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

Alen Sabu\textsuperscript{1}, Changxi Liu\textsuperscript{1}, Akanksha Chaudhari\textsuperscript{1}, Harish Patil\textsuperscript{2}, Wim Heirman\textsuperscript{2}, Trevor E. Carlson\textsuperscript{1}

\textsuperscript{1}National University of Singapore
\textsuperscript{2}Intel Corporation

International Symposium on Performance Analysis of Systems and Software, May 22\textsuperscript{nd} 2022, Singapore
## Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.00 to 09.10</td>
<td>Alen Sabu</td>
<td>Overview of the tutorial</td>
</tr>
<tr>
<td>09.10 to 10.00</td>
<td>Harish Patil</td>
<td>Tools from Intel: Pin, PinPlay, SDE, ELFies</td>
</tr>
<tr>
<td>10.00 to 10.15</td>
<td>Break</td>
<td></td>
</tr>
<tr>
<td>10.15 to 11.00</td>
<td>Akanksha Chaudhari</td>
<td>Simulation and Single-threaded Sampling</td>
</tr>
<tr>
<td>11.00 to 11.20</td>
<td>Break</td>
<td></td>
</tr>
<tr>
<td>11.20 to 12.15</td>
<td>Alen Sabu</td>
<td>Multi-threaded Sampling and LoopPoint</td>
</tr>
<tr>
<td>12.15 to 13.00</td>
<td>Changxi Liu</td>
<td>Running Sniper and LoopPoint Tools</td>
</tr>
<tr>
<td>Time</td>
<td>Speaker</td>
<td>Topic</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>09.00 to 09.10</td>
<td>Alen Sabu</td>
<td>Overview of the tutorial</td>
</tr>
<tr>
<td>09.10 to 10.00</td>
<td>Harish Patil</td>
<td>Tools from Intel: Pin, PinPlay, SDE, ELFies</td>
</tr>
<tr>
<td>10.00 to 10.15</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>10.15 to 11.00</td>
<td>Akanksha Chaudhari</td>
<td>Simulation and Single-threaded Sampling</td>
</tr>
<tr>
<td>11.00 to 11.20</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>11.20 to 12.15</td>
<td>Alen Sabu</td>
<td>Multi-threaded Sampling and LoopPoint</td>
</tr>
<tr>
<td>12.15 to 13.00</td>
<td>Changxi Liu</td>
<td>Running Sniper and LoopPoint Tools</td>
</tr>
</tbody>
</table>
• 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
<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.00 to 09.10</td>
<td>Alen Sabu</td>
<td>Overview of the tutorial</td>
</tr>
<tr>
<td>09.10 to 10.00</td>
<td>Harish Patil</td>
<td>Tools from Intel: Pin, PinPlay, SDE, ELFies</td>
</tr>
<tr>
<td>10.00 to 10.15</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>10.15 to 11.00</td>
<td>Akanksha Chaudhari</td>
<td>Simulation and Single-threaded Sampling</td>
</tr>
<tr>
<td>11.00 to 11.20</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>11.20 to 12.15</td>
<td>Alen Sabu</td>
<td>Multi-threaded Sampling and LoopPoint</td>
</tr>
<tr>
<td>12.15 to 13.00</td>
<td>Changxi Liu</td>
<td>Running Sniper and LoopPoint Tools</td>
</tr>
</tbody>
</table>
LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu¹, Changxi Liu¹, Akanksha Chaudhari¹, Harish Patil², Wim Heirman², Trevor E. Carlson¹

¹National University of Singapore
²Intel 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
```

http://pintool.intel.com
PinPlay: Software-based User-level Capture and Replay

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

- Initial memory/register + injections
- Replayer + Simulator
- Internal states initialized
- Initial memory image
- Arch. state
- foo.reg
- foo.text

Inject events: based on instruction counts
- foo.sel / foo.reg (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)
- foo.race (per-thread)

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

- Thread T2 cannot execute instruction 5 until T4 executes instruction 1
- Thread T1 cannot execute instruction 2 until T2 executes instruction 2

**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

User-specified callbacks:
- per process
- per-thread

Memory image (.text)

Injections (.sel)

Thread order (.race)

Arch. State (.reg)

Startup code

User-specified code

Application Memory

Arch. State (per thread)

[http://pinelfie.org](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

<table>
<thead>
<tr>
<th></th>
<th>basic</th>
<th>sim</th>
<th>perf</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How to create</strong></td>
<td><code>scripts/pinball2elf.basic.sh pinball</code></td>
<td><code>scripts/pinball2elf.sim.sh pinball</code></td>
<td><code>scripts/pinball2elf.perf.sh pinball perf.out</code></td>
</tr>
<tr>
<td><strong>Exits gracefully?</strong></td>
<td>NO, either hangs or dumps core</td>
<td>NO, either hangs or dumps core</td>
<td>YES, when retired instruction count reaches pinball icount</td>
</tr>
<tr>
<td><strong>Environment variables used</strong></td>
<td>NONE</td>
<td><code>ELFIE_VERBOSE=0/1</code></td>
<td>&quot;ELFIE_WARMUP&quot; to decide whether to use warmup</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ELFIE_COREBASE=X</code></td>
<td>&quot;ELFIE_PCCONT&quot; to decide how to end warmup/simulation regions</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Set affinity : thread 0 -&gt; core X</code></td>
<td>ELFIE_PERFLIST, enables performance counting</td>
</tr>
</tbody>
</table>

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.0.perf.xml
  └── perf.out.2.perf.txt

ROI start: TSC 4805110586217756
Thread start: TSC 48051110623843452
Simulation end: TSC 4805110625045322
  Sim-end-icount 3436
  hw_cpu_cycles:36148 hw_instructions:3476
  sw_task_clock:141901
Thread end: TSC 48051110625366502
ROI end: TSC 4805110625366502
  hw_cpu_cycles:40097 hw_instructions:4455
  sw_task_clock:188637
**PinPoints** == Pin + SimPoint

Profile with a **pin-based profiler**

Program Execution

Basic-block-vectors

Analyze with SimPoint

Find phases

Choose one simulation point per phase

Intervals: 30 million Instructions each

PinPoint 1: **Weight 30%**

PinPoint 2: **Weight 70%**

PinPoints
**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

Test-program → PinPlay Logger → Whole program pinball → PinPoints → PinPlay Re-logger → Region pinballs → Simulate

Profiler + SimPoint

PinPoints
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](https://www.spec.org/cpu2006/research/simpoint.html)
2. University of Texas at Austin
3. Northwestern University
   Public Release and Validation of SPEC CPU2017 PinPoints
**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

**Record:** … dcfg-driver -dcfg

**Replay:** w/custom PinPlay tool using DCFG API

**DCFG JSON file**

pinball
**PinPlay + DCFG**: Stronger Repeatability

- **Test-program**
- **PinPlay Logger + DCFG generation**
- **Whole program pinball**
- **PinPoints**
- **LoopPoint Profiler + SimPoint**
- **Invariant region markers**
- **Computation loop entries (NOT synchronization)**

DCFJSON file
LoopPoint: Simulation alternatives

Program + input

Whole-program pinball + DCFG

Profile and find representative regions

PinPoints file

1. pinball-driven

2. ELFie-driven

3. Binary-driven

Sniper

GEM5

ELFie

Region pinball

pinball2elf

MT pinballs

Requirement: Execution invariant region specification

(PC+count for compute loop entries)
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
```

- **bbv files** (*bb*), **PinPoints file** (*csv, .CSV*)
- **Region pinballs**
- **Whole-program pinball + DCFG**

Create **LoopPoint** region pinballs and replay them
Summary: Simulation of Multi-threaded Programs: Tools & Methodologies

Where to simulate?

- SDE + LoopPoint
  Compute-loop iterations as ‘Unit of work’

How to simulate?

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

Are the regions representative?

1. Simulation (Sniper) -based
2. ELFie-based / Binary+ROIperf (not covered)

Whole-program performance vs Region-predicted performance
## Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.00 to 09.10</td>
<td>Alen Sabu</td>
<td>Overview of the tutorial</td>
</tr>
<tr>
<td>09.10 to 10.30</td>
<td>Harish Patil</td>
<td>Tools from Intel: Pin, PinPlay, SDE, ELFies</td>
</tr>
<tr>
<td>10.30 to 10.45</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>10.45 to 11.30</td>
<td>Akanksha Chaudhari</td>
<td>Simulation and Single-threaded Sampling</td>
</tr>
<tr>
<td>11.30 to 11.40</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>11.40 to 12.20</td>
<td>Alen Sabu</td>
<td>Multi-threaded Sampling and LoopPoint</td>
</tr>
<tr>
<td>12.20 to 13.00</td>
<td>Changxi Liu</td>
<td>Running Sniper and LoopPoint Tools</td>
</tr>
</tbody>
</table>
LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu¹, Changxi Liu¹, Akanksha Chaudhari¹, Harish Patil², Wim Heirman², Trevor E. Carlson¹

¹National University of Singapore
²Intel 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

• 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.

Fig. 1: Moore Law number of transistor per device: past, present, future
[Intel]
Architectural Trends in Processor Design

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

Fig. 2: Transistor innovations over time

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

Whoa! So many ideas... Which one do I pick?!

Oh, just pick the BEST one.

Architect #1

Cool

Architect #2

Idea #1

Idea #2

Idea #3

...
Exploration and Evaluation of New Ideas

The Important Question:

So how do we then explore new ideas quickly and evaluate them accurately to find the BEST idea?
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.
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.
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**
An “evaluation” of the evaluation methods:

• Theoretical proof
• Analytical modelling
• Simulation
• Prototyping
• Actual implementation
An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation
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
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
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.
An “evaluation” of the evaluation methods:

• Theoretical proof
• Analytical modelling
• Simulation
• Prototyping
• Actual implementation
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.
An “evaluation” of the evaluation methods:

• Theoretical proof
• Analytical modelling
• Simulation
• Prototyping
• Actual implementation
An “evaluation” of the evaluation methods:

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

Relatively Expensive
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
An “evaluation” of the evaluation methods:

- Theoretical proof
- Analytical modelling
- Simulation
- Prototyping
- Actual implementation
An “evaluation” of the evaluation methods:

• Theoretical proof
• Analytical modelling
• Simulation
• Prototyping
• Actual implementation
An “evaluation” of the evaluation methods:

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

[Text box: Most expensive]
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.
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

The most feasible way to explore and evaluate large-scale, complex architectural designs in terms of time, cost and efficiency!
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 \(\rightarrow\) Inaccurate evaluation/results \(\rightarrow\) 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

SIMULATION TECHNIQUES

Classification based on level of detail
- Functional Simulators
- Timing Simulators

Classification based on inputs
- Trace-driven
- Execution-driven
- Checkpoint-driven
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] \(\rightarrow\) fail to capture global variations in program behavior and performance.

![Graph showing data](image_url)
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] \(\rightarrow\) fail to capture global variations in program behavior and performance.
  ▪ [Workload reduction] \(\rightarrow\) benchmark behavior varies significantly across test, train and reference inputs \(\rightarrow\) do not reflect the actual performance.
• 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)
Large-scale program behaviors vary significantly over their run times. Difficulty 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**
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.

Basic Block Vector: 1 1 0 1 5 5 1

Indexed by Basic Block IDs
Maintains the execution count for each Basic Block

Sampled Simulation Techniques: SimPoint
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.

\[
\text{EuclideanDist}(a, b) = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} \quad \text{ManhattanDist}(a, b) = \sum_{i=1}^{D} |a_i - b_i|
\]

- Depicted by Basic Block Similarity Matrices.

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

$$
\text{confidence interval} = \pm \left[ \frac{z \cdot V_x}{\sqrt{n}} \right] \cdot X
$$

(where $X$ is the mean, and $z = 100 \left(1 - \frac{\alpha}{2}\right)$ 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:

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

(where $V_x'$ is the coefficient of variation for the obtained sample)
Sampled Simulation Techniques: SMARTS

- SMARTS uses Systematic Sampling:

  Start sampling at offset $j$
  
  Each unit consists of $U$ instructions

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

  Total sample size:
  
  $n = \frac{N}{k}$ units
  OR
  $n \times U = n \times \frac{N}{k}$ instructions
Simulation:

- U(k -1) – W instructions are functionally simulated and large structures may be warmed.
- U instructions are measured as a sampling unit using detailed simulation.
- W instructions of detailed simulation warm state before each sampling unit.
- Values of U and W depend on workload characteristics.
Sampled Simulation Techniques: SMARTS

- Evaluation results:
  - Average error:
    - 0.64% for CPI
    - 0.59% for EPI
  - Speedup over full-stream simulation:
    - 35x for 8-way out-of-order processors
    - 60x for 16-way out-of-order processors

By simulating fewer than 50 million instructions in detail per benchmark.
### Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.00 to 09.10</td>
<td>Alen Sabu</td>
<td>Overview of the tutorial</td>
</tr>
<tr>
<td>09.10 to 10.30</td>
<td>Harish Patil</td>
<td>Tools from Intel: Pin, PinPlay, SDE, ELFies</td>
</tr>
<tr>
<td>10.30 to 10.45</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>10.45 to 11.30</td>
<td>Akanksha Chaudhari</td>
<td>Simulation and Single-threaded Sampling</td>
</tr>
<tr>
<td>11.30 to 11.40</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>11.40 to 12.20</td>
<td>Alen Sabu</td>
<td>Multi-threaded Sampling and LoopPoint</td>
</tr>
<tr>
<td>12.20 to 13.00</td>
<td>Changxi Liu</td>
<td>Running Sniper and LoopPoint Tools</td>
</tr>
</tbody>
</table>
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 22\(^{nd}\) 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

Simulation run 1

- a: 100M ins
- b: 100M ins
- c: 100M ins
- d: 100M ins
- e: 100M ins

(time)
Extending Single-threaded Techniques

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

Simulation run 1:
- a: 100M ins
- b: 100M ins
- c: 100M ins
- d: 100M ins
- e: 100M ins

Simulation run 2:
- a: 100M ins
- b: 100M ins
- c: 100M ins
- d: 100M ins
- e: 100M ins
SimPoint or SMARTS ➢ Instruction count-based techniques

- Inconsistent regions for multi-threaded applications

Simulation run 1

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>u</td>
</tr>
<tr>
<td>b</td>
<td>v</td>
</tr>
<tr>
<td>c</td>
<td>w</td>
</tr>
<tr>
<td>d</td>
<td>x</td>
</tr>
<tr>
<td>e</td>
<td>y</td>
</tr>
</tbody>
</table>

Time

Simulation run 1
Extending Single-threaded Techniques

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

Simulation run 1:
- T0:
  - a: 100M ins
  - u: 100M ins

Simulation run 2:
- T0:
  - a: 100M ins
  - u: 100M ins

- T1:
  - b: 100M ins
- c: 100M ins
- d: 100M ins
- e: 100M ins
- v: 100M ins
- w: 100M ins
- x: 100M ins
- y: 100M ins

- T1:
  - a: 100M ins
  - u: 100M ins
  - v: 100M ins
  - w: 100M ins
  - x: 100M ins
  - y: 100M ins
  - c: 100M ins
  - d: 100M ins
  - e: 100M ins

Time:
- T0: T1
- T1: T2
Multi-threaded Sampling

FlexPoints

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

Source: 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
BarrierPoint

+ Designed for barrier-synchronized multi-threaded workloads

✓ Scales well with number of barriers

✗ Slow when *inter-barrier regions* are large

Carlson et al., “BarrierPoint: Sampled simulation of multi-threaded applications”, ISPASS’14
Multi-threaded Sampling

TaskPoint

+ Designed for task-based workloads

✅ Uses analytical models to improve accuracy

❌ Works only for the particular workload type

```c
#pragma omp task
    label(task type 1)
    do_something();
```
The Unit of Work

### Single-threaded Sampling
- **SimPoint**
- **SMARTS**

### Multiprocessor Sampling
- **Flex Points**

### Multi-threaded Sampling
- **Time-based sampling**

### Application-specific Sampling
- **BarrierPoint**
- **TaskPoint**

---

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
5. Carlson et al., “BarrierPoint: Sampled simulation of multi-threaded applications”, ISPASS’14
The Unit of Work

We consider generic loop iterations as the unit of work

Single-threaded Sampling

<table>
<thead>
<tr>
<th>SimPoint(^1)</th>
<th>SMARTS(^2)</th>
<th>Instruction count</th>
</tr>
</thead>
</table>

Multiprocessor Sampling

<table>
<thead>
<tr>
<th>Flex Points(^3)</th>
<th>Instruction count</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>BarrierPoint(^5)</th>
<th>Inter-barrier regions</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TaskPoint(^6)</th>
<th>Task instances</th>
</tr>
</thead>
</table>

---

\(^1\) 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

1. Loop-based Profiling

How to simulate

Program binary, inputs
Overall Methodology

1. Loop-based Profiling
2. Region Analysis and Clustering
Looppoints Specification

How to simulate

Program binary, inputs
Overall Methodology

Where to simulate

1. Loop-based Profiling
2. Region Analysis and Clustering
3. Checkpoints Generation
4. (Warmup +) Detailed Region Simulation

How to simulate

Program binary, inputs
Region Checkpoints
Checkpoint driven
Loopponts Specification
Overall Methodology

Where to simulate:
- Program binary, inputs
- 1. Loop-based Profiling
- 2. Region Analysis and Clustering
- Looppoints Specification

How to simulate:
- 3. Checkpoints Generation
- Region Checkpoints
- Checkpoint driven
- 4. (Warmup +) Detailed Region Simulation
- Binary driven

1. Loop-based Profiling
2. Region Analysis and Clustering
3. Checkpoints Generation
4. (Warmup +) Detailed Region Simulation
5. Performance Extrapolation
Loop-based Profiling
Loop-based Profiling

1. Loop-based Profiling

- Application Execution Recording
- DCFG\(^1\) Generation
- Flow-control
- Slice Generation (PC, count)
- Synchronization Filtering
- Per-thread Feature Vectors
- Vector Concatenation

1. Loop-based Profiling

- Program binary, runtime
- Checkpoint Generation
- Region checkpointing
- Checkpoint driven
- Binary driven
- Performance Extrapolation
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
• 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
• **Goal:** Filter out synchronization during profiling
  - Profiling data should contain only *real* work

• **Solutions**
  - Automatic detection using loop analysis
  - Ignore synchronization library code (Ex. libiomp5.so, libpthread.so)

---

**Loop-based Profiling: Sync Filtering**

- **Slice Generation**
  - Flow-control
  - Main image
  - Sync library

- **Synchronization Filtering**
  - Instruction count [Graph Image]
Goal: Filter out synchronization during profiling
- Profiling data should contain only real work

Solutions
- Automatic detection using loop analysis
- Ignore sync library code (Ex. libiomp5.so, libpthread.so)

1Li et al., "Spin detection hardware for improved management of multithreaded systems," TPDS, 2006
Ignore sync library code (Ex. libiomp5.so, libpthread.so)

---

Li et al., "Spin detection hardware for improved management of multithreaded systems," TPDS, 2006
Ignore sync library code (Ex. libiomp5.so, libpthread.so)

Application execution

Profile data

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 ($\#threads \times 100 \text{ M}$)
  - 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

- **Region start/stop**
  - Global instruction count reaches threshold ($#\text{threads} \times 100\ M$)
  - 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

- **Region start/stop**
  - Global instruction count reaches threshold ($\#\text{threads} \times 100 \ M$)
  - Region boundary at a loop entry/exit – use DCFG analysis

- **Looppoint region markers** ($\text{PC}, \text{count}_\text{PC}$)
  - Global count of loop entries: invariant across executions
  - Simulate the same *amount of work*

+ $\text{Program execution} \quad \text{Threshold Instructions} \quad \text{Region/Slice}$

+ $\text{(PC}_1, \text{count}_1) \quad \text{Loop A} \quad \text{Loop B} \quad \text{Loop A} \quad \ldots \quad \text{Loop B} \quad \text{Loop A} \quad \text{(PC}_2, \text{count}_2)$

- **Slice Generation (PC, count)**
- **Flow-control**
- **Synchronization Filtering**
Loop-based Profiling: Slice Generation

• Basic Block (BB)
  § Section of code with single entry and exit
• Basic Block Vector (BBV)
  § Execution fingerprint of an application interval
  § Vector with one element for each basic block
  § $\text{Exec Wt} = \text{entry count} \times \text{number of instructions}$

<table>
<thead>
<tr>
<th>ID</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>
| 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 |

Example Assembly Code
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
  - \( \text{Exec Wt} = \text{entry count} \times \text{number of instructions} \)

### Example Assembly Code

<table>
<thead>
<tr>
<th>BB</th>
<th>Example Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>srl a2, 0x8, t4</td>
</tr>
<tr>
<td></td>
<td>and a2, 0xff, t12</td>
</tr>
<tr>
<td></td>
<td>addl zero, t12, s6</td>
</tr>
<tr>
<td></td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0x25, v0</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0, t0</td>
</tr>
<tr>
<td></td>
<td>bis v0, t0, v0</td>
</tr>
<tr>
<td></td>
<td>bne v0, 0x120018c48</td>
</tr>
<tr>
<td>B</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td>C</td>
<td>ble t7, 0x120018bb4</td>
</tr>
</tbody>
</table>

**ID:** A B C

- **BB Exec Count:** < 1, 20, 0, ...>
- **weigh by Block Size:** < 8, 3, 1, ...>

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

<table>
<thead>
<tr>
<th>BB</th>
<th>Example Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>srl a2, 0x8, t4</td>
</tr>
<tr>
<td></td>
<td>and a2, 0xff, t12</td>
</tr>
<tr>
<td></td>
<td>addl zero, t12, s6</td>
</tr>
<tr>
<td></td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0x25, v0</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0, t0</td>
</tr>
<tr>
<td></td>
<td>bis v0, t0, v0</td>
</tr>
<tr>
<td></td>
<td>bne v0, 0x120018c48</td>
</tr>
<tr>
<td>B</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td>C</td>
<td>ble t7, 0x120018bb4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID:</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB Exec Count:</td>
<td>&lt; 1, 20, 0, ...&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weigh by Block Size:</td>
<td>&lt; 8, 3, 1, ...&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB Exec Wt:</td>
<td>&lt; 8, 60, 0, ...&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Loop-based Profiling: Slice Generation

The example assembly code is as follows:

```
[ A:8, B:60, C:0, ... ]
```

<table>
<thead>
<tr>
<th>BB</th>
<th>Example Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>srl a2, 0x8, t4</td>
</tr>
<tr>
<td></td>
<td>and a2, 0xff, t12</td>
</tr>
<tr>
<td></td>
<td>addl zero, t12, s6</td>
</tr>
<tr>
<td></td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0x25, v0</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0, t0</td>
</tr>
<tr>
<td></td>
<td>bis v0, t0, v0</td>
</tr>
<tr>
<td></td>
<td>bne v0, 0x120018c48</td>
</tr>
<tr>
<td>B</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td>C</td>
<td>ble t7, 0x120018bb4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ID: A B C

BB Exec Count: < 1, 20, 0, ...>

weigh by Block Size: < 8, 3, 1, ...>

BB Exec Wt: < 8, 60, 0, ...>

Ratio of instructions per thread may differ

- **Global-BBVs**: Concatenate per-thread BBVs to larger Global BBVs
Loop-based Profiling: Vector Concatenation

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

<table>
<thead>
<tr>
<th>BB</th>
<th>Example Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>B</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>C</td>
<td>ble t7, 0x120018bb4</td>
</tr>
<tr>
<td>M</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>gt t7, 0x120018b90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BB</th>
<th>Example Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>B</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>C</td>
<td>ble t7, 0x120018bb4</td>
</tr>
<tr>
<td>M</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>gt t7, 0x120018b90</td>
</tr>
</tbody>
</table>
Loop-based Profiling: Vector Concatenation

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

Example Assembly Code

```
subl    a2, 0x8, t4
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
subl    t7, 0x1, t7
cmple   t7, 0x3, t2
beq     t2, 0x120018b04
ble     t7, 0x120018bb4
```

Thread 0

Thread 1
### Loop-based Profiling: Vector Concatenation

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

#### Example Assembly Code

<table>
<thead>
<tr>
<th>BB ID</th>
<th>Example Assembly Code</th>
<th>BB Exec Wt</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>srl a2, 0x8, t4</td>
<td>&lt; 8</td>
</tr>
<tr>
<td></td>
<td>and a2, 0xff, t12</td>
<td>60</td>
</tr>
<tr>
<td>O</td>
<td>addl zero, t12, s6</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>subl t7, 0x1, t7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0x25, v0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0, t0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bis v0, t0, v0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bne v0, 0x120018c48</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>subl t7, 0x1, t7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gt t7, 0x120018b90</td>
<td></td>
</tr>
</tbody>
</table>

#### BB IDs and Exec Wts

<table>
<thead>
<tr>
<th>BB ID</th>
<th>Exec Wt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

#### Thread 1

- BB ID: A, B, C, ...
- BB Exec Wt: < 8, 60, 0, ...

#### Thread 0

- BB ID: N, O, P, ...
- BB Exec Wt: < 5, 90, 3, ...

---

Loop-based Profiling: Vector Concatenation

![Diagram of thread execution with BB IDs and execution weights]
Loop-based Profiling: Vector Concatenation

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

Example Assembly Code

```
subl    t7, 0x1, t7
gt      t7, 0x120018b90
ble     t7, 0x120018bb4
subl    t7, 0x1, t7
cmpeq   t7, 0x3, t2
beq     t2, 0x120018b04
subl    t7, 0x1, t7
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
```

Loop-based Profiling: Vector Concatenation

<table>
<thead>
<tr>
<th>BB ID:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB Exec Wt:</td>
<td>&lt; 8, 60, 0, ... &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BB ID:</th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB Exec Wt:</td>
<td>&lt; 5, 90, 3, ... &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Global-BBV
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 {
2848     ....
2849     for (x=0; x < (ssize_t)image->columns; x++)
2850         {
2851             for (v=0; v < (ssize_t)kernel->height; v++) {
2852                 for (u=0; u < (ssize_t)kernel->width; u++, k--) {
2853                     ....
2854                 } /* u */
2855             }/* v */
2856         } /* x */
2857     } /* y */
2858     ....

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 {
2848     ...
2849     for (x=0; x < (ssize_t) image->columns; x++)
2850     {
2851         ...
2852         for (v=0; v < (ssize_t) kernel->height; v++) {
2853             ...
2854             } /* u */
2855         }
2856     }
2857     /* y */
2858 }
2859 /* x */
2860 /* 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 \((\text{PC}, \text{count}_{\text{PC}})\) 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 \((PC, \text{count}_{PC})\) 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
• Representative regions (loops) 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

$$\text{total runtime} = \sum_{i=\text{rep}_1}^{\text{rep}_N} \text{runtime}_i \times \text{multiplier}_i$$
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 performance.

\[
multiplier_j = \frac{\sum_{i=0}^{m} \text{inscount}_i}{\text{inscount}_j}
\]

\[
total \ runtime = \sum_{i=rep_1}^{rep_N} \ runtime_i \times multiplier_i
\]

\(m\) regions represented by \(j^{th}\) looppoint.
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

## SPEC CPU2017 Analysis

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

Source: SPEC CPU®2017 documentation index
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

abs. runtime error%
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

8 cores
16 cores

abs. runtime error%
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%
Parallel and serial speedup achieved for LoopPoint

SPEC CPU2017 with *train* inputs, 8 threads, *active* wait-policy
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
Parallel and serial speedup achieved for LoopPoint

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

<table>
<thead>
<tr>
<th>Serial</th>
<th>Parallel</th>
<th>Theoretical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>Actual</td>
<td>Theoretical</td>
</tr>
<tr>
<td>Parallel</td>
<td>Actual</td>
<td>Theoretical</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Serial</th>
<th>Parallel</th>
<th>Theoretical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>Actual</td>
<td>Theoretical</td>
</tr>
<tr>
<td>Parallel</td>
<td>Actual</td>
<td>Theoretical</td>
</tr>
</tbody>
</table>

- **8 cores**
  - Serial: 49×
  - Parallel: 1031×

- **16 cores**
  - Serial: 31×
  - Parallel: 606×
Theoretical Speedup comparison with BarrierPoint

SPEC CPU2017 with ref inputs, 8 threads, passive wait-policy

LoopPoint
- Serial
- Parallel

BarrierPoint
- Serial
- Parallel
Theoretical Speedup comparison with BarrierPoint

SPEC CPU2017 with ref inputs, 8 threads, passive wait-policy

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](https://github.com/nus-comparch/looppoint)
  - Page: [https://looppoint.github.io](https://looppoint.github.io)
  - Short talk: [https://youtu.be/Tr6O9MkT42g](https://youtu.be/Tr6O9MkT42g)
  - Questions: [alens@comp.nus.edu.sg](mailto:alens@comp.nus.edu.sg), [tcarlson@comp.nus.edu.sg](mailto:tcarlson@comp.nus.edu.sg)

• **Upcoming tutorial session** ➤ *LoopPoint and ELFies*
  - ISCA 2022, New York City
## Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.00 to 09.10</td>
<td>Alen Sabu</td>
<td>Overview of the tutorial</td>
</tr>
<tr>
<td>09.10 to 10.30</td>
<td>Harish Patil</td>
<td>Tools from Intel: Pin, PinPlay, SDE, ELFies</td>
</tr>
<tr>
<td>10.30 to 10.45</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>10.45 to 11.30</td>
<td>Akanksha Chaudhari</td>
<td>Simulation and Single-threaded Sampling</td>
</tr>
<tr>
<td>11.30 to 11.40</td>
<td></td>
<td>Break</td>
</tr>
<tr>
<td>11.40 to 12.20</td>
<td>Alen Sabu</td>
<td>Multi-threaded Sampling and LoopPoint</td>
</tr>
<tr>
<td>12.20 to 13.00</td>
<td>Changxi Liu</td>
<td>Running Sniper and LoopPoint Tools</td>
</tr>
</tbody>
</table>
LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu¹, Changxi Liu¹, Akanksha Chaudhari¹, Harish Patil², Wim Heirman², Trevor E. Carlson¹

¹National University of Singapore
²Intel Corporation

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
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
Simulation in Sniper

- Functional simulator as frontend
- Interval models as backend

Multiple, single-threaded workloads

Memory hierarchy simulator

Branch predictor simulator
Directly Running Sniper

• Download
  - [https://snipersim.org/w/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:

```bash
$ git clone http://snipersim.org/download/[REDACTED]/git/sniper.git
```

To download Sniper, use the following link or command:

```bash
http://snipersim.org/download/[REDACTED]/packages/sniper-latest.tgz
```

```bash
$ 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:


To download Sniper, use the following link or command:

- [http://snipersim.org/download/package](http://snipersim.org/download/package)

If you have any questions, feel free to post them on our mailing list

[http://groups.google.com/group/snipersim](http://groups.google.com/group/snipersim)
or visit our Frequently Asked Questions page

[http://snipersim.org/w/Frequently Asked Questions](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

Build LoopPoint

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

Build LoopPoint

- make build
  - Build docker image
Build LoopPoint

- make build
- make
  - Run the docker image

```
I have no name!@ef5546e12134   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   spass/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
Build LoopPoint

- make build
- make
- make apps
- make tools SNIPER_GIT_REPO=[1]

  - Build Sniper and LoopPoint tools

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

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
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 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
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:
  ▪ `$SDEBUILDKIT/pinplay-scripts/replay.py --pintool=sde-global-looppoinpoint.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
• Profiling the feature vector of each region

• Command:
  

  - `--pccount_regions`: (PC, count)-based region information
  - `--looppont: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
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)
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

```plaintext
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-pc-count, region-length, region-weight, region-multiplier, region-type
4 # RegionId = 1 Slice = 0 Icount = 0 Length = 800000067 Weight = 0.12500 Multiplier = 1.000 ClusterSlicecount = 1 ClusterIcount = 800000066
5 #Start: pc : 0x400880 image: matrix-omp offset: 0x880 absolute_count: 1 source-info: matrix-omp.cpp:17
6 #End: pc : 0x4040c0 image: matrix-omp offset: 0x40c0 absolute_count: 77977888 relative_count: 9837476.0 source-info: matrix-omp.cpp:75
7 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
• 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
The LoopPoint support in Sniper
  - Handle the beginning and ending of representative regions

```c
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

```c
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;
    }
}

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

```python
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

<table>
<thead>
<tr>
<th>application</th>
<th>runtime actual (ns)</th>
<th>runtime predicted (ns)</th>
<th>error (%)</th>
<th>speedup parallel</th>
<th>speedup serial</th>
<th>coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix-omp.1</td>
<td>214544900.0</td>
<td>199674000.0</td>
<td>6.93</td>
<td>8.34</td>
<td>4.24</td>
<td>100.0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>application</th>
<th>runtime actual (ns)</th>
<th>runtime predicted (ns)</th>
<th>error (%)</th>
<th>speedup (parallel)</th>
<th>speedup (serial)</th>
<th>coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix-omp.1</td>
<td>214544900.0</td>
<td>199674000.0</td>
<td>6.93</td>
<td>8.34</td>
<td>4.24</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Thank you!
LoopPoint and ELFies: Tools and Techniques to Accelerate Simulations of Multi-threaded Applications using Checkpointing

Alen Sabu¹, Changxi Liu¹, Akanksha Chaudhari¹, Harish Patil², Wim Heirman², Trevor E. Carlson¹

¹National University of Singapore
²Intel Corporation