application	runtime actual (ns)	runtime predicted (ns)	error (%)	speedup (parallel)	speedup (serial)	coverage (%)
dotproduct-omp.1	592169300.0	414953200.0	29.93	5.86	1.68	100.0

Running LoopPoint

- The LoopPoint driver script
  - Profiling the application
  - Sampled simulation of selected regions
  - Extrapolation of performance results
    - Predicted runtime using sampled simulation
    - The error rate of obtained using sampled simulation

application	runtime actual (ns)	runtime predicted (ns)	error (%)	speedup (parallel)	speedup (serial)	coverage (%)
dotproduct-omp.1	592169300.0	414953200.0	29.93	5.86	1.68	100.0

Coming soon!

- Gem5 support for LoopPoint region specification
- Release of 8-threaded SPEC CPU2017 representative pinballs
- Support for Open-source benchmarks (like NPB)

Thank you!

LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Computer Architecture, June 19th 2022, USA