A method for automatically generating analogue benchmark suites using low-level hardware metrics

Simon McIntosh-Smith
University of Bristol, UK
simonm@cs.bris.ac.uk

Owen Thomas
Red Oak Consulting
owen@redoakconsulting.co.uk
Introduction

• There are challenges in constructing and maintaining benchmark suites, especially when the original codes are complex or classified
  • Thus may not be shared directly with vendors or partners
• Can we *automatically* construct benchmarks that closely model the low-level behavior of target workloads?
Methodology – I

• Select a set of open-source benchmarks that exhibit a reasonably wide range of behavior in terms of low-level hardware metrics:
  • Cycles per instruction
  • Cache misses (instruction, data, multiple levels)
  • DRAM accesses
  • Data Translation Lookaside Buffer Misses (page faults)
  • Branch instruction rate
  • Mispredicted branches rate

• These metrics were selected as the most important in terms of their effect on the observed performance in a modern CPU architecture
Methodology – II

- Characterise target codes\(^1\);
- Characterise potential analogue benchmarks\(^1\);
- Create benchmark analogues;
- Assemble parameterised benchmark suite.

\(^1\) Multiple copies in parallel, one per core
Characterisation tools

- **System Tap**
  - Standard Linux tool
  - Records kernel-level stats about memory use, I/O etc

- **Oprofile**
  - Standard Linux tool
  - Records low-level hardware metrics (cycles per instruction, branch misprediction rate etc)

- Sampling issues have to be managed
Target benchmark characterization on Nehalem

Nehalem hardware counters from a twelve hour target job mix sample run
Nehalem cycles per instruction
Characterising targets

• Do we run them:
  • On one core at a time,
  • On all the cores at the same time, or
  • In a random mix?

• Observed the impact of running each target in parallel with itself

• Some counter sampling rate issues (too few and too many are both problems).
Target characterization on Nehalem

Runtime in seconds vs. Degree of parallelism

- toy 9M
- toy 6M
- macduff 25 50
- toy 3M
- sol33
- test_bitperm
- Bithand 1 22
- macduff 24 40
- linalg 1000000 100
- Bithand 1 21
- Bithand 1 20

BRISTOL
Target characterization on Magny-Cours

Runtime in seconds

Degree of parallelism

Runtime in seconds
Analogue benchmarks
Analogue benchmark examples

- Fhourstones
  - Something about Fhourstones
- Gups
  - Something about Gups
- Single_core
  - Something about single_core
Characterising analogues

For each target platform $T$
  For each analogue benchmark $A$
    For each parameterisation $P$
      $A_T(P)$ - run $A$ on $T$ with parameters $P$
      Record low-level hardware metrics for $A_T(P)$
Fhourstones on Nehalem
Fhourstones on Magny-Cours
Gups on Magny-Cours
Gups runtimes

Runtime in seconds

- Nehalem
- Magny-Cours
Effects of parallelism on sweeps

Runtime in seconds

- **P = 1**
- **P = 2**
- **P = 8**

Fhourstones sweep on Nehalem
Effects of parallelism on sweeps

Runtime in seconds

Gups sweep on Magny-Cours
Effects of parallelism on sweeps

Runtime in seconds

- P = 1
- P = 2
- P = 24

Single_core sweep on Magny-Cours
Assembling analogue benchmark suites
Exploration of analogue benchmark suite methodologies

There are a number of different ways we can construct an analogue job mix that models a target job mix:

1. Manually select one analogue to represent one target
2. Automatic point-wise construction of a composite analogue job mix
3. Residual refinement of a composite analogue job mix
Method 1: hand selection of analogues for targets (Nehalem)
The derived benchmark suite was assessed by comparing the breakdown of the percentage of CPU time for each analogue over a nine hour extended run of the analogue benchmark job mix.
Method 1 results summary

• Across the whole target and analogue job mixes, the differences in the runtimes represent an 8.2% overestimate on Nehalem and a 1.3% overestimate on Magny-Cours.

• In summary, the parameterization was successful, delivering analogue benchmark job mixes that estimate performance to within 10% overall of the target benchmark job mix.
Method 2

Automatic point-wise construction of a composite analogue job mix

• In this approach we took an aggregate view of the target job mix, using the Oprofile metrics averaged across all the target benchmarks across all the platforms. This gives a set of thirty metrics which describe the aggregate ‘shape’ of the target job mix across all platforms.

• We then took the sweep dataset previously constructed by running each analogue benchmark with a range of different input parameters across all the platforms, resulting in 141 data points, each with the same thirty metrics as the aggregate for the target job mix.

• An analogue job mix was then constructed by creating a pool of 100 job slots which were incrementally tested against all of the 141 analogues to determine whether a better least-squares fit would result from the replacement of the previous job by the new one. An iteration involved testing each of the job slots on turn.

• The stopping criteria were that are either the number of iterations exceeded 1000 or the difference in the least squares error between iterations falls below 0.01.
Method 2 results

The analogue job mix constructed using this algorithm converged quickly, taking just seven passes to meet the stopping criteria. The convergence of the least-squares error is shown above.
Method 2 results (Nehalem)
Method 2 results (Nehalem)

- Cycles/Inst: 3.2%
- DTLB misses: 3.2%
- LLC misses: 3.2%
- L2 lines in: 3.2%
- L1D lines in: 10.9%
- Branches: -26.0%
- Mispredicted: 98.6%
- L1I misses: 39.6%
Method 2 results (Nehalem)

• The method 2 analogue job mix has been constructed in a way that makes it behave very closely to that of the target job mix, as measured by the Oprofile hardware metrics.
• All of the most important metrics show errors of less than 4% between the target and analogue job mixes.
• Larger errors are limited to the less important metrics that have lower sampling rates.
• With this approach the performance across the whole analogue job mix is indicated by the cycles per instruction metric.
Method 3

Residual refinement of a composite analogue job mix

- An aggregate view of the target job mix is taken using the Oprofile metrics summarised across all the target benchmarks and across all the platforms.
- We then attempt to iteratively build a representative workload using as our measure of fitness the least squares difference between the average metric scores of the new job mix with those of the target job mix.
- This method differs from method 2 in that job slots are filled incrementally. Thus the algorithm starts from an empty job mix and incrementally looks for analogues which reduce the residual error.
- 10.3.3. The error for the first 100 slots is show in Figure 44 below. The x axis represents the job slot.
Method 3 results

The least squares difference rapidly converges to around 11.5 and by the time we have used 5000 job slots this has only dropped to 11.4. Thus by the time we have around 100 jobs we have a good analogue job mix that should closely match the behavior of the target job mix.
Method 3 results (Nehalem)
Method 3 results (Nehalem)

- Cycles/Inst: 4.2%
- DTLB misses: -3.4%
- LLC misses: 11.2%
- L2 lines in: 20.9%
- L1D lines in: 12.7%
- Branches: -26.3%
- Mispredicted: 94.1%
- L1I misses: 39.9%
Method 3 results (Nehalem)

- One can see from these results that this new analogue job mix also behaves in a manner very close to that of the target job mix, as measured by the Oprofile metrics.
- The first two most important metrics show errors of less than 5% between the target and analogue job mixes.
- Again, less important metrics with lower sampling rates show larger errors.
- Method 3 did not prove as accurate as method 2, with larger errors in general.
- As with the previous new approach, the performance across the whole analogue job mix is indicated by the cycles per instruction metric.
Comparison of the three methods

- Method 1: hand selection
- Method 2: pointwise construction
- Method 3: residual refinement

The chart compares various metrics across different methods, including:
- Cycles/Inst
- DTLB misses
- LLC misses
- L2 lines in
- L1D lines in
- Branches
- Mispredicted
- L1 misses
Conclusions

• The results show that these approaches are useful, producing analogue benchmark job mixes that are within 10-15% of the runtime of the target benchmark job mix across a range of different platforms.

• We found that the optimal mix of analogue benchmarks varied by platform, suggesting caution is needed when interpreting the results when using the analogue benchmark job mixes.

• Results also showed that we lacked an analogue that exhibited significant branch mispredictions compared to the target benchmarks.

• These techniques can be useful in abstracting away from classified codes.