OptiWISE: Combining Sampling and Instrumentation for Granular CPI Analysis

Yuxin Guo [1 §] Alexandra W. Chadwick [1 §] Márton Erdős [1] Utpal Bora [1]
Ilias Vougioukas [2] Giacomo Gabrielli [3] Timothy M. Jones [1]

[1]*University of Cambridge* [2]*Arm, USA* [3]*Arm, UK*

Summary

- Problem: Existing profiling techniques do not always correctly and directly identify the optimisation opportunities.
- Our Goal: Develop a profiling tool providing fine-grained CPI metrics, finding the bottleneck of applications easily.
- Key idea: Run the sampling twice: once for sampling and another for instrumentation, and then combine their results.
- Implementation: Sampling is done by `perf`, and `DynamoRIO` does instrumentation. The CPI metric is computed by the ratio of samples to execution counts.
- Results: OptiWISE accurately estimates the CPI of instructions and sets of instructions at varying granularities (from single instruction to loop) with acceptable overhead: 8.1× geometric mean slowdown tested on SPEC CPU 2017.

Background

- **Sampling**-based profiling
  - Generate interrupts to read hardware counters.
  - If interrupts are generated periodically, then a higher number of samples on an instruction indicates a higher execution time.
  - But it has no idea about why there is a high execution time.

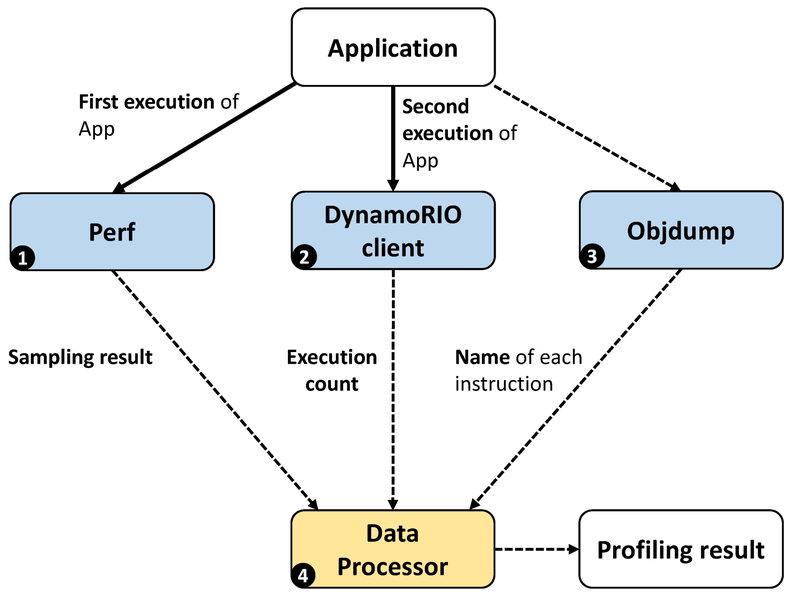
- **Instrumentation**-based profiling
  - Insert monitoring code to read times or obtain other program behaviours (e.g., execution count of each instruction).
  - But this information is not usually related to real performance: inserted code strongly impacts the performance.

Motivation of OptiWISE

- Using sampling or instrumentation alone does not always identify the true bottleneck.
- **But the optimisation insights are clear after combining sampling and instrumentation.**

	Instruction	OptiWISE Samples	OptiWISE Executions	OptiWISE CPI
→	00 **sub** $0x2770d4ee,%eax	574	256*10⁶	0.91
	04 **xor** $0x7aa3411f,%eax	618	256*10⁶	0.97
	0f **sub** $0x1,%edx	0	256*10⁶	0.00
	12 **jne** 00	65	256*10⁶	0.10
	14 **mov** %eax,%edx	0	1*10⁶	0.00
	16 **add** $0x1,%ecx	0	1*10⁶	0.00
	19 **and** $0x1fffffff,%edx	1	1*10⁶	0.40
	1f **xor** (%rsi,%rdx,4),%eax	247	1*10⁶	98.97
	22 **cmp** %ecx,%ebx	0	1*10⁶	0.00
	24 **mov** $0x100,%edx	0	1*10⁶	0.00
→	29 **jne** 00	0	1*10⁶	0.00

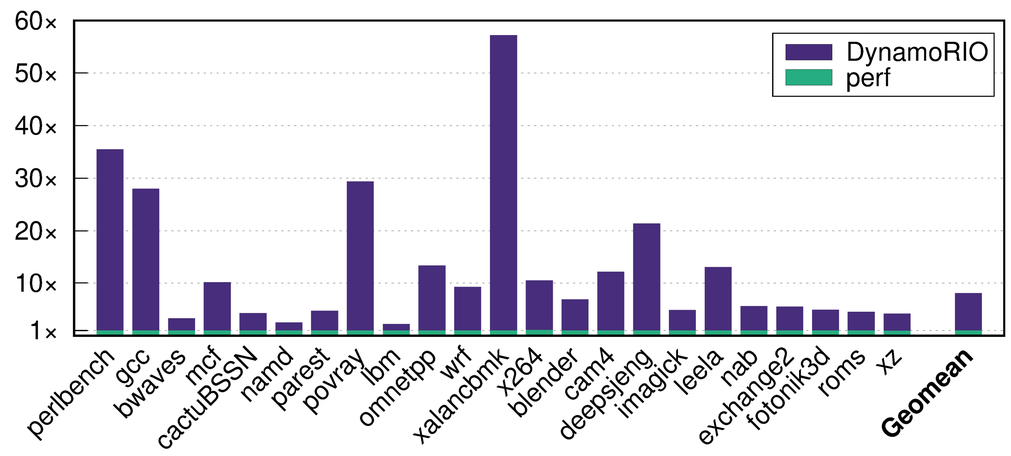
Implementation of OptiWISE



- **Perf** is used to sample the application (first execution), generating interrupts periodically to read the hardware CPU cycle counter.
- **DynamoRIO** is used to instrument the application (second execution), obtaining the execution counts of each instruction and a CFG with edge counts of the program.
- **Objdump** is used to read the name of each instruction and debug information (if available).
- All the above information is fed into a static data processor to output the profiling information.

Results

- Overhead
  - SPEC CPU 2017 benchmarks.
  - Evaluated on an Intel Xeon W-2195 system.
    Ubuntu 20.04, 2.30GHz, 256GB memory, 1.1/18/24 MiB L1/L2/L3 cache.
  - 8.1× geomean slowdown and 57× for the worst case.



- Accuracy
  - OptiWISE's accuracy only depends on the sampling part (i.e., perf).
  - Some instructions are never sampled due to the out-of-order execution.
  - Samples may not be attributed to the correct instruction, known as 'skid'.

- Case Study
  - We optimise three workloads based on their OptiWISE profiling results.
    505.mcf, 531.deepsjeng, and 603.bwaves.
  - OptiWISE clearly shows the optimisation opportunities.
    E.g., branch miss predictions, cache misses.
  - Optimisations give 12%, 6.8%, and 2% whole-benchmark speedups.

Paper Code

This work was supported by EPSRC (grant EP/W00576X/1) and Arm.

§ These authors contributed equally to this paper.

contact: yg413@cam.ac.uk