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

**Alen Sabu[1], Changxi Liu[1], Akanksha Chaudhari[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]**

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Performance Analysis of Systems and Software, May 22nd 2022, Singapore

# Agenda

| Time | Speaker | Topic |
|---|---|---|
| 09.00 to 09.10 | Alen Sabu | Overview of the tutorial |
| 09.10 to 10.00 | Harish Patil | Tools from Intel: Pin, PinPlay, SDE, ELFies |
| 10.00 to 10.15 | | Break |
| 10.15 to 11.00 | Akanksha Chaudhari | Simulation and Single-threaded Sampling |
| 11.00 to 11.20 | | Break |
| 11.20 to 12.15 | Alen Sabu | Multi-threaded Sampling and LoopPoint |
| 12.15 to 13.00 | Changxi Liu | Running Sniper and LoopPoint Tools |

# Agenda

| Time | Speaker | Topic |
|---|---|---|
| 09.00 to 09.10 | Alen Sabu | Overview of the tutorial |
| 09.10 to 10.00 | Harish Patil | Tools from Intel: Pin, PinPlay, SDE, ELFies |
| 10.00 to 10.15 | Break | |
| 10.15 to 11.00 | Akanksha Chaudhari | Simulation and Single-threaded Sampling |
| 11.00 to 11.20 | Break | |
| 11.20 to 12.15 | Alen Sabu | Multi-threaded Sampling and LoopPoint |
| 12.15 to 13.00 | Changxi Liu | Running Sniper and 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

# Simulation and Sampling Overview

- Speaker: Akanksha Chaudhari
  - Research Assistant, National University of Singapore
- Topics Covered
  - Architectural exploration and evaluation
  - Simulation as a tool for performance estimation
  - Methods for fast estimation using simulation
  - State-of-the-art single-threaded sampled simulation techniques

# LoopPoint Methodology

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

# Simulation and Demo

- Speaker: Changxi Liu
  - PhD Student, National University of Singapore
- Topics Covered
  - Overview of Sniper simulator
  - High-level structure of LoopPoint code
  - Demo on how to use LoopPoint tools
  - Integrating workloads to run with LoopPoint

# Agenda

| Time | Speaker | Topic |
|---|---|---|
| 09.00 to 09.10 | Alen Sabu | Overview of the tutorial |
| 09.10 to 10.00 | Harish Patil | Tools from Intel: Pin, PinPlay, SDE, ELFies |
| 10.00 to 10.15 | Break | |
| 10.15 to 11.00 | Akanksha Chaudhari | Simulation and Single-threaded Sampling |
| 11.00 to 11.20 | Break | |
| 11.20 to 12.15 | Alen Sabu | Multi-threaded Sampling and LoopPoint |
| 12.15 to 13.00 | Changxi Liu | Running Sniper and LoopPoint Tools |

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

**Alen Sabu[1], Changxi Liu[1], Akanksha Chaudhari[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]**

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Performance Analysis of Systems and Software, May 22nd 2022, Singapore

Session 1

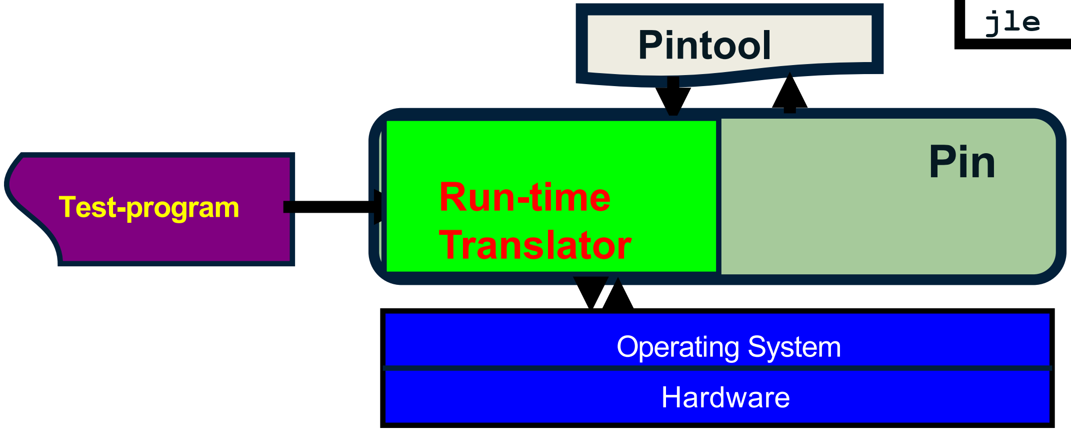
# Tools and Methodologies

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>
```



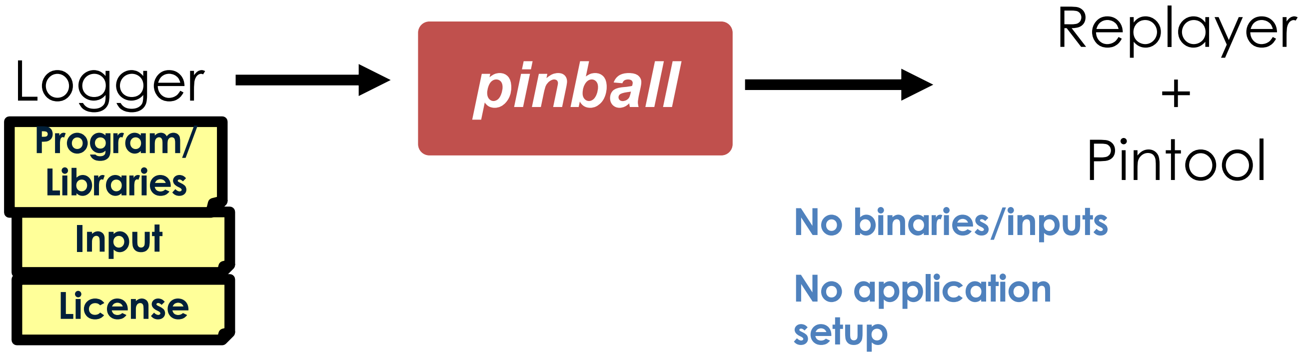


```
$ pin -t pintool -- test-program
```

Normal output +
*Analysis output*

[http://pintool.intel.com](http://pintool.intel.com)

# ***PinPlay***: Software-based User-level Capture and Replay




No binaries/inputs

No application setup

No license checking

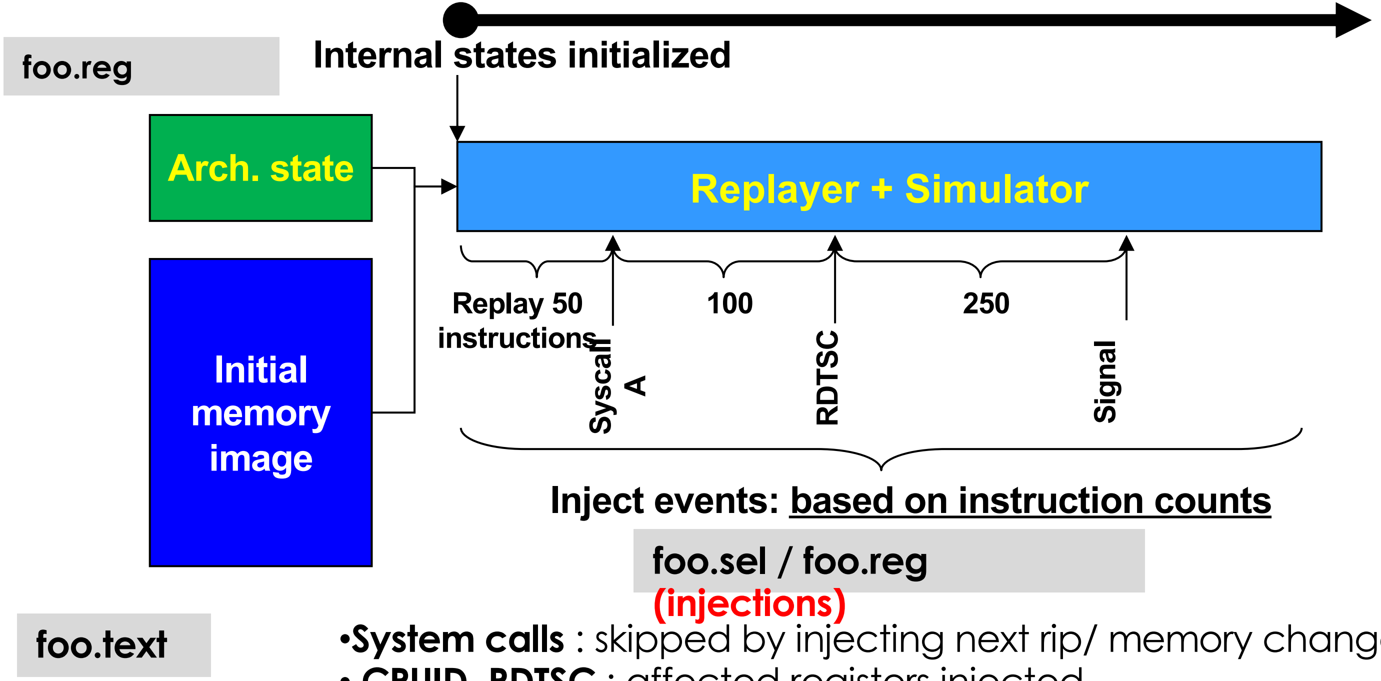
**Platforms :** Linux, Windows, MacOS

**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




- **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)

foo.reg (per-thread)

foo.sel (per-thread)

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

[T1] 2 T2 2

[T1] 3 T2 3

[T2] 5 T4 1

Thread T2 cannot execute instruction 5 until T4 executes instruction 1

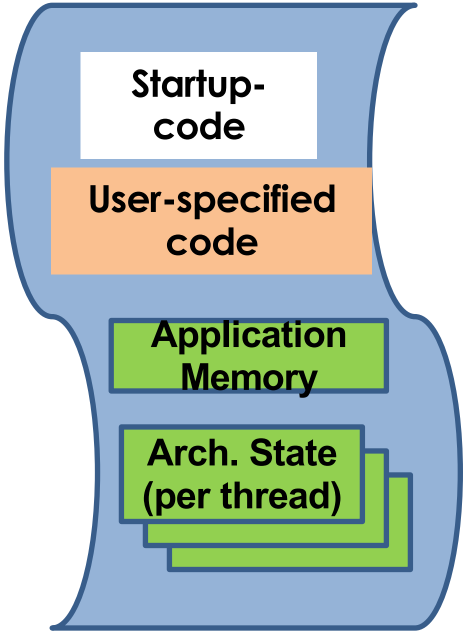
Thread T1 cannot execute instruction 2 until T2 executes instruction 2

foo.race (per-thread)

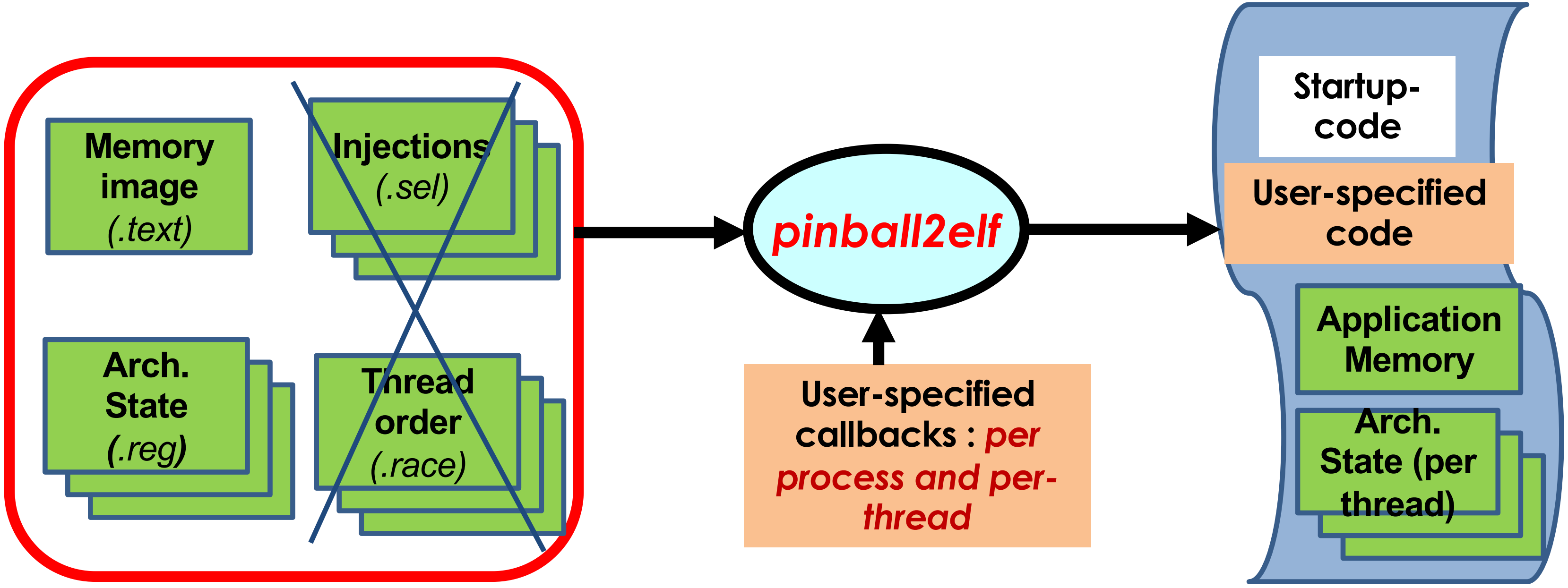
**MT Pinball == race-files provide determinism**

# 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<br>Simulator handles exit | YES, when retired instruction count reaches pinball icount |
| Environment variables used | NONE | ELFIE_VERBOSE=0/1<br>ELFIE_COREBASE=X<br>Set affinity : thread 0 → core X, thread 1 → core x+1 | "ELFIE_WARMUP" to decide whether to use warmup<br>"ELFIE_PCCONT" to decide how to end warmup/simulation regions<br>ELFIE_PERFLIST, enables performance counting |

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

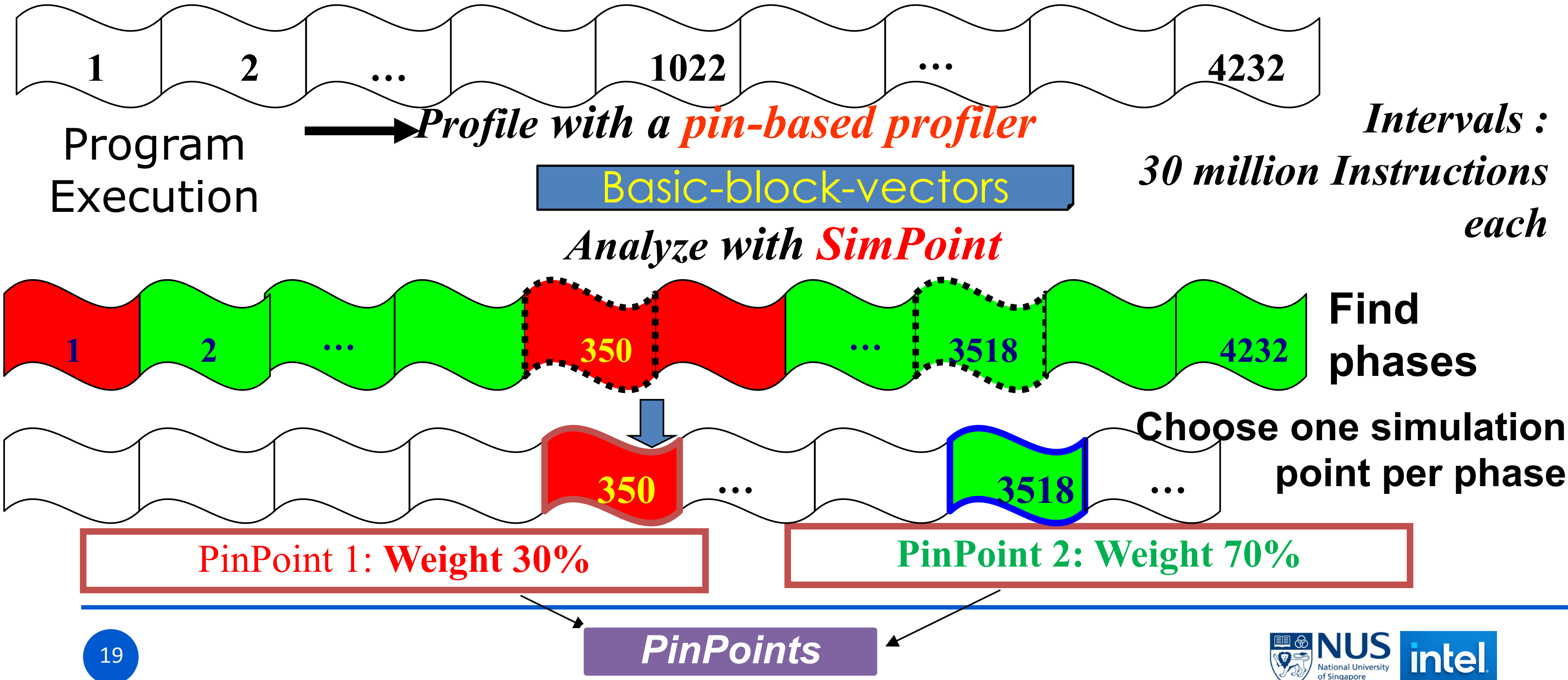
# Example: *ELFIE_PERFLIST* with a *perf ELFie*

*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')*

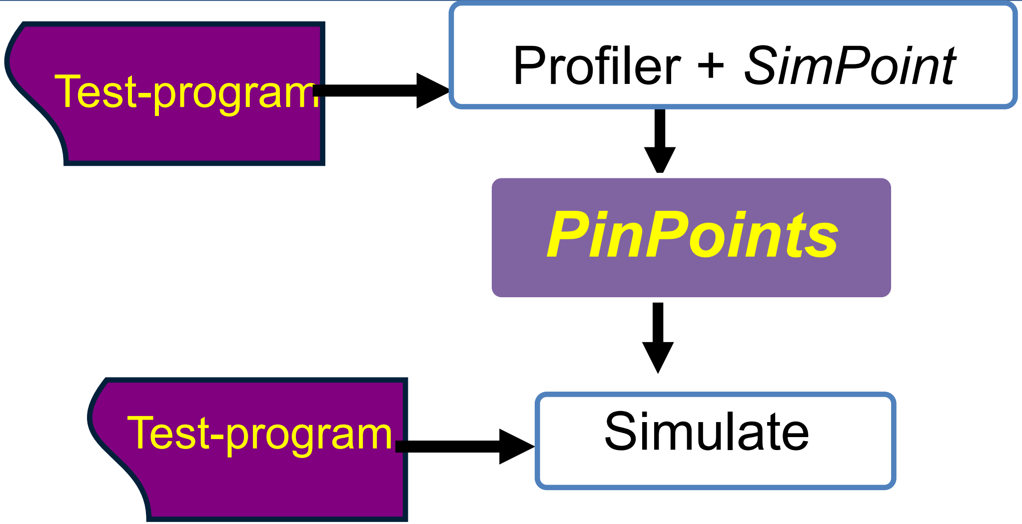
```
% cd examples/MT
% ../../scripts/pinball2elf.perf.sh pinball.mt/log_0 perf.out
% setenv ELFIE_PERFLIST "0:0,0:1,1:1"
% pinball.mt/log_0.perf.elfie
├── perf.out.0.perf.txt
├── perf.out.1.perf.txt
├── perf.out.2.perf.txt
```

```
ROI start: TSC 480511110586217756
Thread start: TSC 48051110623843452
-----------------------------------------------
Simulation end: TSC 48051110625045322
      Sim-end-icount 3436
hw_cpu_cycles:36148 hw_instructions:3476
sw_task_clock:141901
-----------------------------------------------
Thread end: TSC 48051110625366502
ROI end: TSC 48051110625959364
hw_cpu_cycles:40097 hw_instructions:4455
sw_task_clock:188637
```

# *PinPoints == Pin + SimPoint*



# *PinPoints* : The repeatability challenge

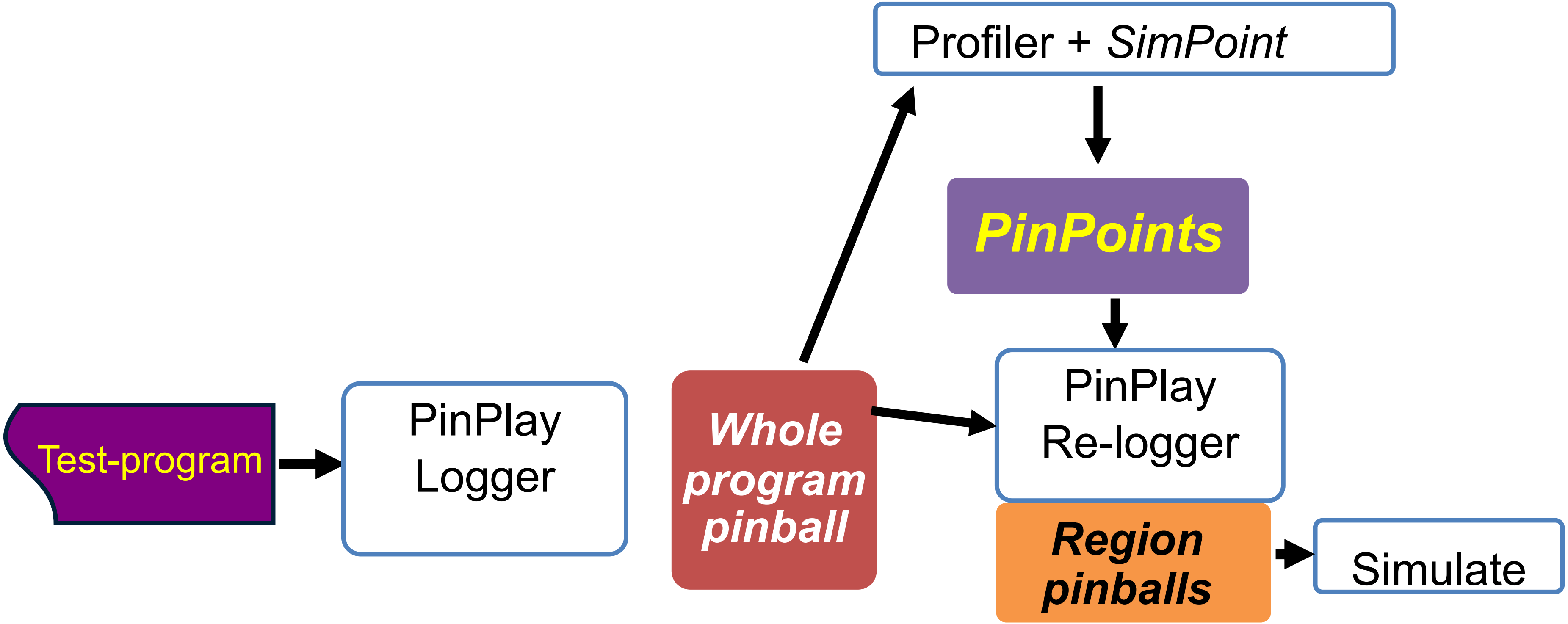



**Problem:** Two runs are not exactly same → PinPoints missed (PC marker based)

[ *"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

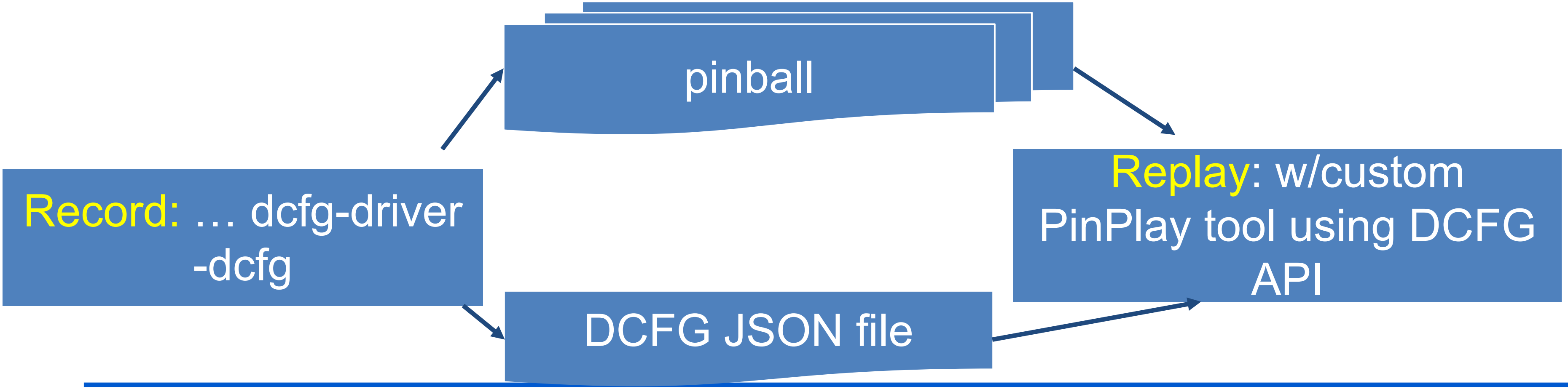


NUS National University of Singapore intel

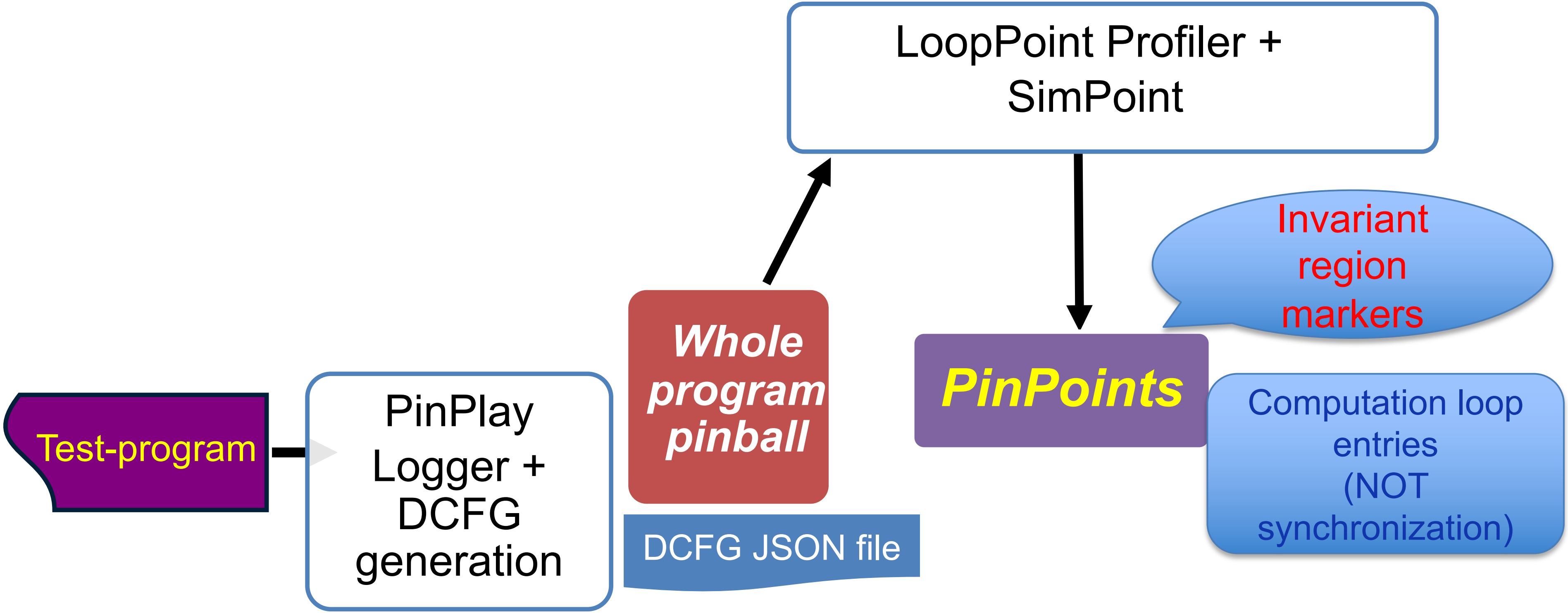
# *DCFG* Generation with *PinPlay*

**Dynamic Control-Flow Graph (DCFG)**

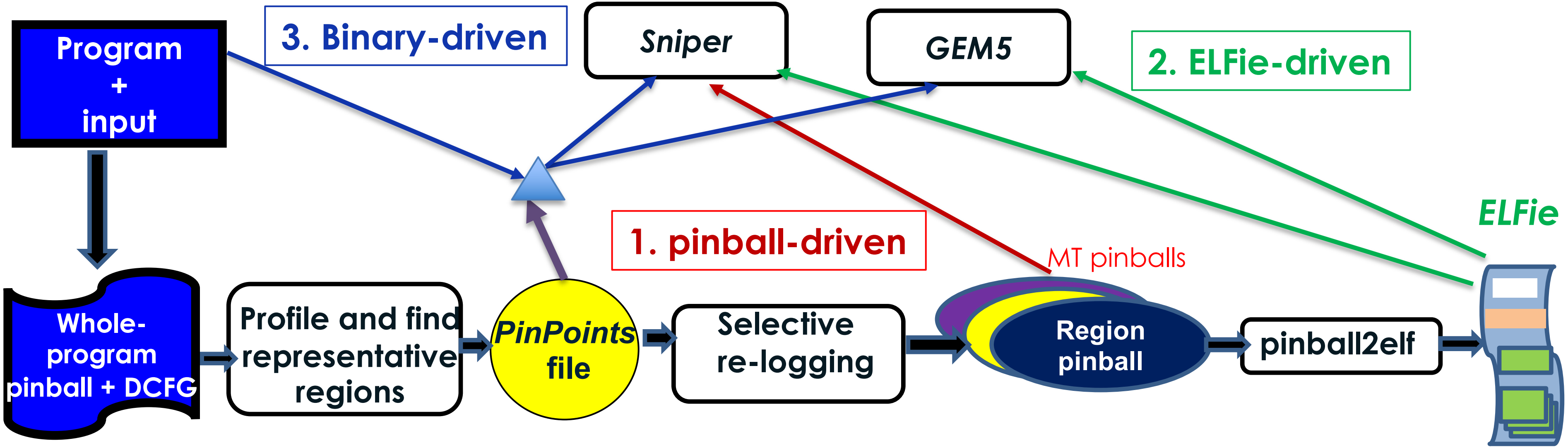
Directed graph extracted for a specific execution:
Nodes → basic blocks
Edges →control-flow : augmented with per-thread execution counts



# *PinPlay + DCFG :* Stronger Repeatability



# *LoopPoint:* Simulation alternatives



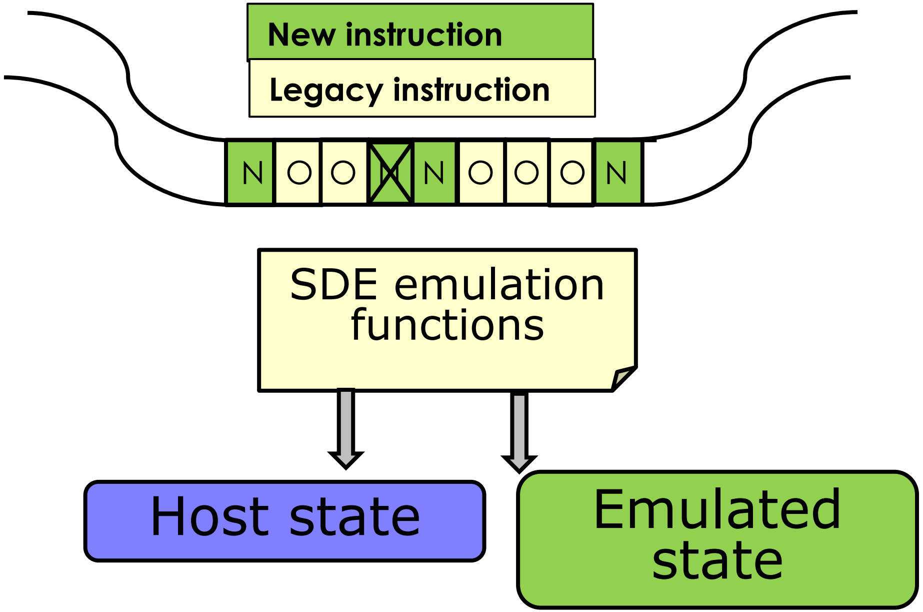

**Requirement**: Execution invariant region specification (PC+count for compute loop entries)

# 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)
- **Goal**: 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
```

- dotproduct.1_282016.Data → *bbv files (*.bb), PinPoints file(*.csv, *.CSV)*
- dotproduct.1_282016.pp → *Region pinballs*
- whole_program.1 → *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**



NUS National University of Singapore intel

# Agenda

| Time | Speaker | Topic |
|---|---|---|
| 09.00 to 09.10 | Alen Sabu | Overview of the tutorial |
| 09.10 to 10.30 | Harish Patil | Tools from Intel: Pin, PinPlay, SDE, ELFies |
| 10.30 to 10.45 | | Break |
| 10.45 to 11.30 | Akanksha Chaudhari | Simulation and Single-threaded Sampling |
| 11.30 to 11.40 | | Break |
| 11.40 to 12.20 | Alen Sabu | Multi-threaded Sampling and LoopPoint |
| 12.20 to 13.00 | Changxi Liu | Running Sniper and LoopPoint Tools |

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

Alen Sabu[1], Changxi Liu[1], Akanksha Chaudhari[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Performance Analysis of Systems and Software, May 22nd 2022, Singapore

Session 2

# Simulation and Sampling

AKANKSHA CHAUDHARI, RESEARCH ASSISTANT
NATIONAL UNIVERSITY OF SINGAPORE

# Architectural Trends in Processor Design

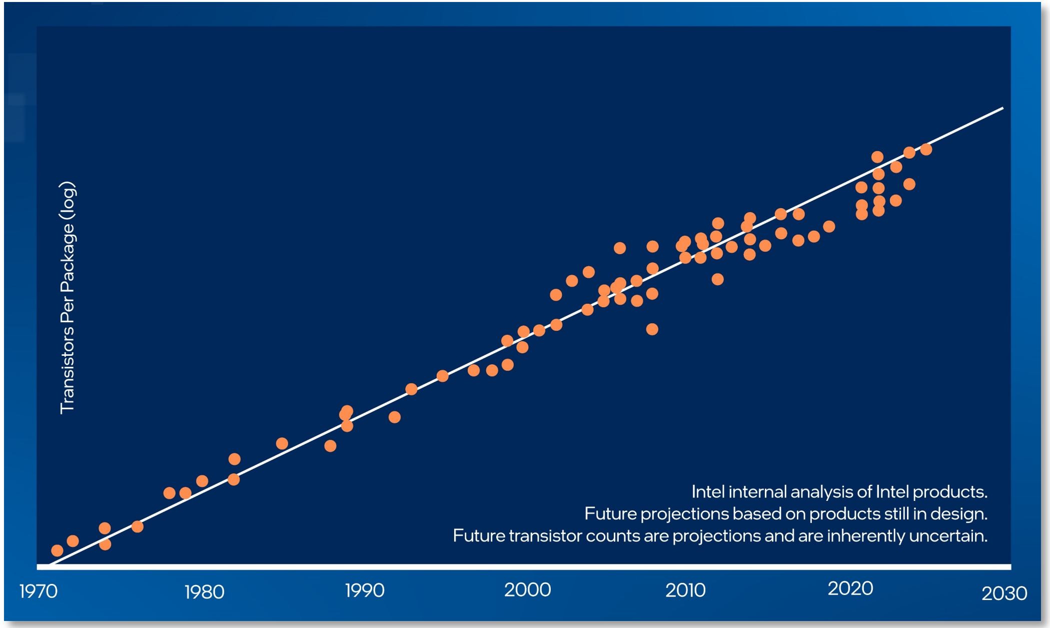



Fig. 1: Moore Law number of transistor per device: past, present, future [Intel]

- 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

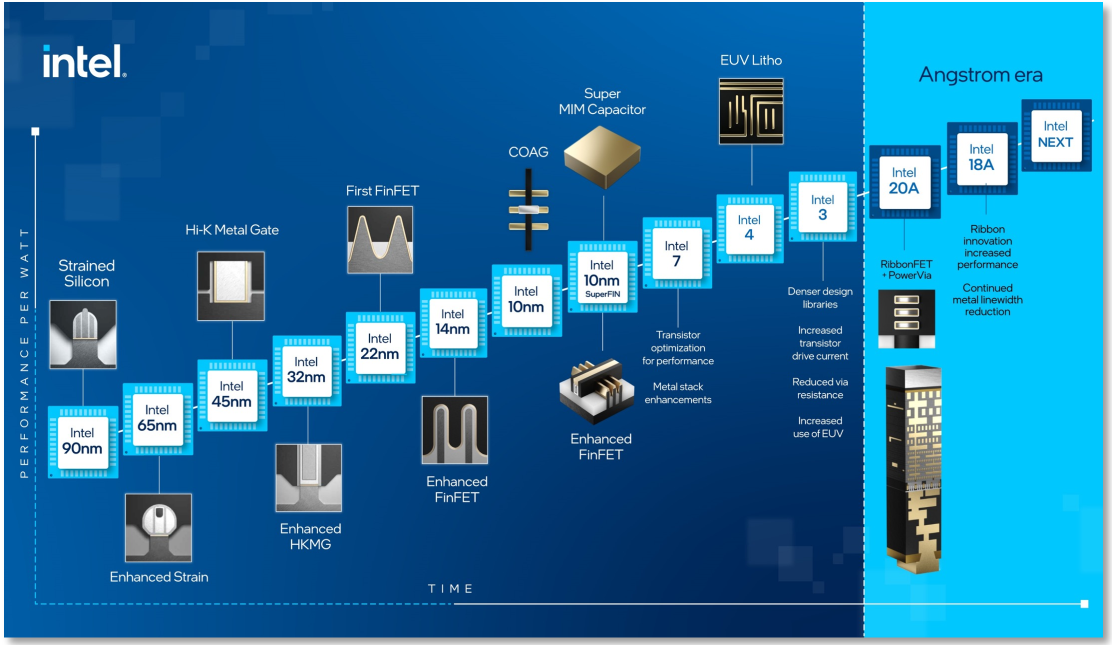



Fig. 2: Transistor innovations over time

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



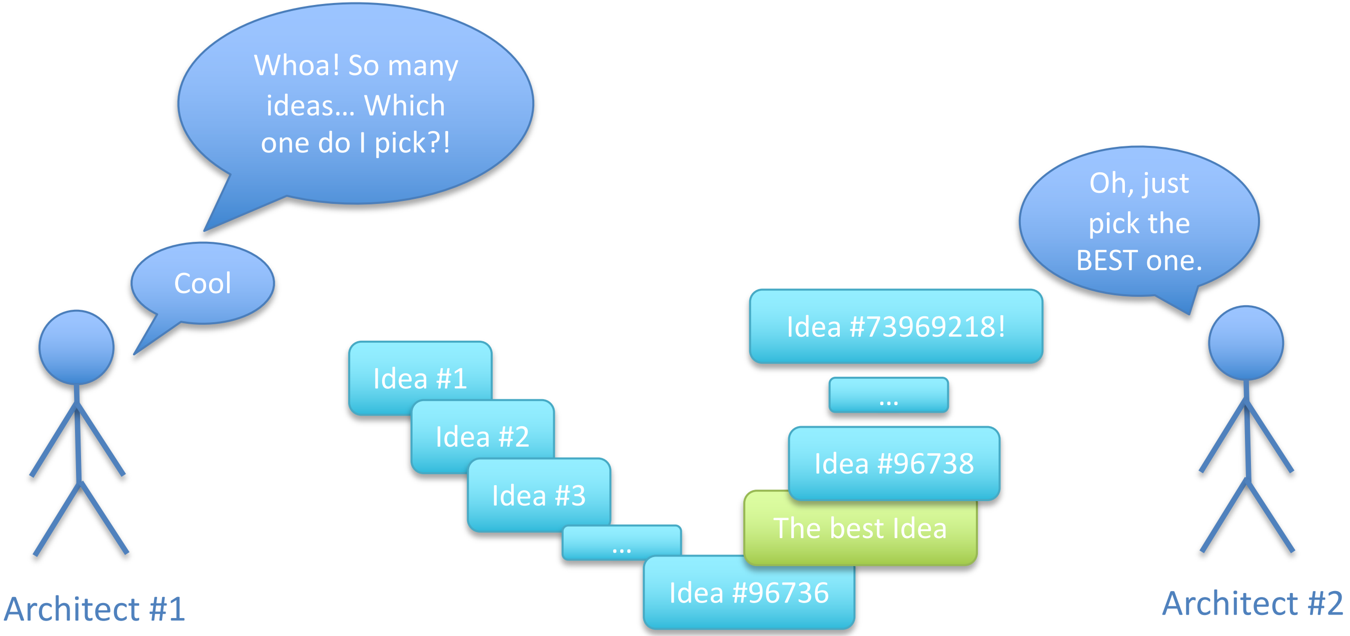
Source: https://www.intel.com/

NUS National University of Singapore intel

# Exploration and Evaluation of New Ideas

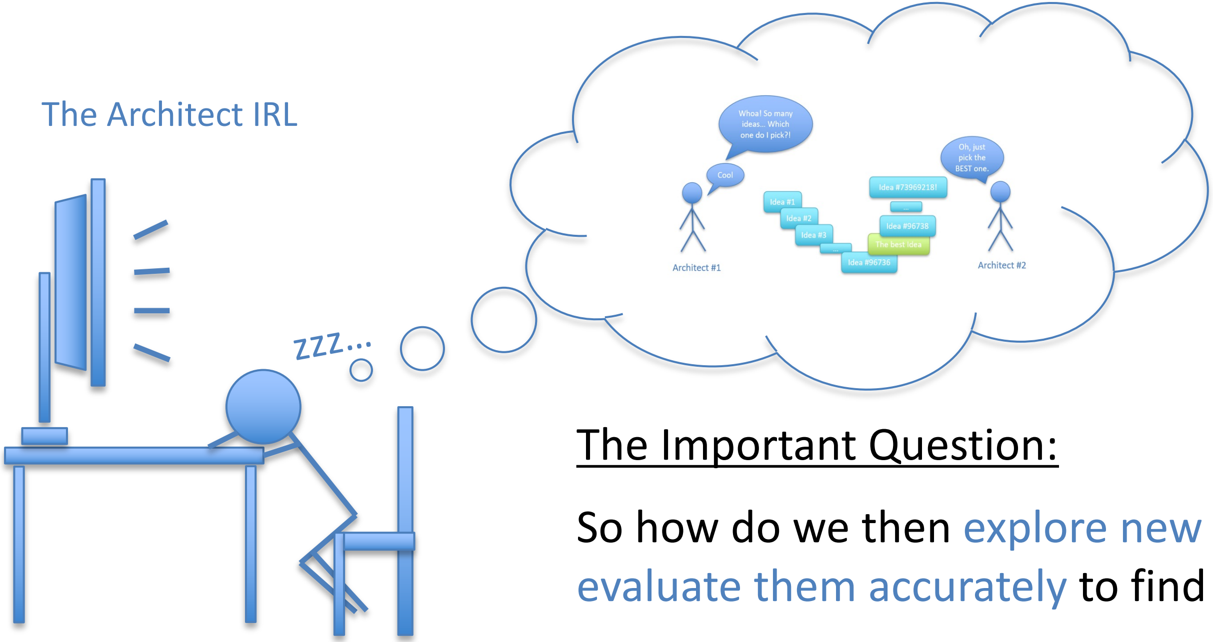
- Architecture is rapidly evolving domain with a lot of new research directions.
- A plethora of design choices are available:
  - Ranging from the choice of components, the choice of operating modes of each component, the choice of interconnects used, the choice of algorithms employed, etc.
- The process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

# Exploration and Evaluation of New Ideas

- Important questions when considering any idea:
  - Does it work?
  - How well does it work?
- Generally speaking, good idea optimizes a finite set of performance metrics, say M.

Let M = $\{m_1, m_2, \ldots m_n\}$,

where $m_i \in$ M can be computational speed, energy efficiency, memory utilization etc.

# Exploration and Evaluation of New Ideas

A variety of different evaluation methods are available:

- Theoretical proofs

  Formally proving the correctness/efficiency of the proposed design using computational theory and mathematical logic.

- Analytical modeling

  Mathematically modeling the proposed design at some level of abstraction to analyze/quantify its performance.

# Exploration and Evaluation of New Ideas

A variety of different evaluation methods are available:

- Simulation (at varying degrees of abstraction and accuracy)

  A model that mimics the system behavior demonstrating its key functions and operations accurately.

- Prototyping using existing systems

  Implementing a draft version of the proposed design using existing systems (such as FPGAs) to evaluate the performance.

- Actual implementation

# Exploration and Evaluation of New Ideas

An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Theoretical proofs / accurate modeling of practical systems can be extremely complex

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Theoretical proofs / accurate modeling of practical systems can be extremely complex

Difficult to evaluate a design for all possible workload profiles

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Theoretical proofs / accurate modeling of practical systems can be extremely complex

Difficult to evaluate a design for all possible workload profiles

Worst-case estimates can be misleading

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Note that, using analytical modeling in conjunction with simulation can provide significant quantifiable benefits

# Exploration and Evaluation of New Ideas

An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping

  Relatively Expensive
- Actual implementation

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Relatively Expensive

Limited by the capability of the systems used to model

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Most expensive

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

Most expensive

Not always feasible to implement and evaluate each idea
Especially if we have too many options to choose from

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation

# Exploration and Evaluation of New Ideas

An "evaluation" of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation → The most feasible way to explore and evaluate large-scale, complex architectural designs in terms of time, cost and efficiency!
- Prototyping
- Actual implementation

# Simulation: An Overview

- Simulation enables the modeling of new research ideas at varying degrees of abstraction and accuracy.
- Main goals:
  - Enables fast exploration of design space (to discover the next big idea!).
  - Evaluation of new research ideas by estimating their relative performances.
  - Evaluation, debugging and understanding the behavior of existing systems.
- How does a simulator work?
  - Mimics system behavior to reflect its performance in terms of the metric of interest (ex: Instructions per cycle, Runtime, etc).

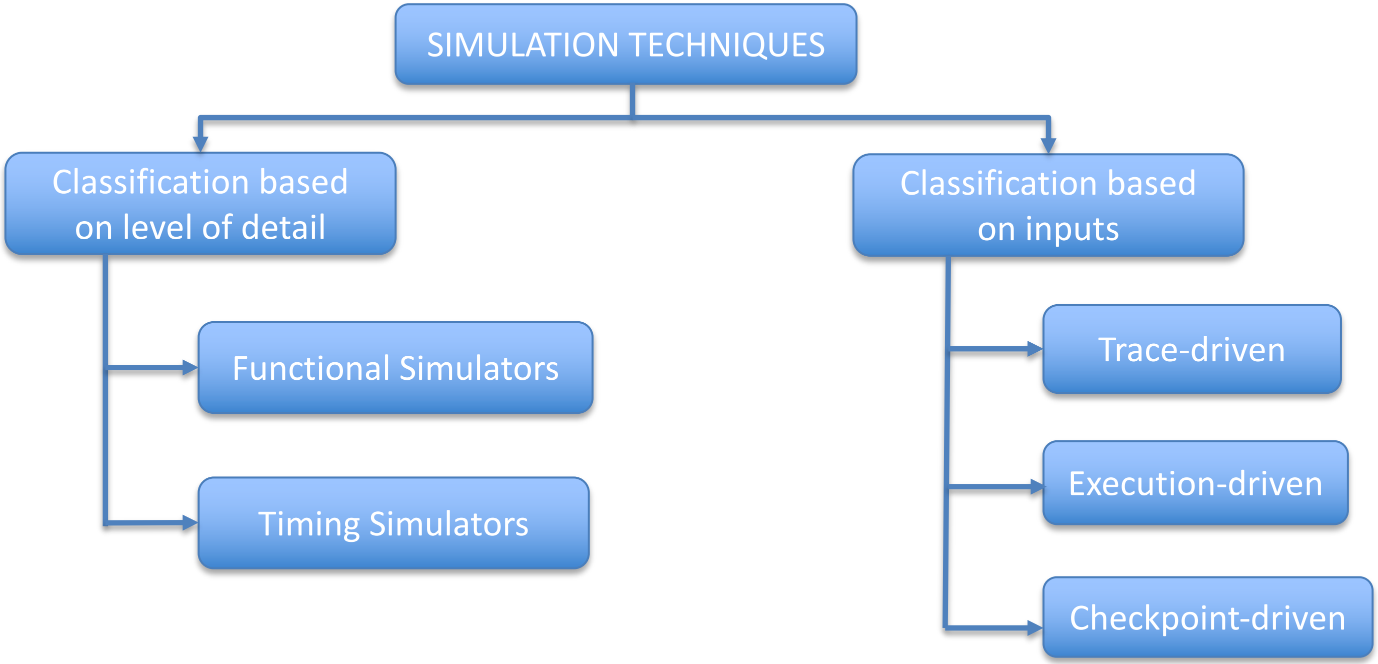
# Simulation: An Overview

- Caution: Inaccurate simulation → Inaccurate evaluation/results → Wrong conclusions.
  - Ex: Inaccurate assumptions, inaccurate extrapolation of performance, etc.
- Very important to select the right simulation technique!

# Simulation: An Overview

- An ideal simulation technique:
  - High speed: For faster exploration.
  - High flexibility: For wider exploration.
  - High accuracy/low simulation error: For accurate evaluation.
- Practical simulation techniques → tradeoffs:
  - Speed vs. accuracy
  - Speed vs. flexibility
  - Flexibility vs. accuracy

# Different Simulation Techniques



# Techniques to Simulate Faster

- Partially simulating to extrapolate performance:
  - 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

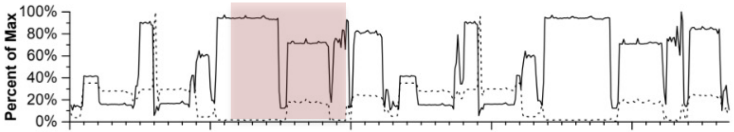
- Workload reduction
  - Simulating for reduced input sets
  - Simulating for reduced loop counts in workloads

# Techniques to Simulate Faster

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

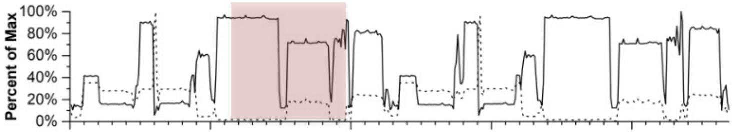
# Techniques to Simulate Faster

- 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.

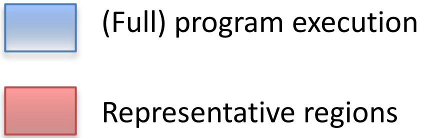



  - [Workload reduction] → benchmark behavior varies significantly across test, train and reference inputs → do not reflect the actual performance.

# Sampled Simulation to the Rescue!

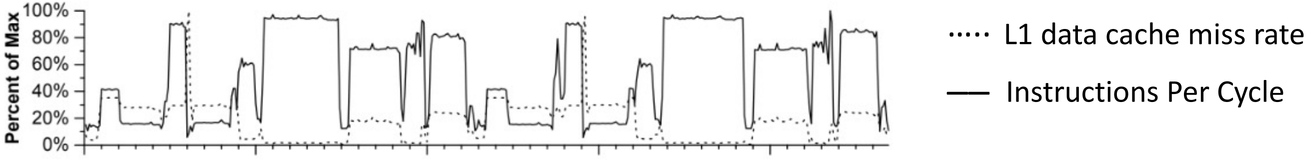
- Sampling enables the simulation selective representative regions of an application.
  - "Representative regions" refer to the subset of regions in the application that reflect the behavior of the entire system when extrapolated.
- How to select these "representative regions"?
  - Targeted sampling (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

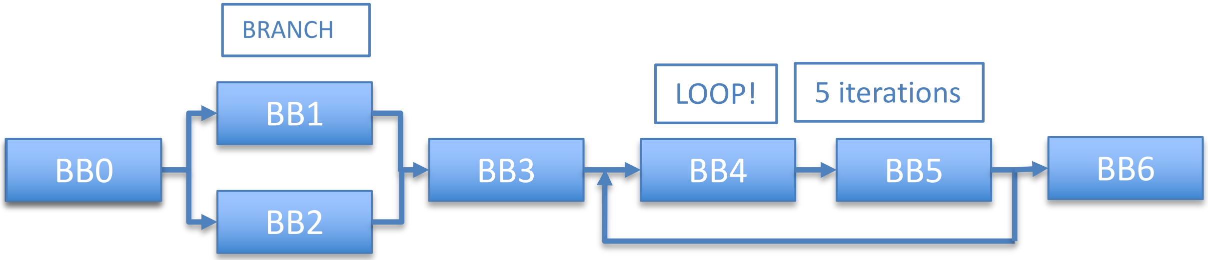
- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
    - Creating a Basic Block Similarity Matrix
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

# Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
    - Creating a Basic Block Similarity Matrix
  - 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 run in a given interval during the program execution.




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

Indexed by Basic Block IDs

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

Maintains the execution count for each Basic Block

# Sampled Simulation Techniques: SimPoint

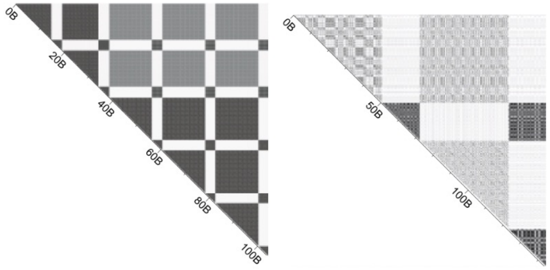
- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
    - Creating a Basic Block Similarity Matrix
  - STEP 2: Clustering of Basic Block Vectors
    - Random Projection
    - K-means Clustering
  - STEP 3: Identifying representative regions

# Sampled Simulation Techniques: SimPoint

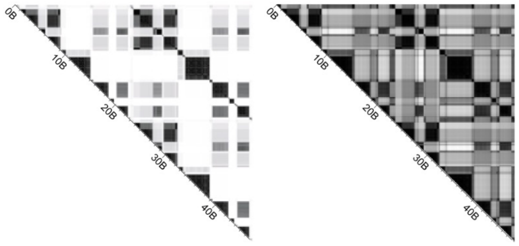
- Basic Block Similarity: Measured using Euclidean or Manhattan Distances.

$$EuclideanDist(a,b) = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} \qquad ManhattanDist(a,b) = \sum_{i=1}^{D} |a_i - b_i|$$

- Depicted by Basic Block Similarity Matrices.




Using Manhattan distances




Using Euclidian distances

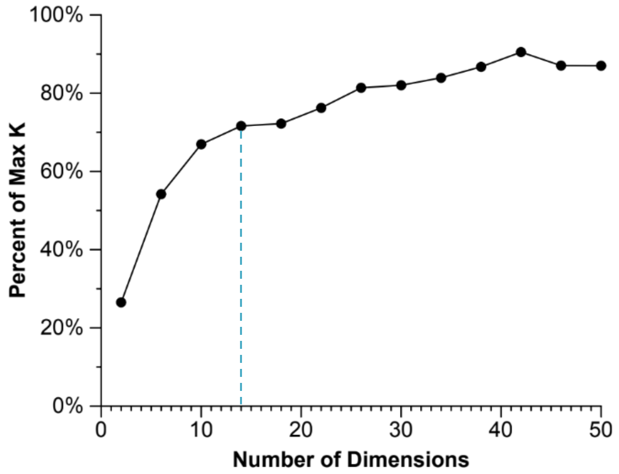
- The program execution progresses along the diagonal of the matrix.
- Point at (x, y) gives similarity index between $BBV_x$ and $BBV_y$.
- ↑ darkness → ↑ similarity

# Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
    - Creating a Basic Block Similarity Matrix
  - 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.



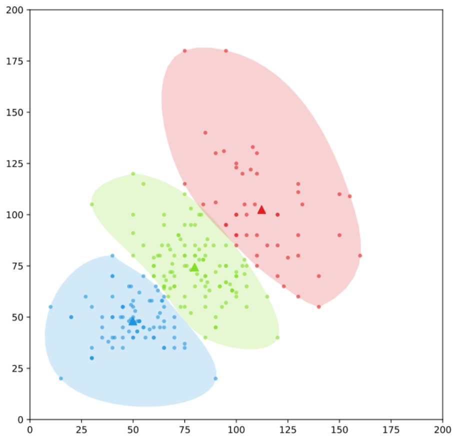
# Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
    - Creating a Basic Block Similarity Matrix
  - 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.



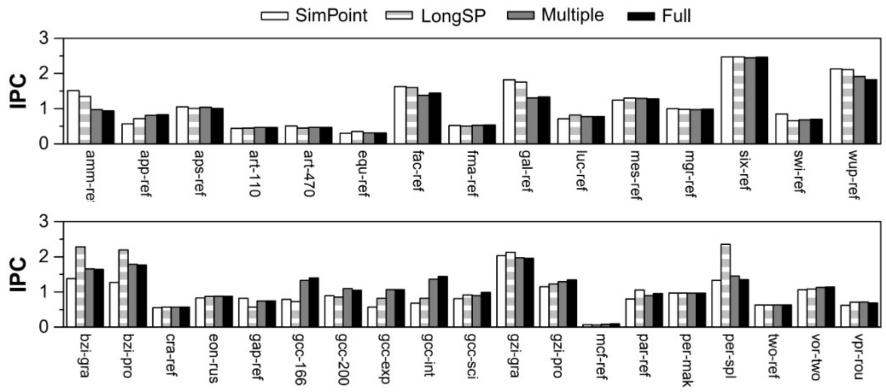

*Choosing k: The clustering that achieves a BIC score that is at least 90% of the spread between the largest and smallest BIC score is chosen.*

# Sampled Simulation Techniques: SimPoint

- How SimPoint works:
  - STEP 1: Basic block profiling
    - Generating the Basic Block Vectors
    - Creating a Basic Block Similarity Matrix
  - 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

- STEP 1: Determine n based on the required confidence (assuming the coefficient of variation $V_x$) using the following equation:

$$confidence\ interval = \mp \left[\frac{z.V_x}{\sqrt{n}}\right].X$$

(where $X$ is the mean, and $z$ = 100 (1 - α/2) is the percentile of the standard normal distribution)

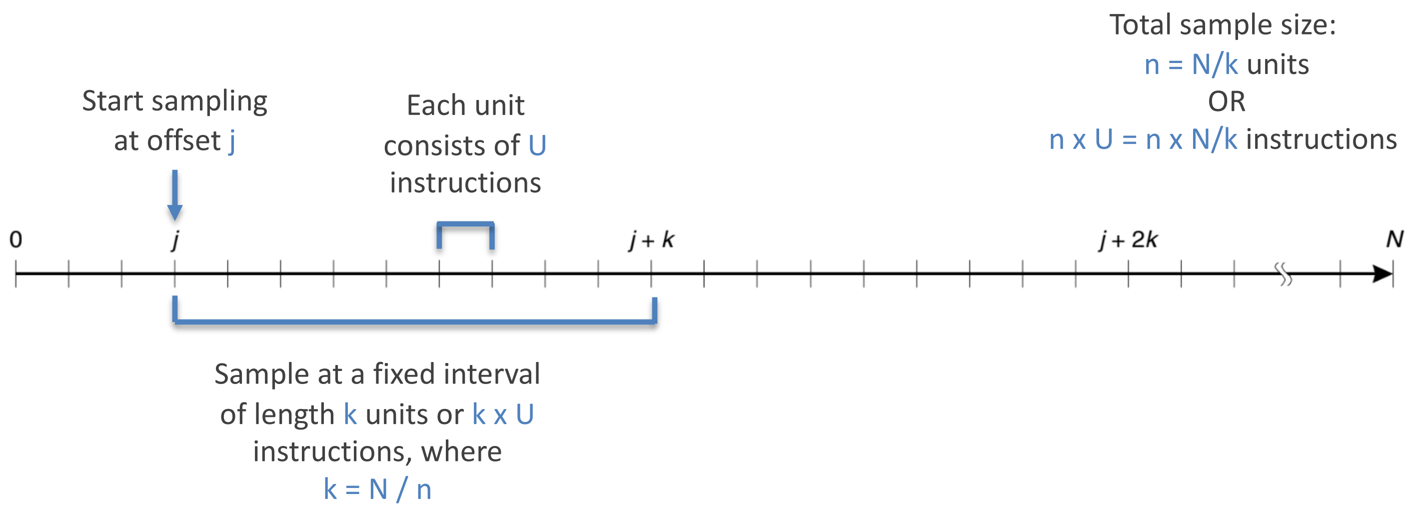
- STEP 2: If the initial sample does not achieve the desired confidence, compute n using the equation:

$$n \geq \left[\left(\frac{z.V_x{'}}{\varepsilon}\right)^2\right]$$

(where $V_x$' is the coefficient of variation for the obtained sample)

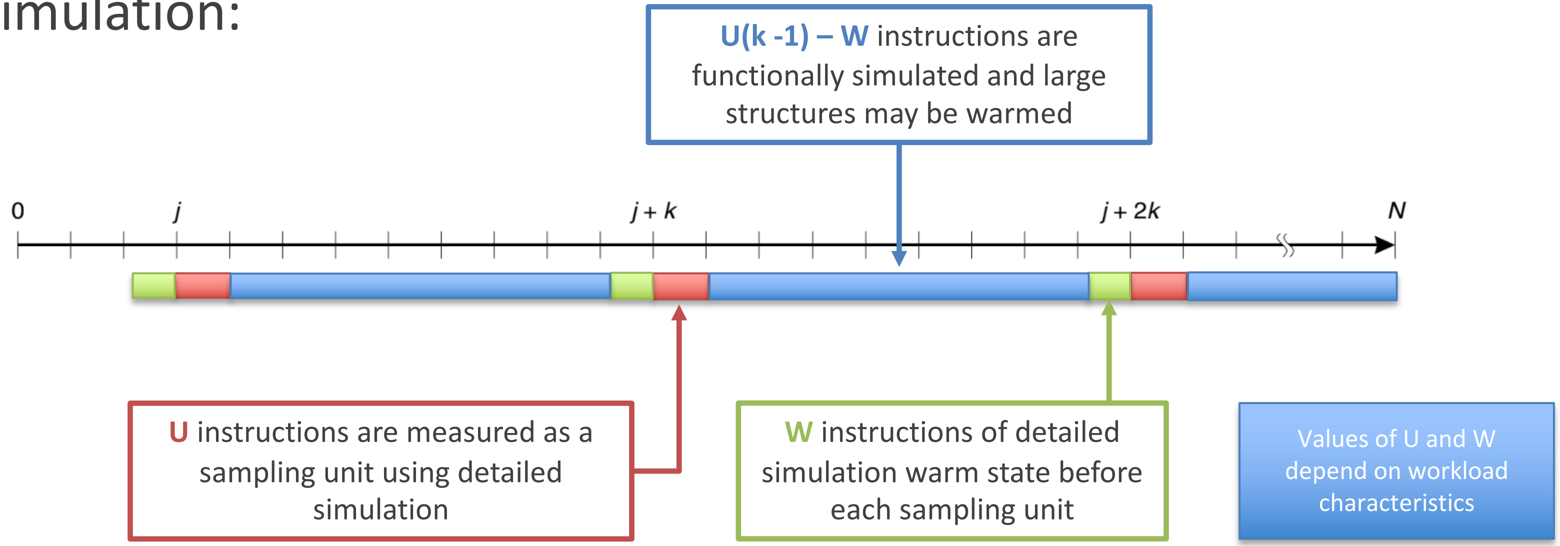
# Sampled Simulation Techniques: SMARTS

- SMARTS uses Systematic Sampling:



# Sampled Simulation Techniques: SMARTS

- Simulation:



# 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

# Agenda

| Time | Speaker | Topic |
|---|---|---|
| 09.00 to 09.10 | Alen Sabu | Overview of the tutorial |
| 09.10 to 10.30 | Harish Patil | Tools from Intel: Pin, PinPlay, SDE, ELFies |
| 10.30 to 10.45 | Break | |
| 10.45 to 11.30 | Akanksha Chaudhari | Simulation and Single-threaded Sampling |
| 11.30 to 11.40 | Break | |
| 11.40 to 12.20 | Alen Sabu | Multi-threaded Sampling and LoopPoint |
| 12.20 to 13.00 | Changxi Liu | Running Sniper and LoopPoint Tools |

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

Alen Sabu[1], Changxi Liu[1], Akanksha Chaudhari[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Performance Analysis of Systems and Software, May 22nd 2022, Singapore

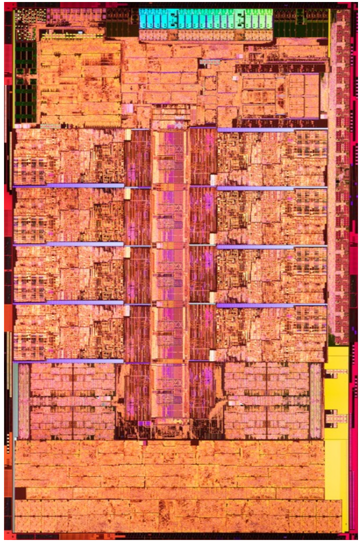
Session 3

# The LoopPoint Methodology

ALEN SABU, PHD CANDIDATE
NATIONAL UNIVERSITY OF SINGAPORE

# Simulation in the Post-Dennard Era

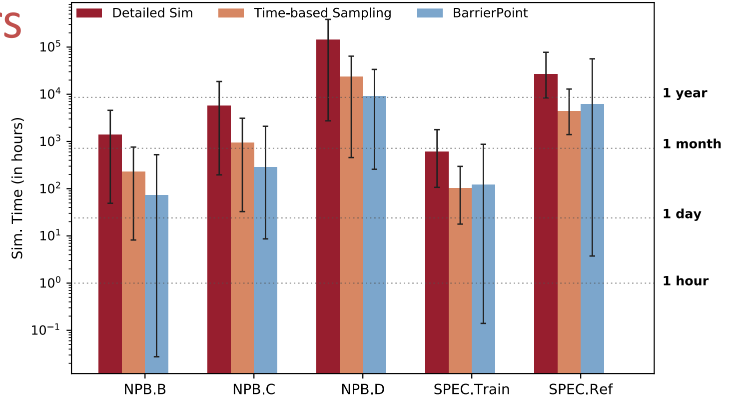
- Modern architectures require smarter simulators
- Microarchitectural simulation is slow
  - NPB (D), SPEC CPU2017 (ref) can take years
  - Solution – Simulate representative sample



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

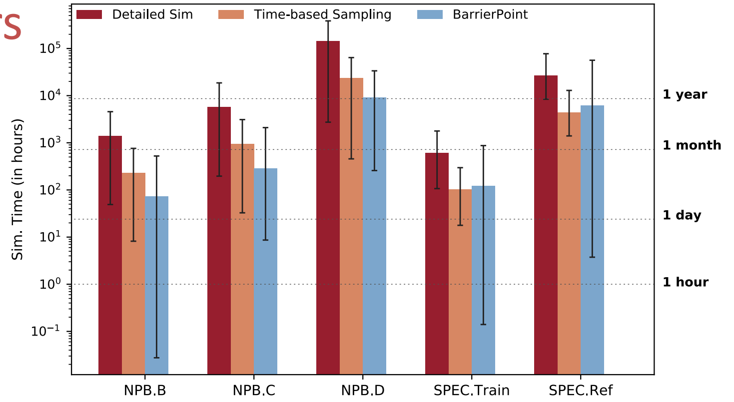



Benchmarks with *8* threads, *static* schedule, *passive* wait-policy, simulated at *100 KIPS*.

# 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**

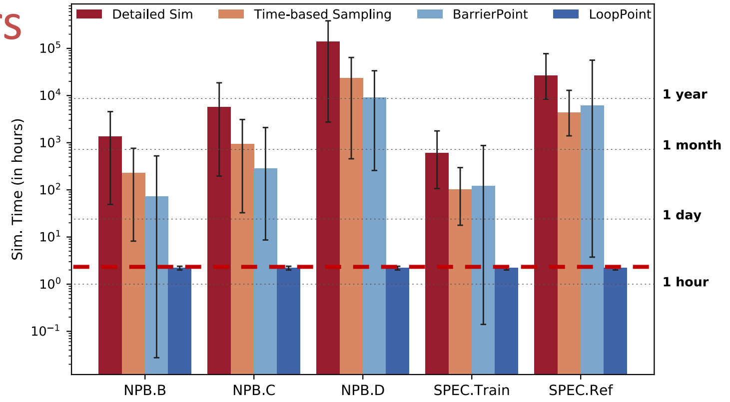



Benchmarks with *8* threads, *static* schedule, *passive* wait-policy, simulated at *100 KIPS*.

# 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*.

# 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

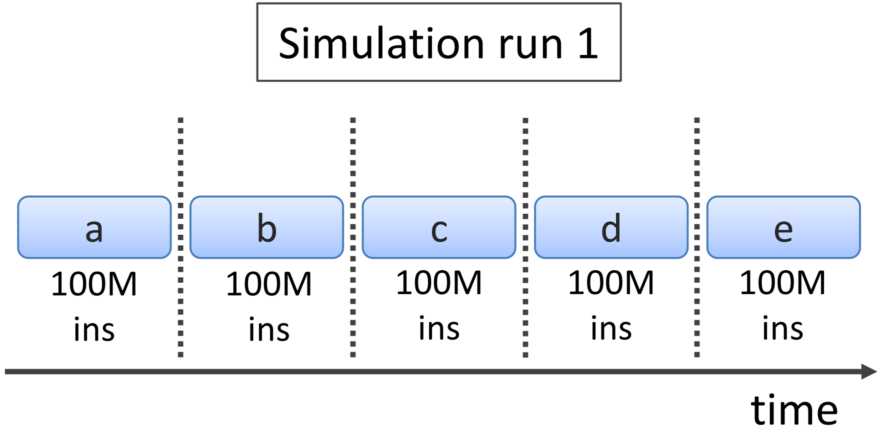
Differentiating thread waiting from real work



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

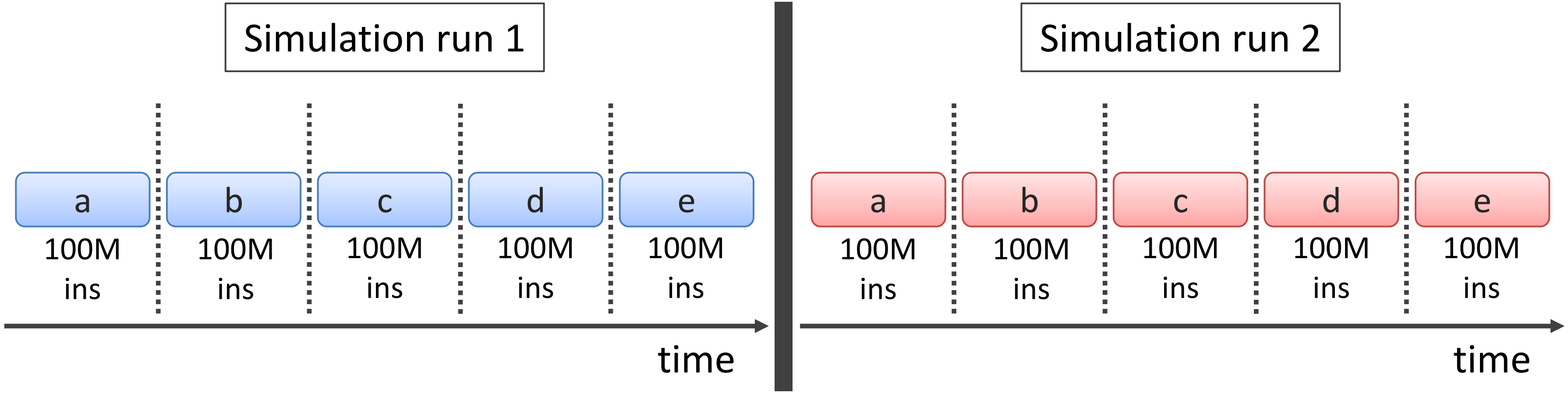
# Extending Single-threaded Techniques

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



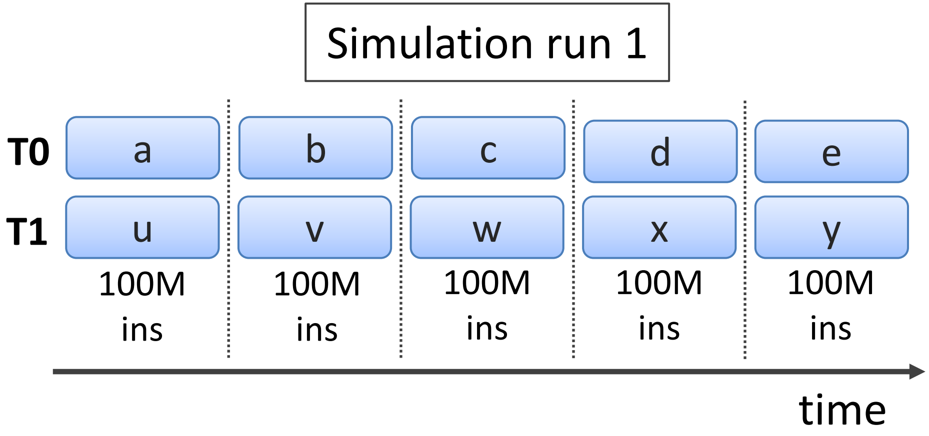
# Extending Single-threaded Techniques

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



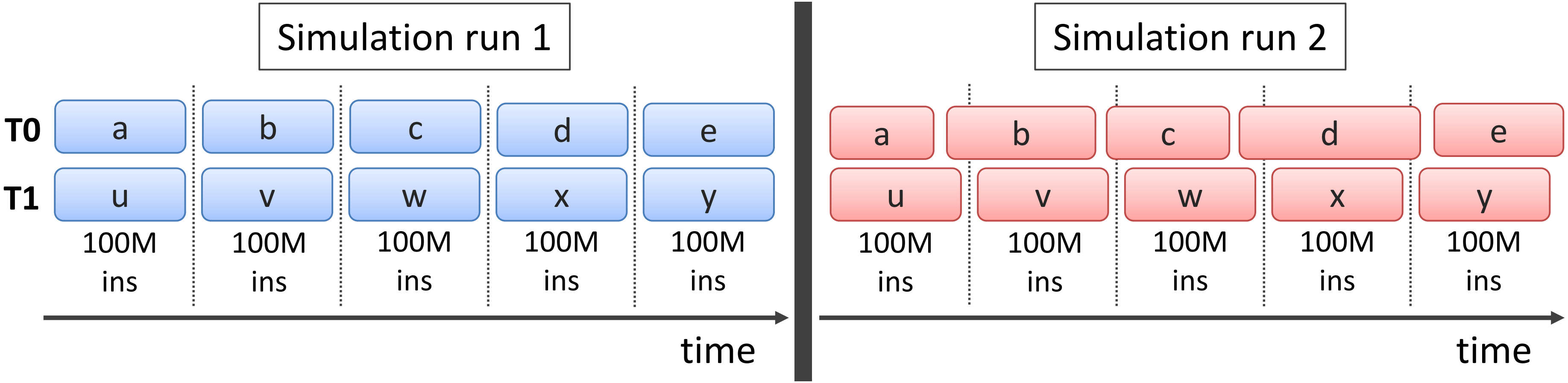
# Extending Single-threaded Techniques

- 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



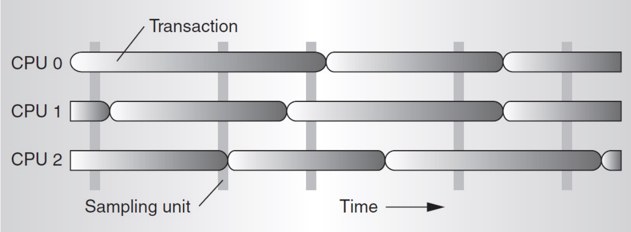



NUS National University of Singapore intel

# Multi-threaded Sampling

## FlexPoints

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



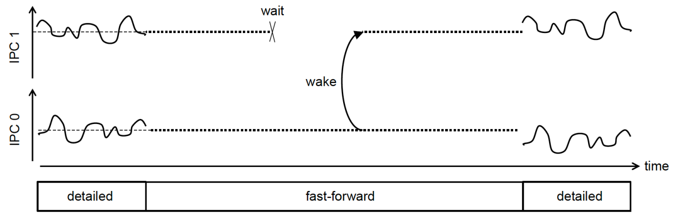



Wenisch et al., “SimFlex: statistical sampling of computer system simulation”, IEEE Micro’06

# Multi-threaded Sampling

## Time-based Sampling

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



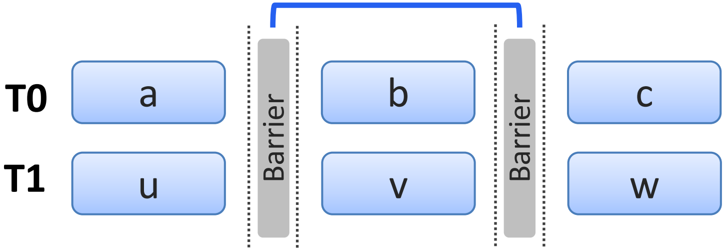


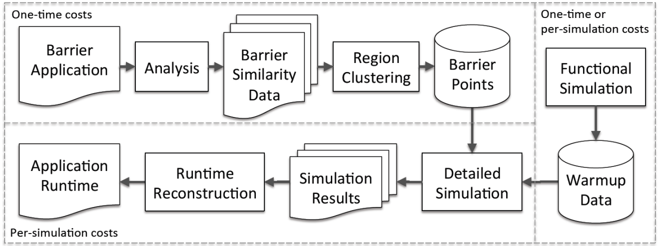

Carlson et al., “Sampled Simulation of Multithreaded Applications”, ISPASS’13
Ardestani et al., "ESESC: A fast multicore simulator using time-based sampling." HPCA, 2013

# Multi-threaded Sampling

## 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

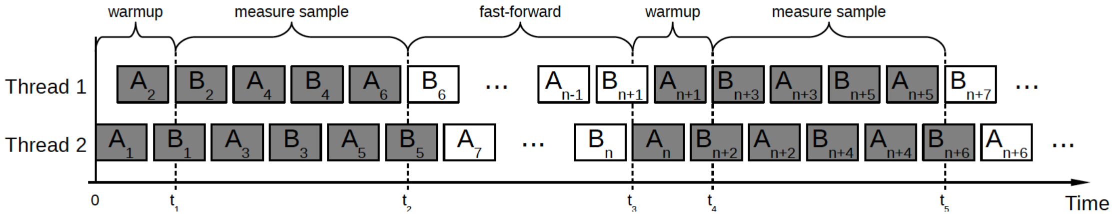
NUS National University of Singapore intel

# Multi-threaded Sampling

## 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 | | | Multiprocessor Sampling | | |
|---|---|---|---|---|---|
| SimPoint[1] SMARTS[2] | → | Instruction count | Flex Points[3] | → | Instruction count |
| Time-based sampling[4] | → | Time | 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

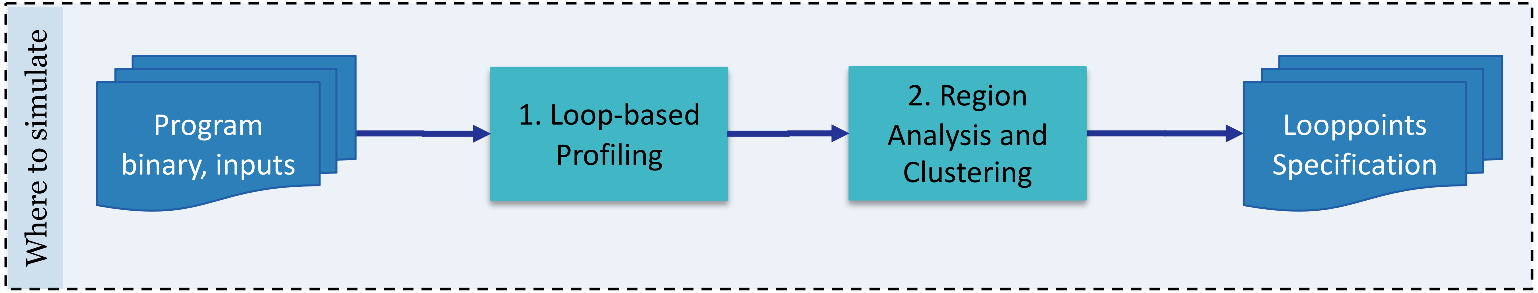
Where to simulate

How to simulate

# Overall Methodology

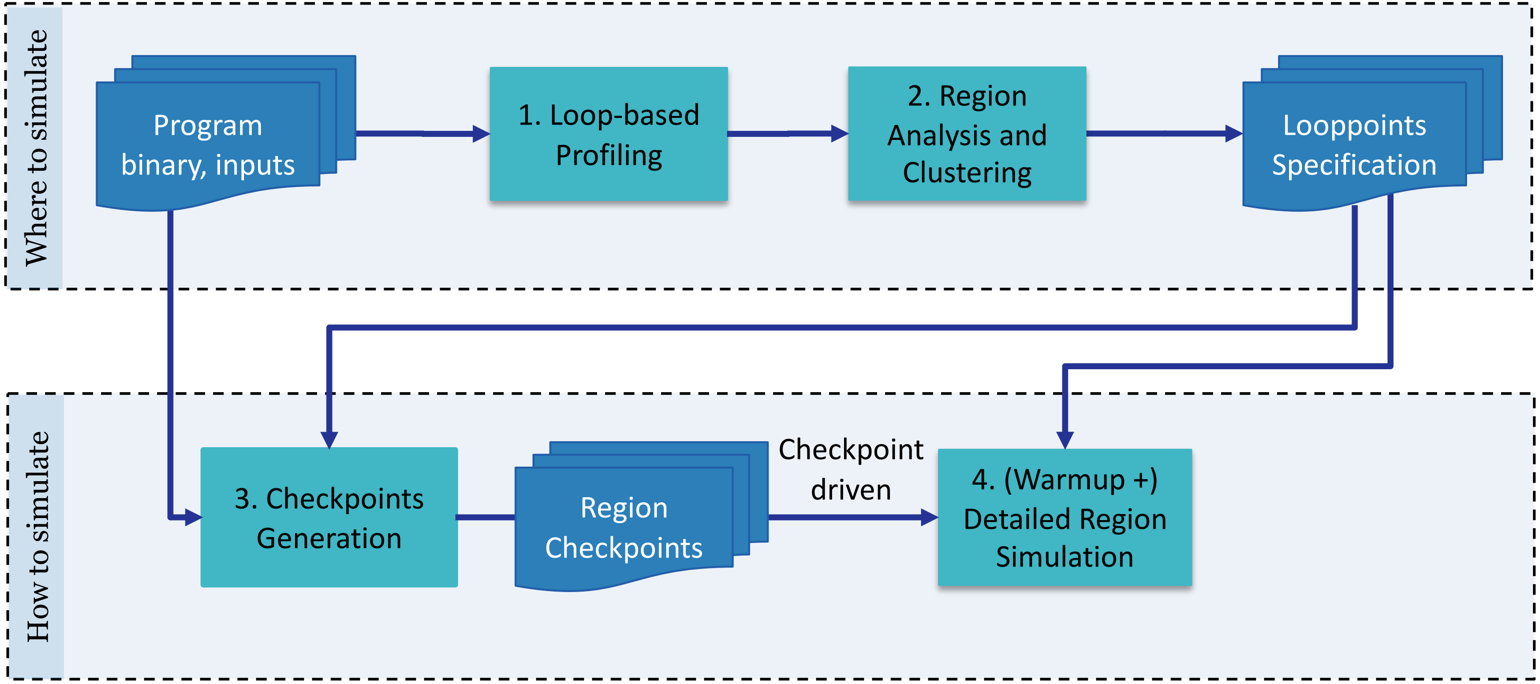

How to simulate

# Overall Methodology

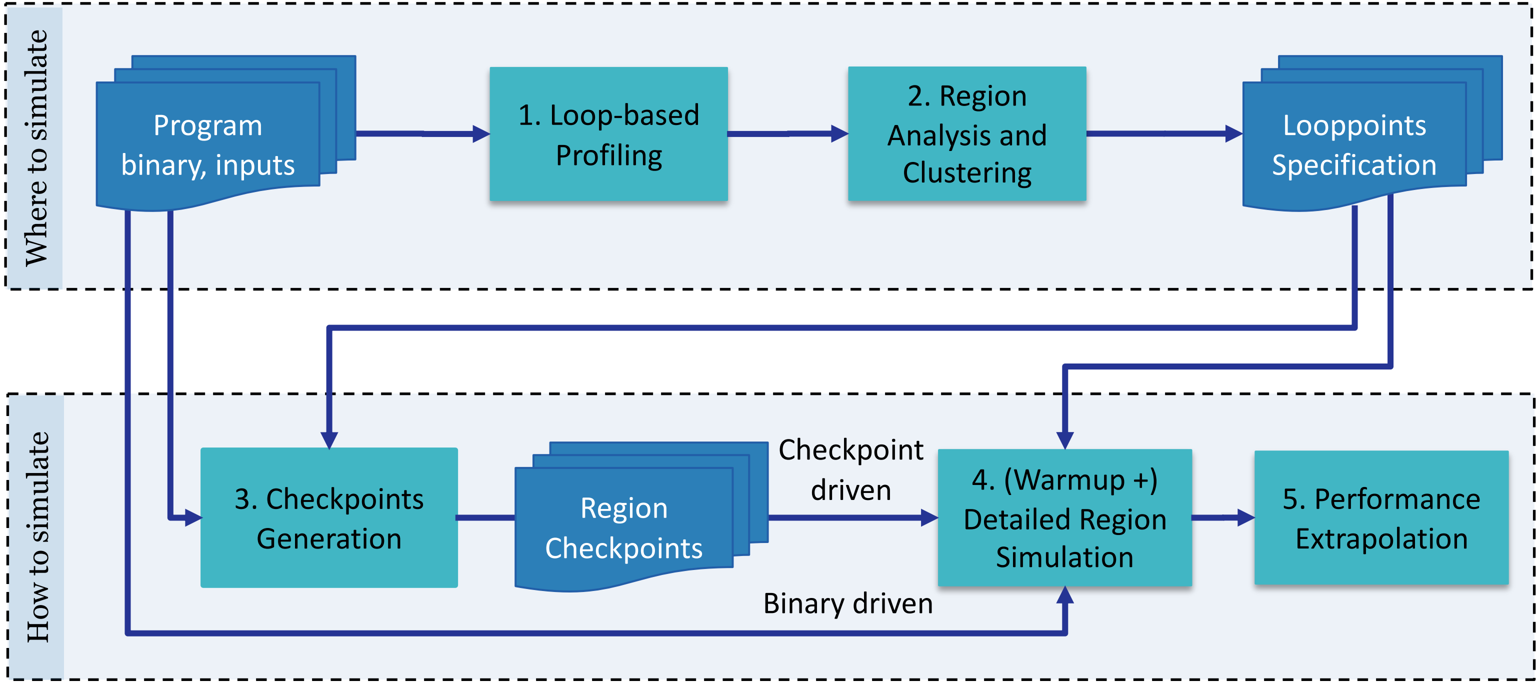



How to simulate

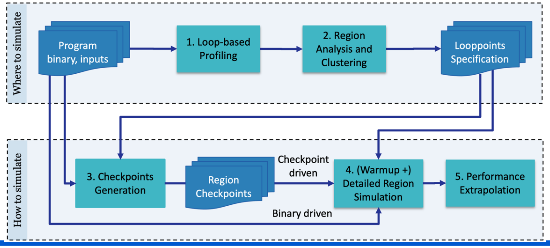
# Overall Methodology



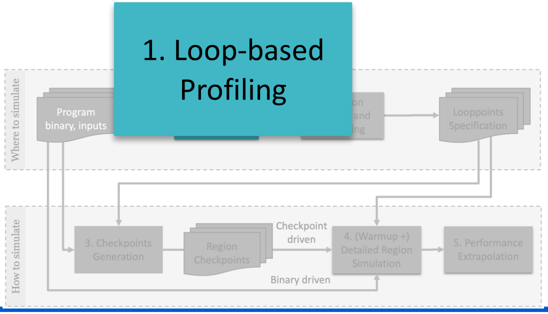
# Overall Methodology



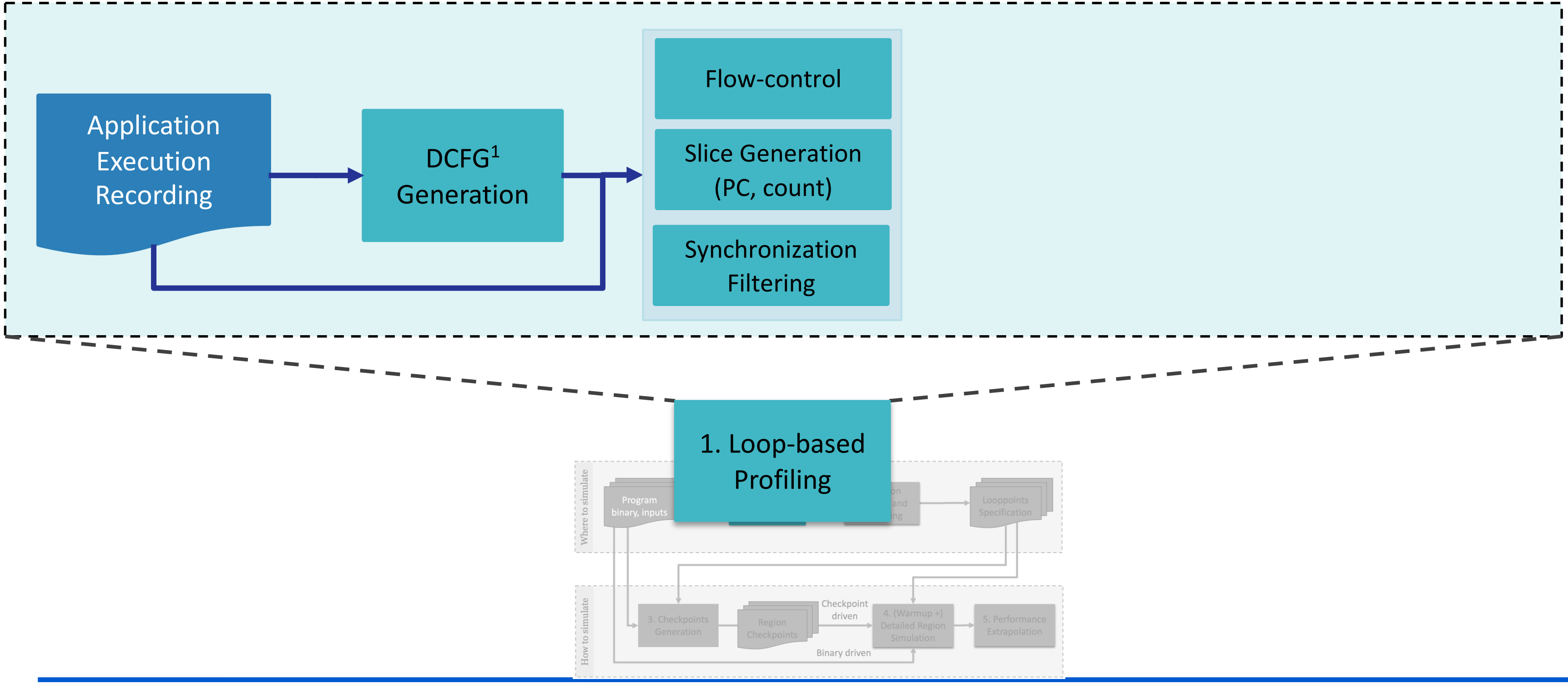
# Loop-based Profiling



# Loop-based Profiling

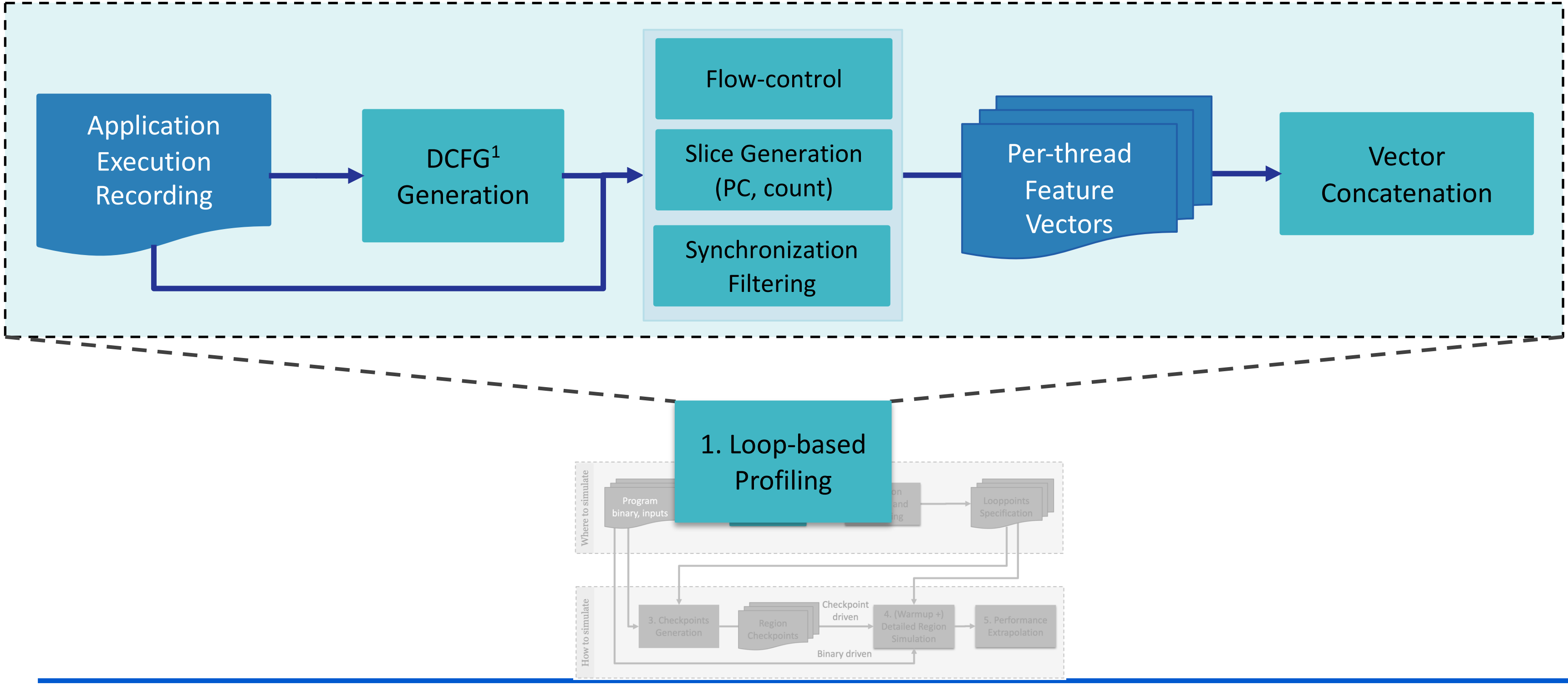


# Loop-based Profiling




DCFG: Dynamic Control-Flow Graph

# Loop-based Profiling




DCFG: Dynamic Control-Flow Graph

# Loop-based Profiling

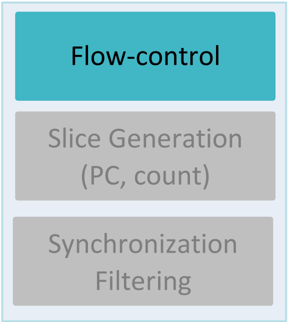
Flow-control

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



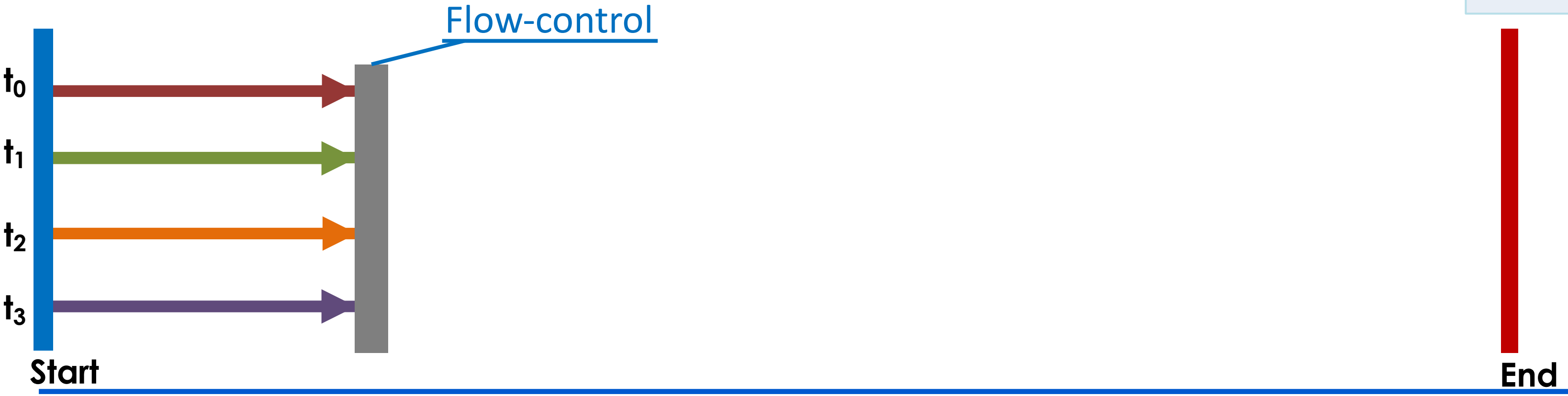
# Loop-based Profiling: Flow-control

- 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

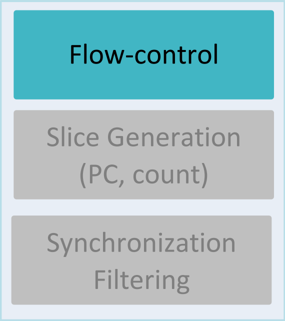
Slice Generation (PC, count)

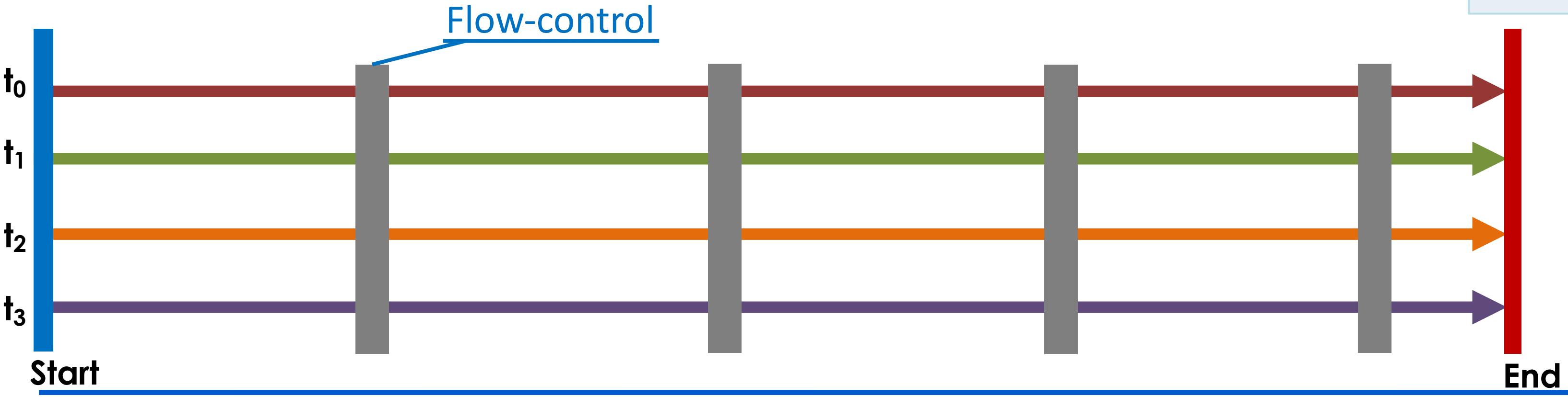
Synchronization Filtering



# Loop-based Profiling: Flow-control

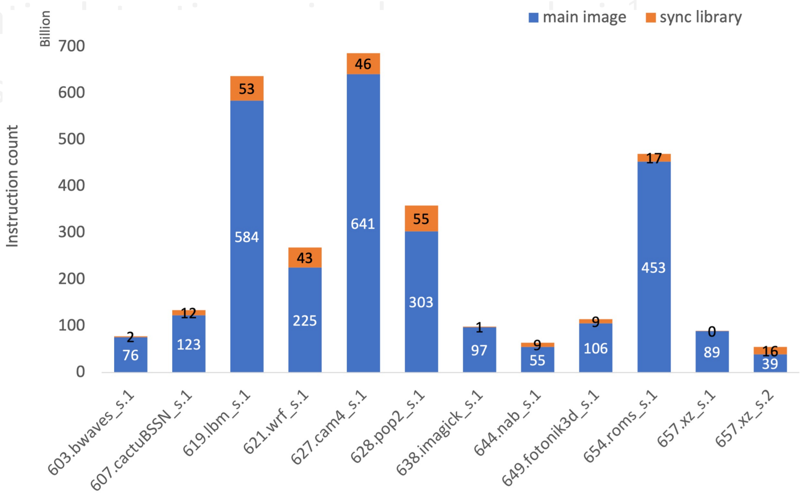
- Load Imbalance can affect profiling
  - Make sure threads make equal forward progress
- Implementation: Control the forward progress of threads
  - Synchronize threads (barriers) externally at regular intervals
  - Make sure all threads execute similar number of instructions



# 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

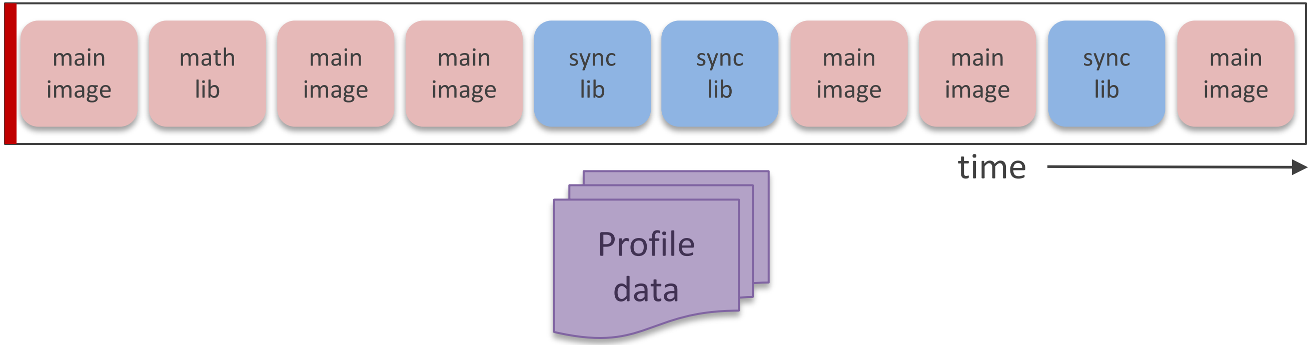
Ignore sync library code (Ex. `libiomp5.so`, `libpthread.so`)

Flow-control

Slice Generation (PC, count)

**Synchronization Filtering**

Application execution

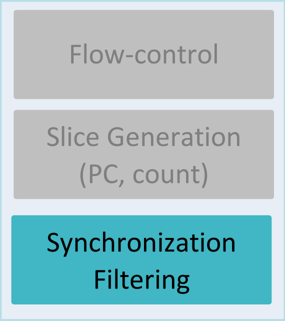


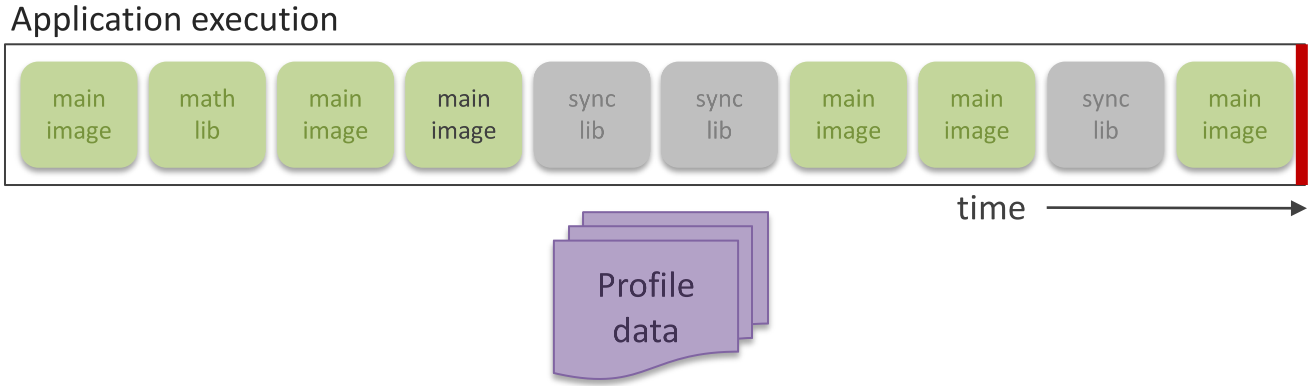



[1]Li et al., "Spin detection hardware for improved management of multithreaded systems," TPDS, 2006

# Loop-based Profiling: Sync Filtering

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






[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

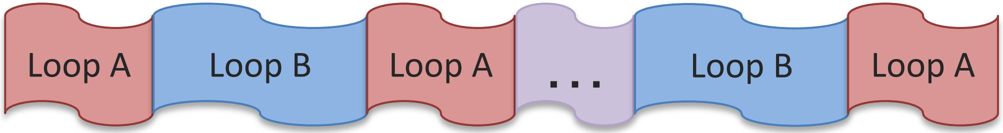
# Loop-based Profiling: Slice Generation

- 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)**

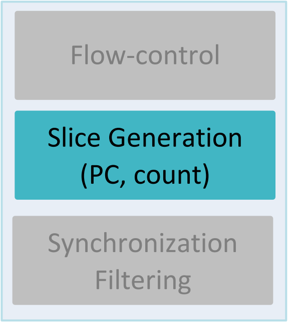
Synchronization Filtering

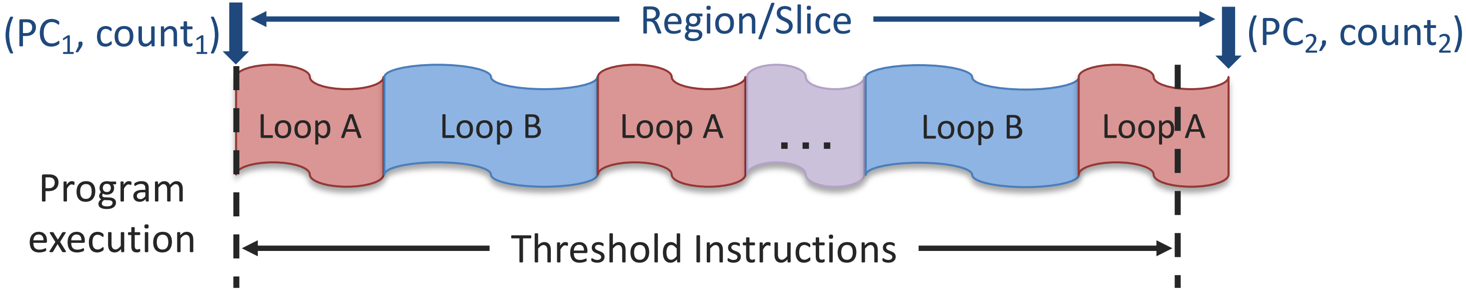



Program execution

# 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***



# 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*

**ID: A B C**

| 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 |
| … | … | |



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

NUS National University of Singapore intel

# 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

NUS National University of Singapore intel

# Loop-based Profiling: Slice Generation

- Basic Block (BB)
  - Section of code with single entry and exit
- Basic Block Vector (BBV)
  - Exec
  - 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 |
| … | … | |

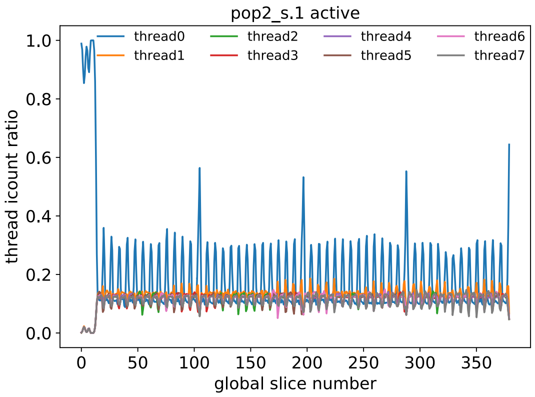
```
                ID:   A   B   C
     BB Exec Count: < 1, 20, 0, …>
weigh by Block Size: < 8,  3, 1, …>
       BB Exec Wt: < 8, 60, 0, …>
```

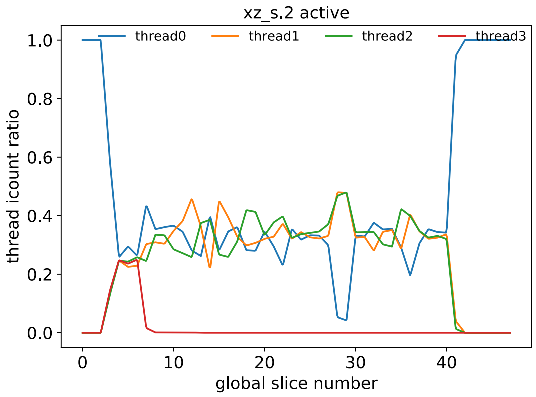


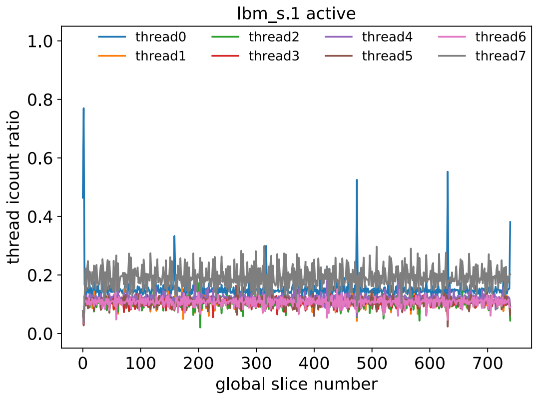
Image source: Sherwood et.al., "Automatically Characterizing Large Scale Program Behavior", ASPLOS 2002

# 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... BBV ... Global BBV

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
- ***Global-BBV*s**: Concatenate per-thre

**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... BBV ... Global BBV

Thread 1

| BB | Example Assembly Code |
|---|---|
| N | srl a2, 0x8, t4 ... |
| P | b... |
| … | |
| Z | s... |

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 |

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

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

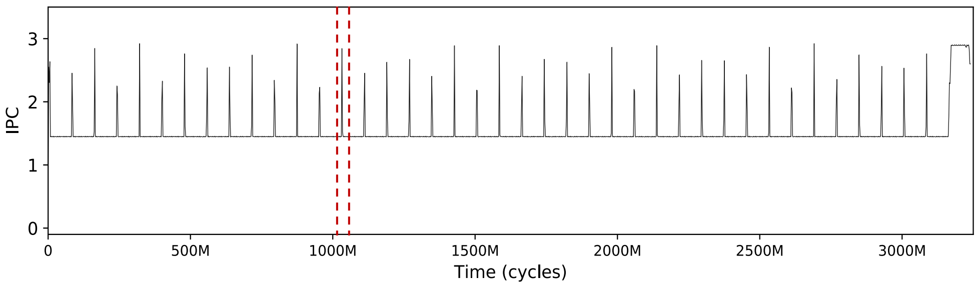
Global-BBV

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

# 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 */
        ……
```

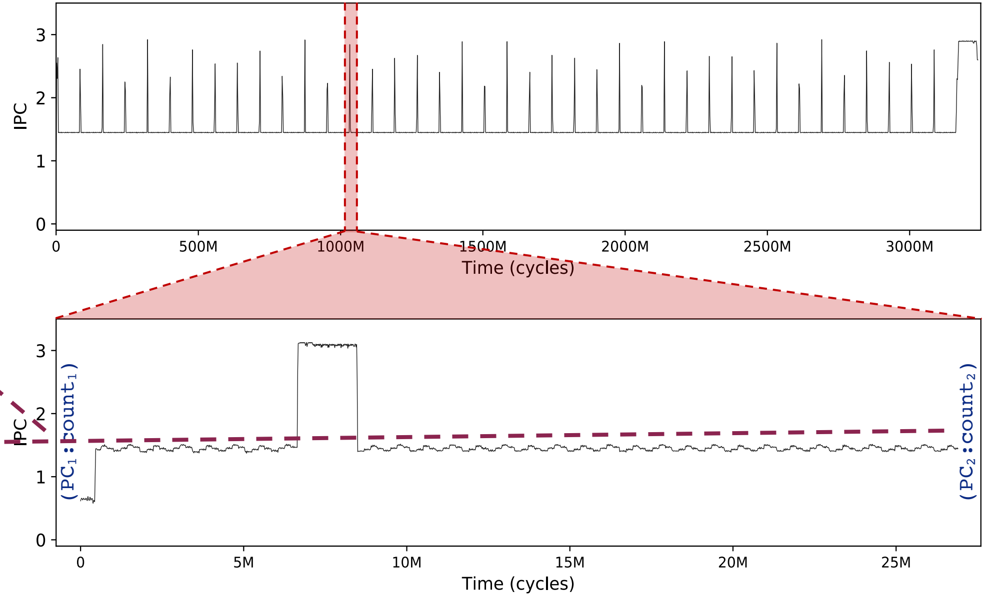



638.imagick_s, train input, 8 threads

# A LoopPoint Region

**638.imagick_s/magick/morphology.c**

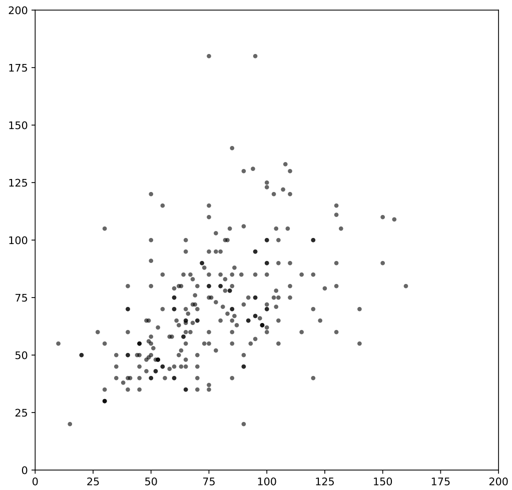
```
2842 #if defined(MAGICKCORE_OPENMP_SUPPORT)
2843   #pragma omp parallel for schedule(static,4) shared(progress,status) \
2844     magick_threads(image,result_image,image->rows,1)
2845 #endif
2846   for (y=0; y < (ssize_t) image->rows; y++)
2847   {
        ……
2886      for (x=0; x < (ssize_t) image->columns; 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

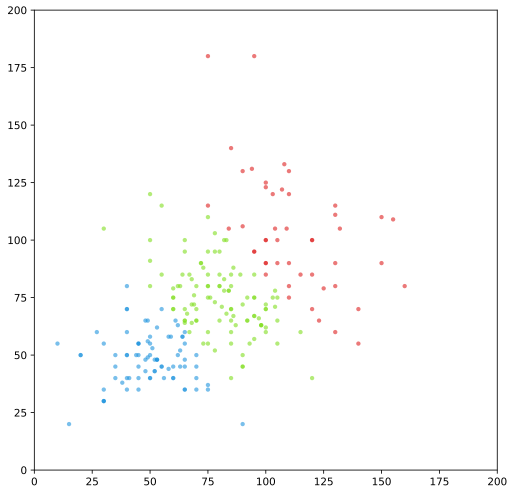
# Identifying Simulation Regions

- 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



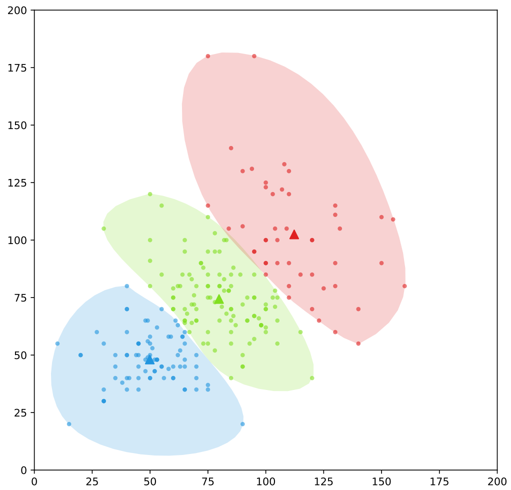
# Identifying Simulation Regions

- 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



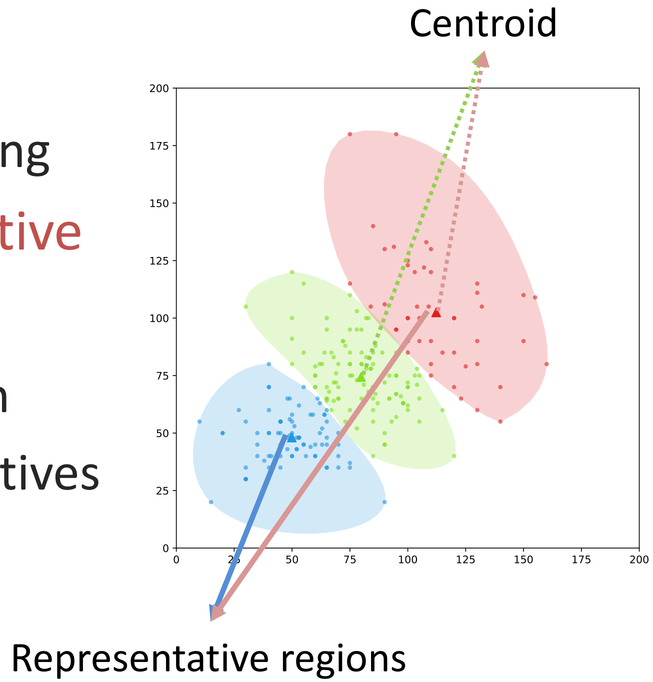
# Identifying Simulation Regions

- 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

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

# 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

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



NUS National University of Singapore

intel

# 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

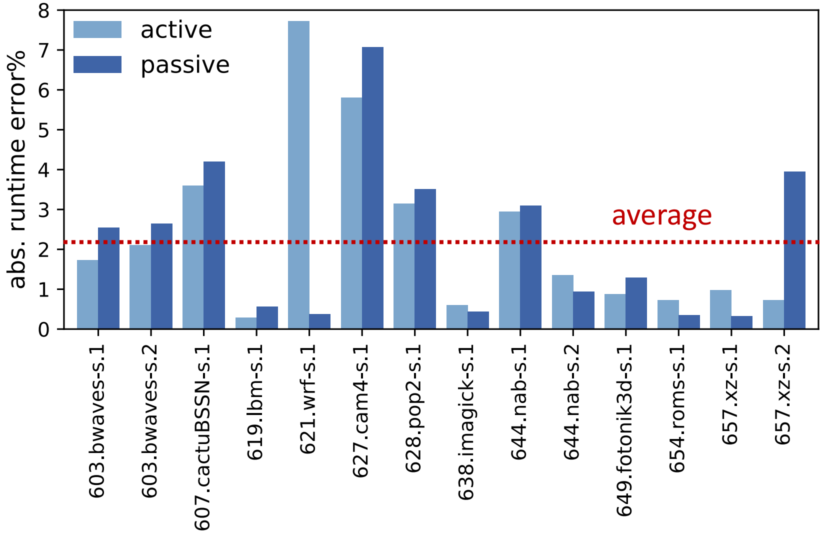
# Workload Type Supported

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

# Accuracy Results

Prediction error wrt. performance of whole application

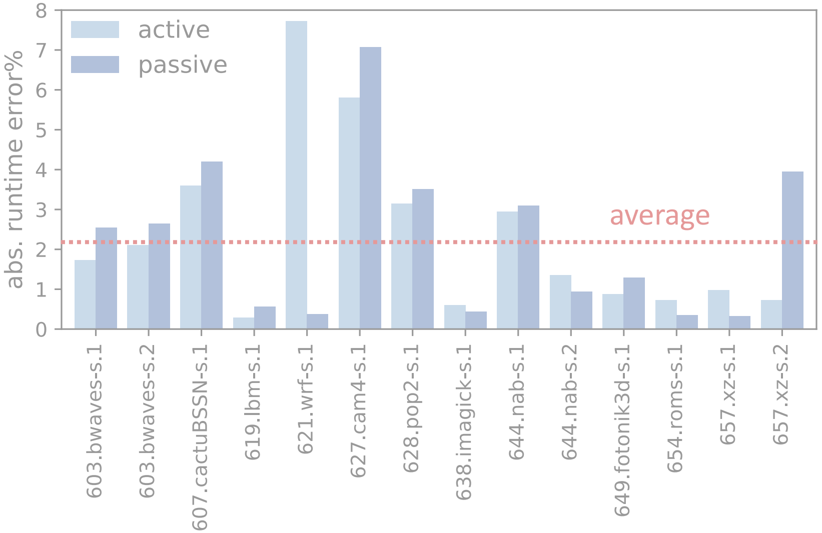
SPEC CPU2017 with train inputs, *8* threads

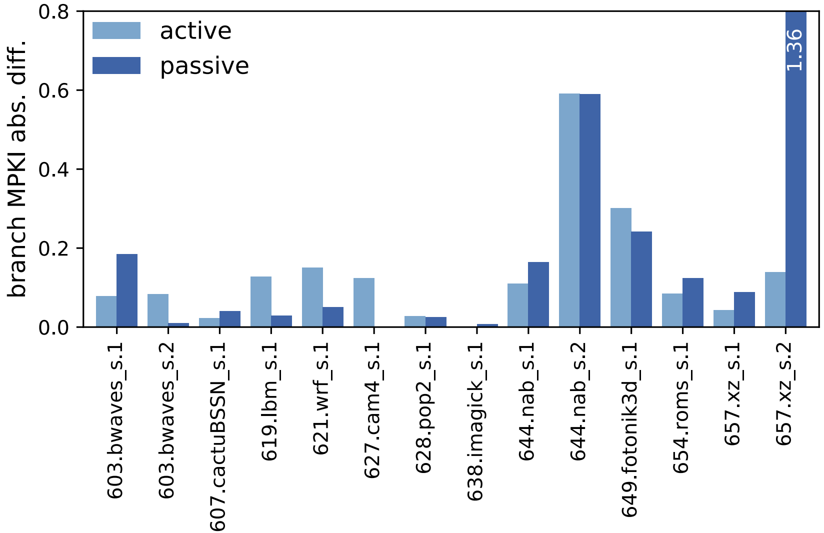


# Accuracy Results

## Prediction error wrt. performance of whole application

SPEC CPU2017 with train inputs, *8* threads

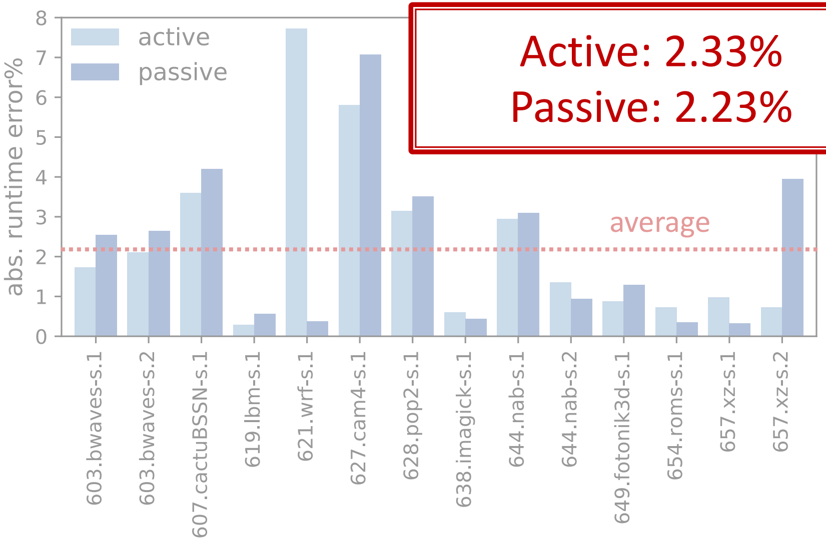


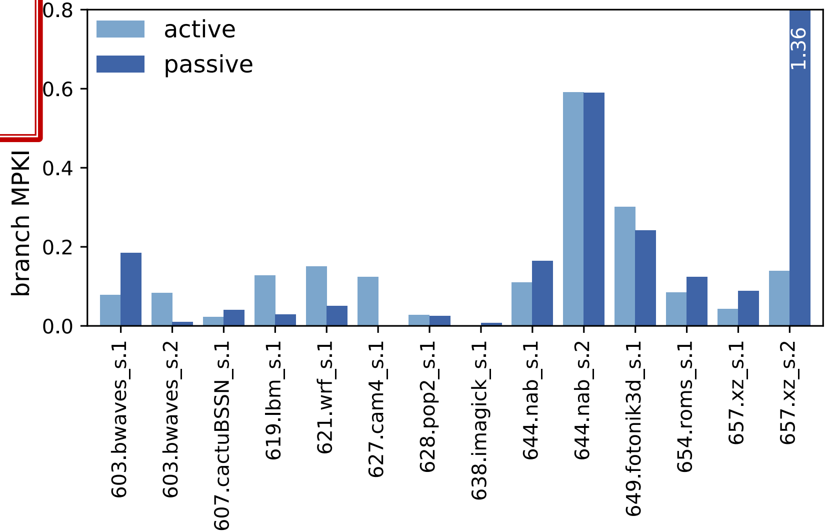
# Accuracy Results

## Prediction error wrt. performance of whole application

SPEC CPU2017 with train inputs, *8* threads



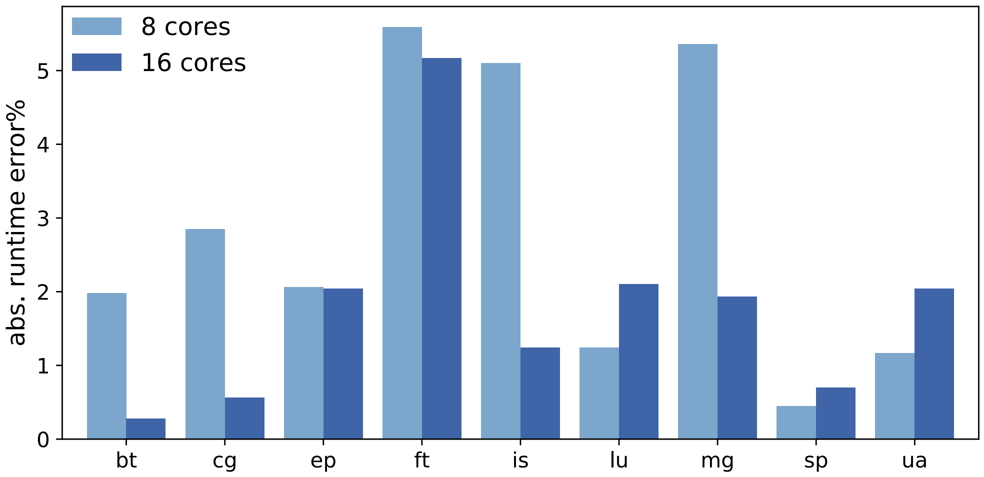

Active: 2.33%
Passive: 2.23%



# Changing Thread Count

Runtime prediction error wrt. whole application runtime

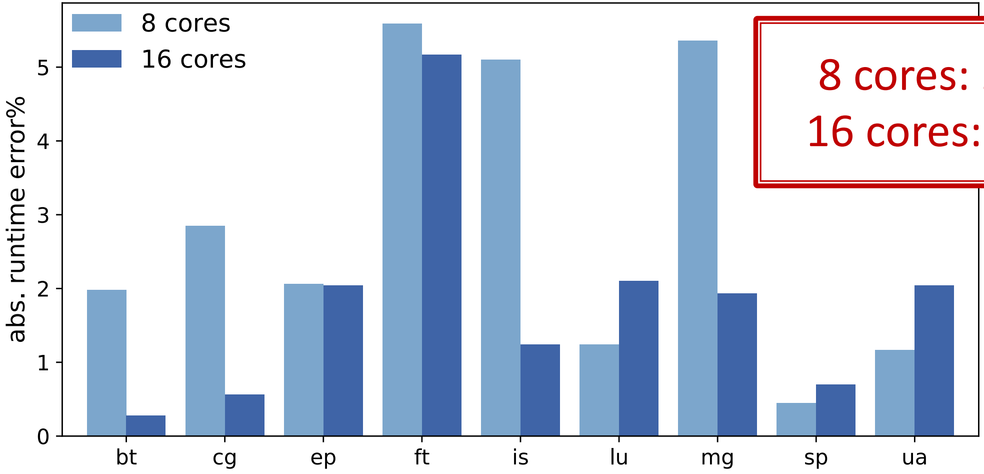
NPB 3.3 with *Class C* inputs, *8* and 16 threads, *passive* wait-policy



# Changing Thread Count

Runtime prediction error wrt. whole application runtime

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

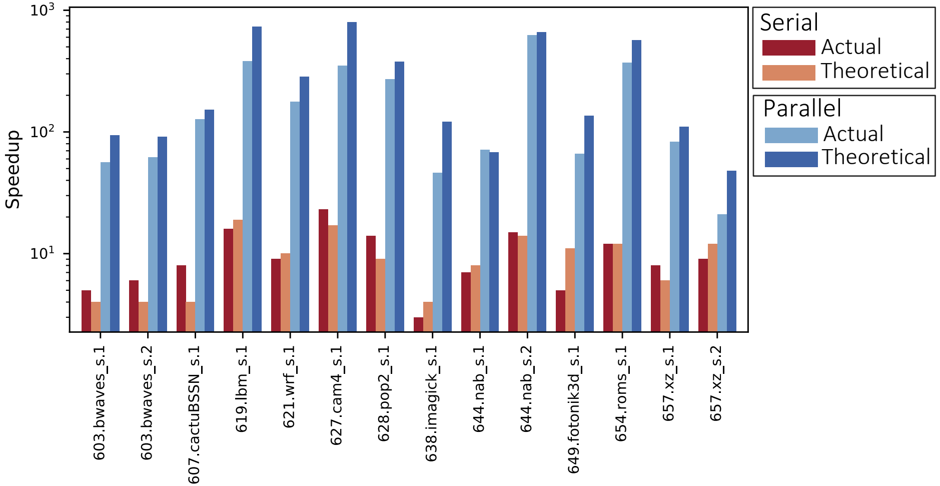



8 cores: 2.87%
16 cores: 1.78%

# Speedup

## Parallel and serial speedup achieved for LoopPoint

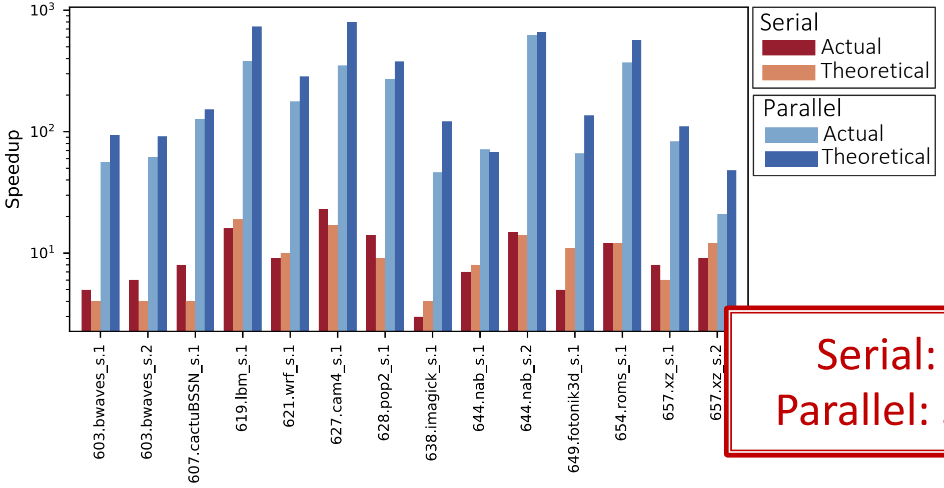
SPEC CPU2017 with *train* inputs, *8* threads, *active* wait-policy



# Speedup

## Parallel and serial speedup achieved for LoopPoint

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

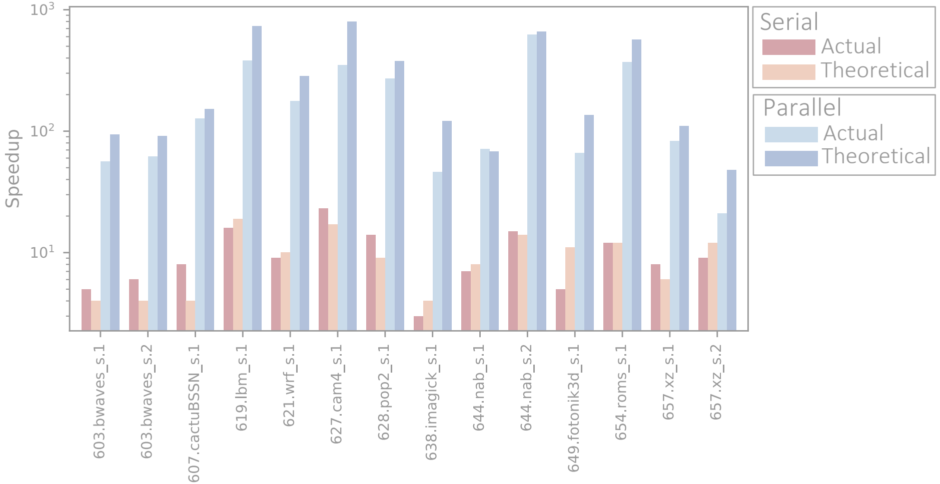



Serial: 9×
Parallel: 303×

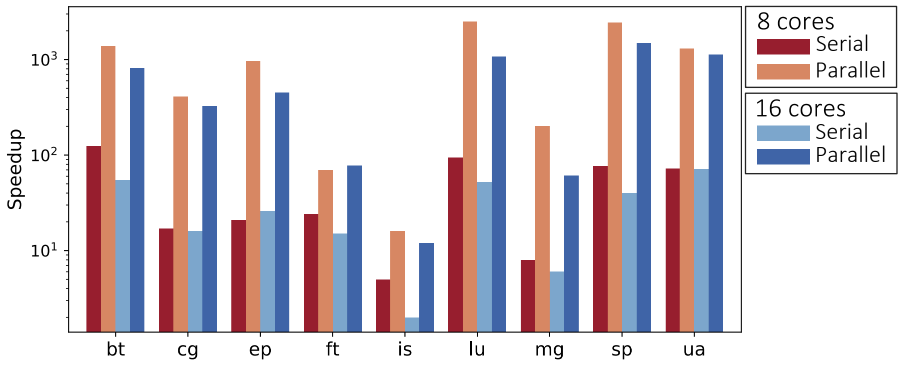
# Speedup

## 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



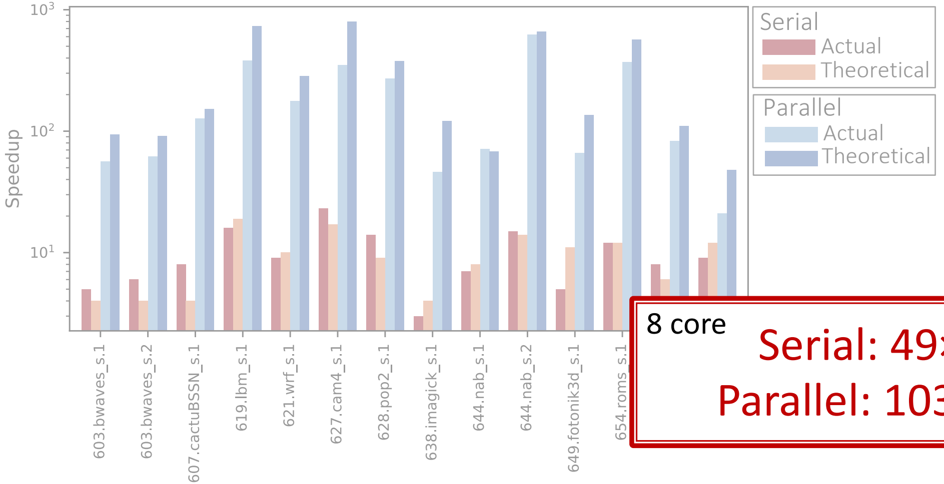



NUS National University of Singapore intel

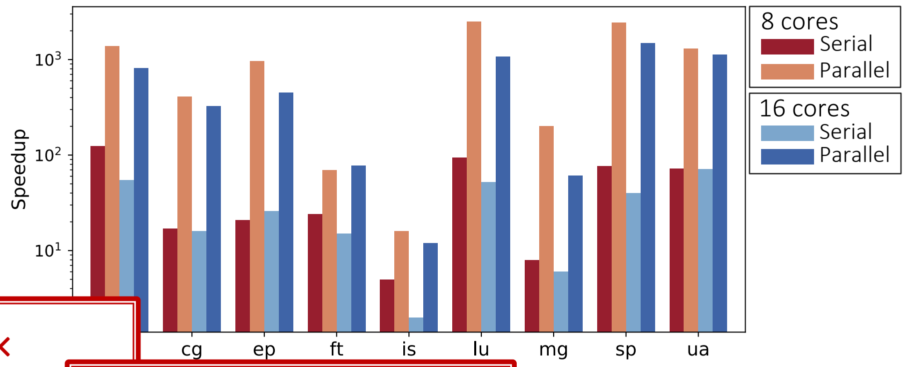
# Speedup

## 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

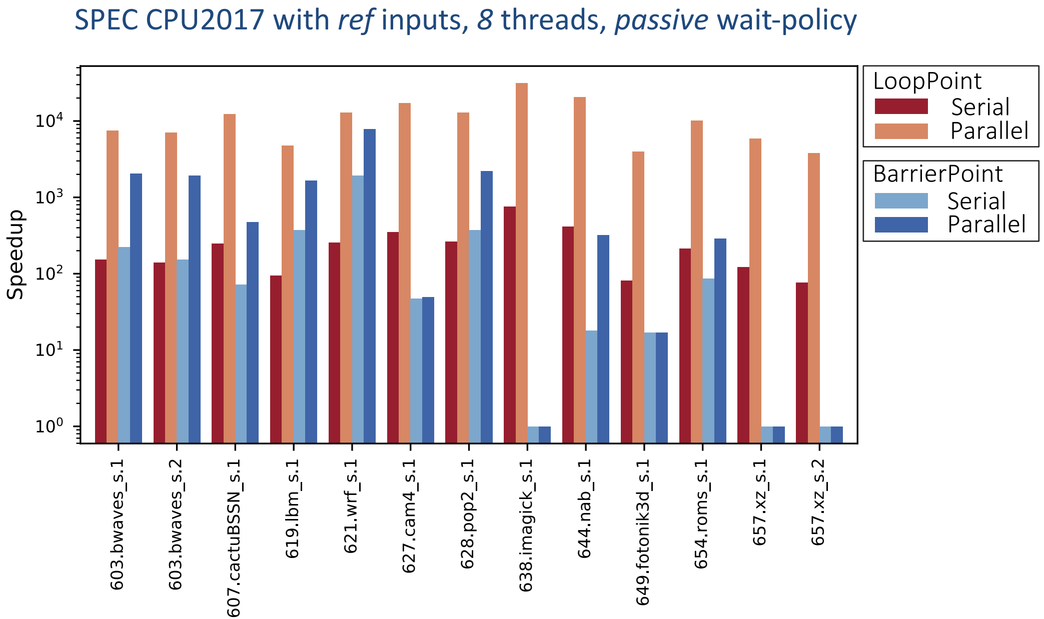



8 core
Serial: 49×
Parallel: 1031×

16 core
Serial: 31×
Parallel: 606×

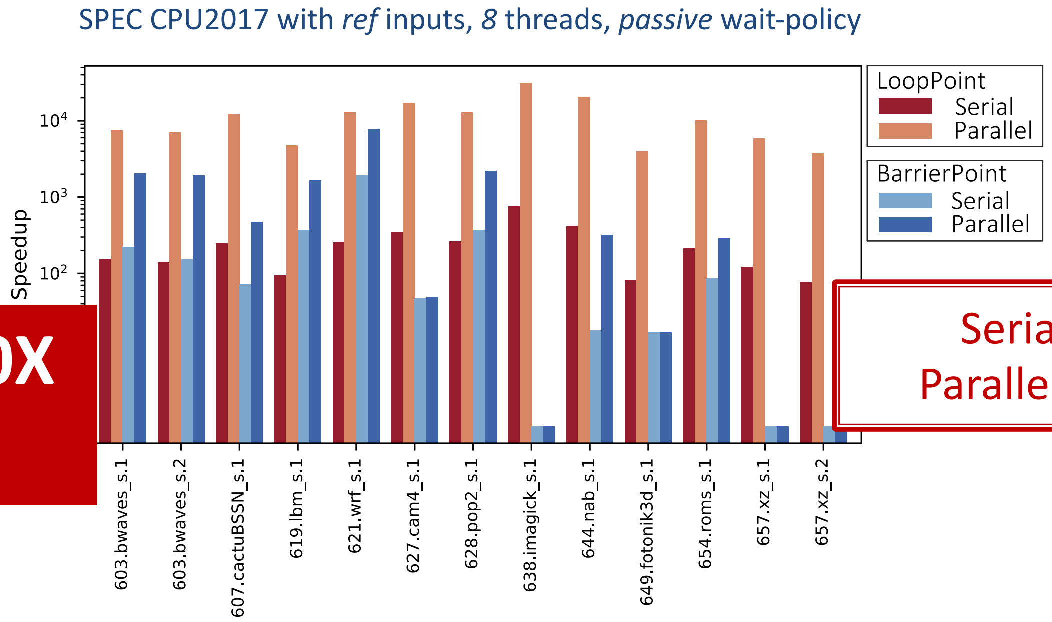
# Speedup

## 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/Tr609MkT42g
  - Questions: alens@comp.nus.edu.sg, tcarlson@comp.nus.edu.sg
- Upcoming tutorial session ➢ *LoopPoint and ELFies*
  - ISCA 2022, New York City

# Agenda

| Time | Speaker | Topic |
| --- | --- | --- |
| 09.00 to 09.10 | Alen Sabu | Overview of the tutorial |
| 09.10 to 10.30 | Harish Patil | Tools from Intel: Pin, PinPlay, SDE, ELFies |
| 10.30 to 10.45 | Break | |
| 10.45 to 11.30 | Akanksha Chaudhari | Simulation and Single-threaded Sampling |
| 11.30 to 11.40 | Break | |
| 11.40 to 12.20 | Alen Sabu | Multi-threaded Sampling and LoopPoint |
| 12.20 to 13.00 | Changxi Liu | Running Sniper and LoopPoint Tools |

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

Alen Sabu[1], Changxi Liu[1], Akanksha Chaudhari[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]

[1]National University of Singapore

[2]Intel Corporation

NUS National University of Singapore

intel

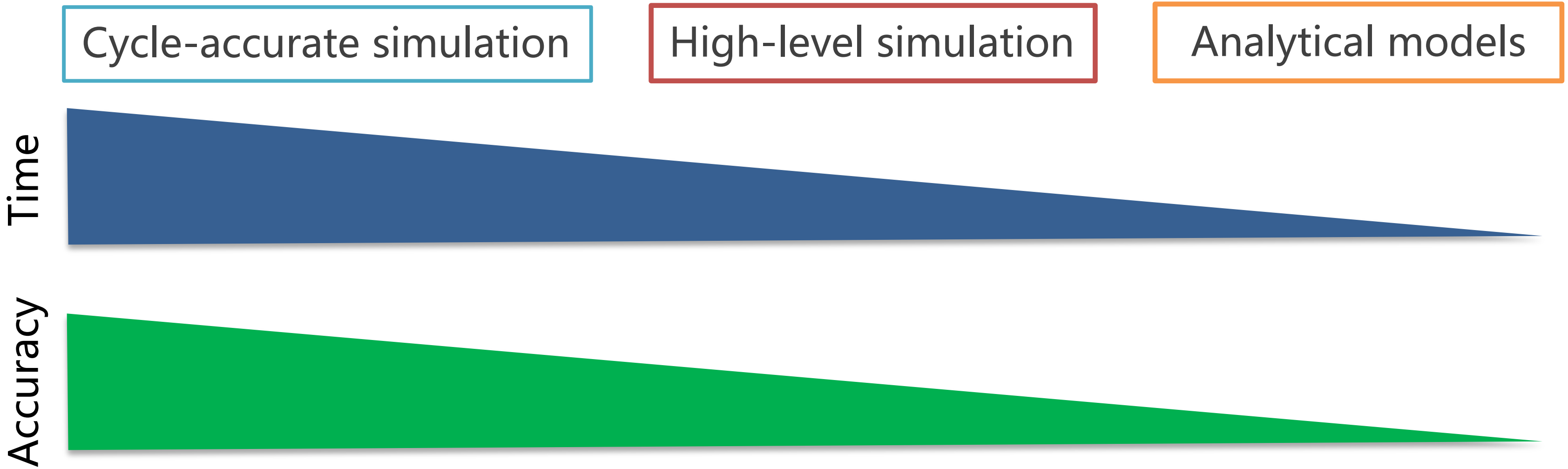
International Symposium on Performance Analysis of Systems and Software, May 22nd 2022, Singapore

Session 4

# Sniper and LoopPoint Demo

CHANGXI LIU, PHD STUDENT
NATIONAL UNIVERSITY OF SINGAPORE

# Simulator Design Waterfall

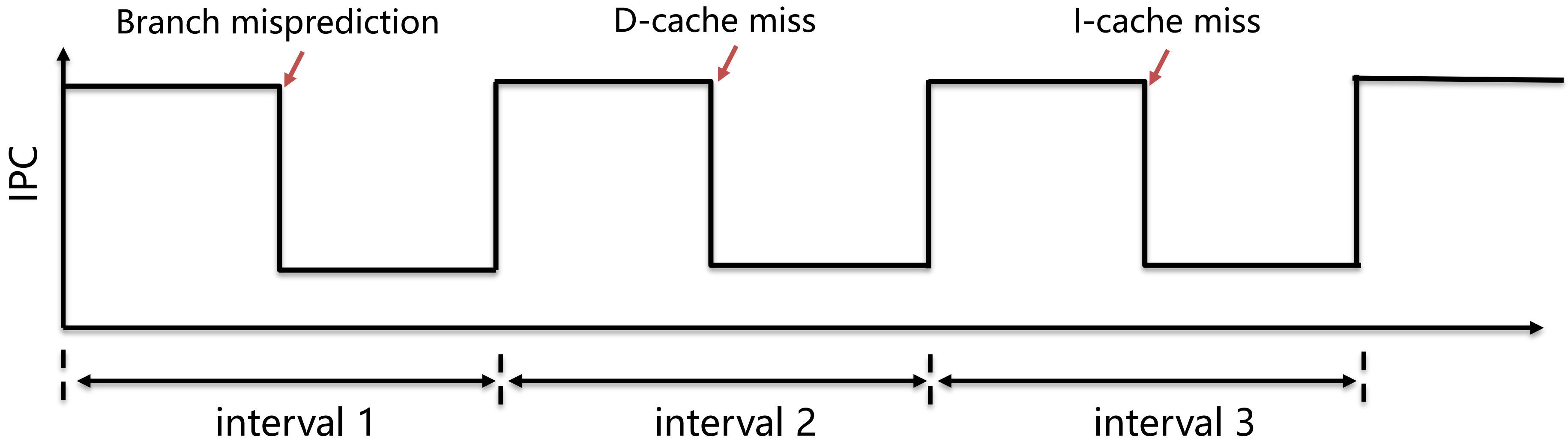



- Cycle-accurate simulation is too slow
- High-level simulation consider both accuracy and execution time

# Sniper: A Fast and Accurate Simulator

- Hybrid simulation approach
  - Analytical interval core model : Interval Model
  - Micro-architecture structure simulation
    - Branch predictors, caches, etc.
- Support multi/many-cores simulation with parallel scales of core number
- Pin-based frontend, can also support dynamoRIO
- Open source https://snipersim.org

# Interval Model

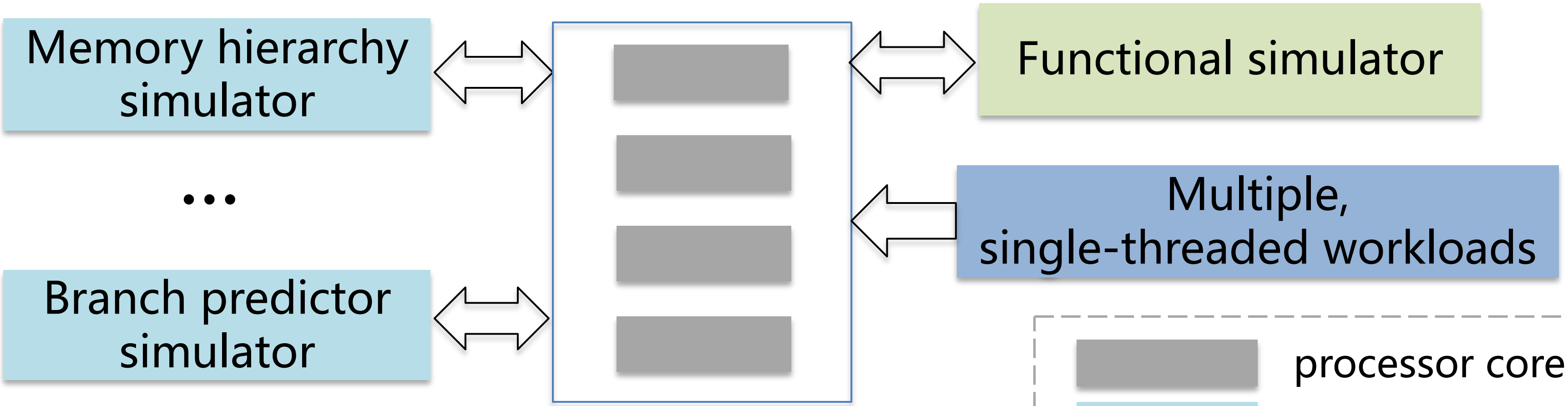


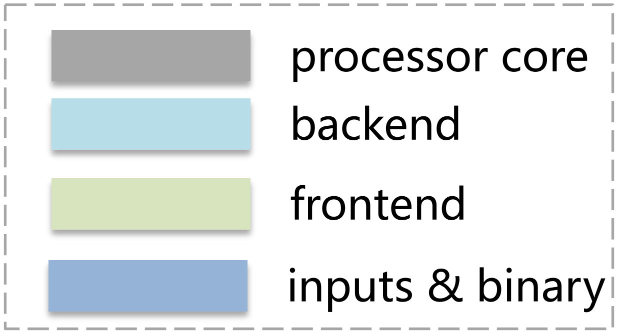

- Split whole application into consecutive intervals



S.Eyerman et al, ACM TOCS, May 2009

# Simulation in Sniper




- Functional simulator as frontend
- Interval models as backend

# Directly Running Sniper

- Download
  - https://snipersim.org/w/Download
  - Register to receive the link via email.

- External Email -

Dear Sniper downloader,

Here are your download instructions for the Sniper Multi-core Simulator.

For use with GIT, clone our repository from the following directory:
$ git clone http://snipersim.org/download/[redacted]/git/sniper.git

To download Sniper, use the following link or command:
http://snipersim.org/download/[redacted]/packages/sniper-latest.tgz
$ wget http://snipersim.org/download/[redacted]/packages/sniper-latest.tgz

If you have any questions, feel free to post them on our mailing list
http://groups.google.com/group/snipersim
or visit our Frequently Asked Questions page
http://snipersim.org/w/Frequently_Asked_Questions

The Sniper Simulator Team

# Directly Running Sniper

- Download
  - [https://snipersim.org/w/Download](https://snipersim.org/w/Download)
  - Register to receive the link via email.
- Simulating an application

```
./run-sniper <options> -- /bin/ls
```

- External Email -

Dear Sniper downloader,

Here are your download instructions for the Sniper Multi-core Simulator.

For use with GIT, clone our repository from the following directory:
$ git clone http://snipersim.org/download/[redacted]/git/sniper.git

To download Sniper, use the following link or command:
http://snipersim.org/download/[redacted]/packages/sniper-latest.tgz
$ wget http://snipersim.org/download/[redacted]/packages/sniper-latest.tgz

If you have any questions, feel free to post them on our mailing list
http://groups.google.com/group/snipersim
or visit our Frequently Asked Questions page
http://snipersim.org/w/Frequently_Asked_Questions

The Sniper Simulator Team

# Build LoopPoint

- Prerequisites
  - X86-based Linux machine
  - C++ with c++11 support
  - Python
  - Docker
  - Sniper link

[1] http://snipersim.org/<path-to-git-repo>.git

# Build LoopPoint

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

[1] http://snipersim.org/<path-to-git-repo>.git

# Build LoopPoint

- `make build`
  - Build docker image

```
or Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support pi
p 21.0 will remove support for this functionality.
Collecting tabulate
  Downloading tabulate-0.8.9.tar.gz (53 kB)
Building wheels for collected packages: tabulate
  Building wheel for tabulate (setup.py): started
  Building wheel for tabulate (setup.py): finished with status 'done'
  Created wheel for tabulate: filename=tabulate-0.8.9-py2-none-any.whl size=33171 sha256=cc5713bdcee7e07c602619a643e2c3132e8e25d18308dc1d0fe06a9ef8b03e12
  Stored in directory: /tmp/pip-ephem-wheel-cache-N3SSxV/wheels/0a/4b/e1/d0e504a346ed0882b93f971fe1122b9de64fabebd9b1d81b9f
Successfully built tabulate
Installing collected packages: tabulate
Successfully installed tabulate-0.8.9
Removing intermediate container 04d7aece39fa
 ---> 98fe9327b5dc
Step 9/9 : RUN pip3 install --no-cache-dir  --upgrade pip &&     pip3 install --no-cache-dir numpy
 ---> Running in 3b699c6d695a
Collecting pip
  Downloading https://files.pythonhosted.org/packages/a4/6d/6463d49a933f547439d6b5b98b46af8742cc03ae83543e4d7688c2420f8b/pip-21.3.1-py3-none-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
Successfully installed numpy-1.19.5
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 e
nvironment instead: https://pip.pypa.io/warnings/venv
Removing intermediate container 3b699c6d695a
 ---> 57f0a752e1e6
[Warning] One or more build-args [TZ_ARG] were not consumed
Successfully built 57f0a752e1e6
Successfully tagged ubuntu:18.04-looppoint
```

# Build LoopPoint

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

```
I have no name!@ef5546e12134[redacted]spass/looppoint$ ls
Dockerfile-ubuntu-18.04  README.md  lplib.py     run-looppoint.py  tools
Makefile                 apps       preprocess   suites.py
I have no name!@ef5546e12134[redacted]ispass/looppoint$
```

# Build LoopPoint

- `make build`
- `make`
- `make apps`
  - Build the provided application
  - matrix-mul demo
  - You can find the source code of the demo in
    - `apps/demo/matrix–omp/`
  - Coming soon: Support for open-source benchmarks (like NPB) with LoopPoint

```
I have no name!@ef5546e12134[redacted]ispass/looppoint$ make apps
make -C apps/demo/matrix-omp
make[1]: Entering directory [redacted]ispass/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/7/libgomp.a(target.o): In function `gomp_targe
(.text+0x8b): warning: Using 'dlopen' in statically linked applications requ
ln -s matrix-omp base.exe
make[1]: Leaving directory [redacted]ispass/looppoint/apps/demo/matrix-omp'
I have no name!@ef5546e12134[redacted]ispass/looppoint$ ls
Dockerfile-ubuntu-18.04  Makefile  README.md  apps  lplib.py  preprocess  ru
I have no name!@ef5546e12134[redacted]ispass/looppoint$ find -name base.exe
./apps/demo/matrix-omp/base.exe
I have no name!@ef5546e12134[redacted]ispass/looppoint$
```

# Build LoopPoint

- `make build`
- `make`
- `make apps`
- `make tools SNIPER_GIT_REPO=[1]`
  - Build Sniper and LoopPoint tools

```
I have no name!@ef5546e12134:[redacted]spass/looppoint$ make tools SNIPER_GIT_REPO=http://snipersim.org/download/[redacted]/git/sniper.git
Downloading Pin
--2022-05-17 07:34:17--  https://software.intel.com/sites/landingpage/pintool/downloads/pin-3.13-98189-g60a6ef199-gcc-linux.tar.gz
Resolving software.intel.com (software.intel.com)... 23.15.96.37, 2600:1417:3f:791::b, 2600:1417:3f:78d::b
Connecting to software.intel.com (software.intel.com)|23.15.96.37|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35897895 (34M) [application/octet-stream]
Saving to: 'STDOUT'

-                                100%[=========================================================================>]  34.23M  11.4MB/s    in 3.0s

2022-05-17 07:34:20 (11.4 MB/s) - written to stdout [35897895/35897895]

patching file pin-3.13-98189-g60a6ef199-gcc-linux/source/tools/InstLib/alarms.H
patching file pin-3.13-98189-g60a6ef199-gcc-linux/source/tools/InstLib/alarms.cpp
Setting SNIPER_GIT_REPO as http://snipersim.org/download/[redacted]/git/sniper.git to download Sniper
Cloning into 'tools/sniper'...
```

```
[CXX   ] sift/recorder/obj-intel64/threads.o
[CXX   ] sift/recorder/obj-intel64/papi.o
[CXX   ] sift/recorder/obj-intel64/bbv_count.o
[CXX   ] sift/recorder/obj-intel64/trace_rtn.o
[CXX   ] sift/recorder/obj-intel64/recorder_base.o
[CXX   ] sift/recorder/obj-intel64/pinboost_debug.o
[CXX   ] sift/recorder/obj-intel64/syscall_modeling.o
make[4]: Entering directory [redacted]ss/looppoint/tools/sniper/sift/recorder/sift'
[CXX   ] sift/recorder/sift/sift_reader.o
[CXX   ] sift/recorder/sift/zfstream.o
[CXX   ] sift/recorder/sift/sift_utils.o
[CXX   ] sift/recorder/sift/sift_writer.o
[LD    ] sift/recorder/sift/libsift.a
make[4]: Leaving directory [redacted]s/looppoint/tools/sniper/sift/recorder/sift'
[LD    ] sift/recorder/obj-intel64/sift_recorder
make[4]: Entering directory [redacted]pass/looppoint/tools/sniper/sift/recorder'
make[4]: Leaving directory [redacted]s/looppoint/tools/sniper/sift/recorder'
make[4]: Entering director[redacted]pass/looppoint/tools/sniper/sift/recorder'
make[4]: Leaving directory [redacted]ss/looppoint/tools/sniper/sift/recorder'
make[4]: Entering directory [redacted]ss/looppoint/tools/sniper/sift/recorder'
make[4]: Leaving directory [redacted]s/looppoint/tools/sniper/sift/recorder'
make[4]: Entering directory[redacted]ss/looppoint/tools/sniper/sift/recorder'
make[4]: Leaving directory [redacted]ss/looppoint/tools/sniper/sift/recorder'
make[4]: Entering directory [redacted]ss/looppoint/tools/sniper/sift/recorder'
make[4]: Leaving directory [redacted]s/looppoint/tools/sniper/sift/recorder'
make[3]: Leaving directory [redacted]ss/looppoint/tools/sniper/sift/recorder'
make[2]: Leaving directory [redacted]ass/looppoint/tools/sniper/sift'
make[2]: Entering director[redacted]pass/looppoint/tools/sniper/standalone'
[DEP   ] standalone/standalone.[redacted]
[DEP   ] standalone/exceptions.d
[CXX   ] standalone/exceptions.o
[CXX   ] standalone/standalone.o
[LD    ] lib/sniper
make[2]: Leaving directory [redacted]ispass/looppoint/tools/sniper/standalone'
make[1]: Leaving directory [redacted]ispass/looppoint/tools/sniper'
I have no name!@ef5546e12134[redacted]spass/looppoint$
```



[1] http://snipersim.org/<path-to-git-repo>.git

# Build LoopPoint

- Opensource code
- We provide the script to help you build the environment
  - `make build`
    - Build docker image
  - `make`
    - Run docker image
  - `make apps`
    - Build the provided applications
  - `make tools SNIPER_GIT_REPO=[1]`
    - Build Sniper and LoopPoint tools

[1] http://snipersim.org/<path-to-git-repo>.git

# Running LoopPoint

- Then run LoopPoint!
  - `./run-looppoint.py -h`
  - Provides the information on how to run the tool
- Example run command
  - `./run-looppoint.py -p demo-matrix-1 -n 8 --force`

# Running LoopPoint

- The driver script of LoopPoint
  - Profiling the application

# Running LoopPoint

- The driver script of LoopPoint
  - 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

# PinPlay

- 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

- A control-flow graph (CFG) is a fundamental structure
- A dynamic control-flow graph (DCFG) is a specialized CFG that adds data from a specific execution of a program
- C++ DCFG APIs is conveniently to used 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

- A control-flow graph (CFG) is a fundamental structure
- A dynamic control-flow graph (DCFG) is a specialized CFG that adds data from a specific execution of a program
- C++ DCFG APIs is conveniently to used 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

- 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 DCFG file name that is generated

# BBV

- 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
  - `-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 driver script of LoopPoint
  - Profiling Final Results:
    - `matrix.1_16448.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-p
count, region-length, region-weight, region-multiplier, region-type

# RegionId = 1 Slice = 0 Icount = 0 Length = 800000067 Weight = 0.12500 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 800000066
#Start: pc : 0x400880 image: matrix-omp offset: 0x880 absolute_count: 1  source-info: matrix-omp.cpp:17
#End: pc : 0x4040c0 image: matrix-omp offset: 0x40c0 absolute_count: 77977888  relative_count: 9837476.0 source-info: matrix-omp.cpp:75
cluster 0 from slice 0,global,1,0x400880,matrix-omp,0x880,1,0x4040c0,matrix-omp,0x40c0,77977888,9837476,800000067,0.12500,1.000,simulation
```

# Running LoopPoint

- The driver script of LoopPoint
  - Profiling Final Results:
    - `matrix.1_16448.global.pinpoints.csv`
    - `(start-pc, start-pc-count ), (end-pc, end-pc-count)`
    - Cluster group id

```
3 # 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-p
  count, region-length, region-weight, region-multiplier, region-type
4
5 # RegionId = 1 Slice = 0 Icount = 0 Length = 800000067 Weight = 0.12500 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 800000066
6 #Start: pc : 0x400880 image: matrix-omp offset: 0x880 absolute_count: 1  source-info: matrix-omp.cpp:17
7 #End: pc : 0x4040c0 image: matrix-omp offset: 0x40c0 absolute_count: 77977888  relative_count: 9837476.0 source-info: matrix-omp.cpp:75
8 cluster 0 from slice 0,global,1,0x400880,matrix-omp,0x880,1,0x4040c0,matrix-omp,0x40c0,77977888,9837476,800000067,0.12500,1.000,simulation
```

# Running LoopPoint

- The driver script of LoopPoint
  - Profiling Final Results:
    - `matrix.1_16448.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-p
count, region-length, region-weight, region-multiplier, region-type

# RegionId = 1 Slice = 0 Icount = 0 Length = 800000067 Weight = 0.12500 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 800000066
#Start: pc : 0x400880 image: matrix-omp offset: 0x880 absolute_count: 1  source-info: matrix-omp.cpp:17
#End: pc : 0x4040c0 image: matrix-omp offset: 0x40c0 absolute_count: 77977888  relative_count: 9837476.0 source-info: matrix-omp.cpp:75
cluster 0 from slice 0,global,1,0x400880,matrix-omp,0x880,1,0x4040c0,matrix-omp,0x40c0,77977888,9837476,800000067,0.12500,1.000,simulation
```

# Running LoopPoint

- The driver script of LoopPoint
  - Profiling the application
    - `matrix.1_16448.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 driver script of LoopPoint
  - Profiling the application
  - Sampled simulation of selected regions

# Sniper

- The LoopPoint support in Sniper
  - Handle the beginning and ending of representative regions

```
VOID Handler(CONTROLLER::EVENT_TYPE ev, VOID * v, CONTEXT * ctxt
 VOID * ip, THREADID tid, BOOL bcast)
{
    switch(ev)
    {
        case CONTROLLER::EVENT_START:
            handleMagic(tid, ctxt, SIM_CMD_USER, 0x0be0000f, 0);
            break;

        case CONTROLLER::EVENT_STOP:
            handleMagic(tid, ctxt, SIM_CMD_USER, 0x0be0000f, 1);
            break;

        default:
            break;
    }
}
```

# Sniper

- The LoopPoint support in Sniper
  - Handle the beginning and ending of representative regions
  - Register this function in pin

```
control_manager.RegisterHandler(Handler, 0, FALSE);
control_manager.Activate();
```

```
VOID Handler(CONTROLLER::EVENT_TYPE ev, VOID * v, CONTEXT * ctxt
 VOID * ip, THREADID tid, BOOL bcast)
{
    switch(ev)
    {
        case CONTROLLER::EVENT_START:
            handleMagic(tid, ctxt, SIM_CMD_USER, 0x0be0000f, 0);
            break;

        case CONTROLLER::EVENT_STOP:
            handleMagic(tid, ctxt, SIM_CMD_USER, 0x0be0000f, 1);
            break;

        default:
            break;
    }
}
```

# Sniper

- The LoopPoint support in Sniper
  - Handle the beginning and ending of representative regions
  - Register this function in pin
  - `./run-sniper -n 8 -gscheduler/type=static -cgainestown -ssimuserroi --roi-script --trace-args=-control start:address:0x4069d0:count235036646:global --trace-args=-control stop:address:0x4069d0:count313177121:global -- <app cmd>`
  - `-control start:address:<PC>:<Count>`
  - `-control end:address:<PC>:<Count>`
  - `PC , Count` : LoopPoint region boundaries

# Running LoopPoint

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

# Extrapolation of Performance Result

- Runtime of corresponding representative region : regionid
- Multiply the ratio : 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 driver script of LoopPoint
  - 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 (%) |
|---|---|---|---|---|---|---|
| matrix-omp.1 | 214544900.0 | 199674000.0 | 6.93 | 8.34 | 4.24 | 100.0 |

# Running LoopPoint

- The driver script of LoopPoint
  - 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 (%) |
|---|---|---|---|---|---|---|
| matrix-omp.1 | 214544900.0 | 199674000.0 | 6.93 | 8.34 | 4.24 | 100.0 |

# Thank you!

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

**Alen Sabu[1], Changxi Liu[1], Akanksha Chaudhari[1], Harish Patil[2], Wim Heirman[2], Trevor E. Carlson[1]**

[1]National University of Singapore

[2]Intel Corporation

International Symposium on Performance Analysis of Systems and Software, May 22nd 2022, Singapore