Metrics for System Investigation

Vojtěch Horký Peter Libič Petr Tůma

2010 – 2021

This work is licensed under a "CC BY-NC-SA 3.0" license. Created to support the Charles University Performance Evaluation lecture. See `http://d3s.mff.cuni.cz/teaching/performance-evaluation` for details.

Contents

1 Overview

System Investigation

Measuring for the purpose of understanding system behavior.

Requirements:
- Directly related to specific system components.
- Configurable to fit variety of investigated systems.
- Reasonably simple to measure during development or operation.

Pitfalls:
- Metric design often influenced by what we can measure.
- Behavior of specific components may be difficult to isolate.
- Relationship to practically observed performance questionable.

2 Intermezzo: Out Of Order Execution

Current Processor Characteristics

Pipelined
Multiple instructions processed at different execution stages.

Superscalar
Multiple instructions dispatched simultaneously to multiple execution units.

Out Of Order Processing
Instructions scheduled for execution and retired based on dependencies.

Speculative Program Execution
Instructions may be executed based on speculation about future state.

[1]

> The paper by Abel et al.: uops.info: Characterizing Latency, Throughput, and Port Usage … doi:10.1145/3297858.3304062 investigates the latency, throughput, and port usage of instructions on Intel processors. The paper introduces relevant metric definitions and describes algorithms used for evaluating individual instructions, with results for many Intel processor architectures available at `http://www.uops.info`. Particularly interesting is the algorithm used to measure port usage, which mixes the instruction under analysis with other instructions known to block particular port combinations to determine exact port usage.

3 Intermezzo: Branch Prediction

Branch Prediction

Condition Prediction
Trying to guess whether a conditional jump will jump or not.

- Concerns most loops and branches in source.
- Short branches also done with conditional instructions.

Target Prediction
Trying to guess where an indirect jump will jump.

- Concerns all virtual method invocations in source.
- Concerns all return statements in source.
- Concerns some switch statements.

Static Branch Prediction

Static Prediction
Predicting without knowledge of past behavior.

Not much can be done:
- Forward jumps predicted as not taken.
- Backward jumps predicted as taken.
- Guess why ?

Prediction With Counters

Single Bit
Remember last state as taken or not taken. Predict same behavior as last time.

- Works for loops with many iterations.
- Poor for many common patterns.

Saturating Two Bits
Use saturating counter that increments vs decrements depending on branch being taken vs not taken. Predict behavior depending on counter value.

- Still poor for many common patterns.

[1] General terminology may not fit when applied on particular processor.

Prediction With History

History

Remember recent history as string of taken or not taken bits. Use history as index to table of saturating counters.

- We already have a hash table of counters anyway.
- Fixes behavior with short patterns that break counters alone.
- History either local for one branch or global across all branches.

Branch Target Buffer

One Target

Simply store last branch target in hash table.

- Not very good with polymorphic targets.
- Some benchmarks suggest success around half of the time.

More Targets

Store multiple targets indexed by history.

- History of past addresses or parts of those.
- Some benchmarks suggest global history better than local.

In Reality ?

Real designs mix more prediction principles.

Intel Sandy Bridge

- Two level predictor with 32 bits global history.
- Branch target buffer size probably around 4096 entries.
- Return target stack for up to 16 nested calls.

AMD Ryzen

- Hybrid predictor with perceptron.
  - Sounds arcane but in fact linear combination of selected history bits.
  - Of course many details are hidden in the training phase.
- Branch target buffer architecture and size not reported.
- Return target stack for up to 32 nested calls.

> The paper by Uzelac et al.: Experiment Flows and Microbenchmarks … doi:10.1109/ISPASS.2009.4919652 investigates the architecture of the branch predictors on a selected Intel processor. The paper describes a number of algorithms used for evaluating individual branch predictor components and documents the discovered branch predictor architecture.

4 Processor Behavior Metrics

Processor Behavior Metrics: Processor View

Overview

Metrics characterising application execution effectivity:

- Instructions per cycle (IPC or inverse CPI).
- Branch prediction hit (miss) count or rate.
- Memory accesses per instruction.
- …

Metric properties:

- Useful for example to appraise code optimisations.
- Typically very much platform specific.

Processor Behavior Metrics: Application View

Overview

Metrics characterising application execution demands:

- Instruction mix in general terms.
- Average lifetime of register values.
- General predictability of branch instructions.
- …

Metric properties:

- Very hard to define meaningful metrics and values.
- Platform independent measurement possible. `http://boegel.kejo.be/ELIS/mica`

5 Intermezzo: Memory Hierarchy

Memory Hierarchy Features

Translation Caching

Adress translation caches remember recent virtual to physical mappings.

Content Caching

Content caches remember recent data and hold recent writes.

Prefetching

Regular access patterns trigger prefetching.

Coherency

Single memory illusion maintained.

Look at the paper by Hackenberg et al.: Comparing Cache Architectures … doi:10.1145/1669112.1669165. Look at the paper by Molka et al.: Cache Coherence Protocol and Memory Performance … doi:10.1109/ICPP.2015.83.
The paper by Abel et al.: Reverse Engineering of Cache Replacement Policies … doi:10.1109/ISPASS.2014.6844475 documents the cache replacement policy of two selected Intel processors and points out that the observations fit partially randomized policies.
The paper by Vila et al.: CacheQuery: Learning Replacement Policies … doi:10.1145/3385412.3386008 reverse engineers the cache replacement policies of three selected Intel processors using automata learning techniques.
The paper by Maurice et al.: Reverse Engineering Intel Last-Level Cache … doi:10.1007/978-3-319-26362-5_3 reverse engineers the last level cache indexing functions for multiple Intel processors using hardware performance counters.

6 Memory Related Behavior Metrics

Cache Relevant Behavior Metrics

Overview

Metrics characterize memory access patterns.

- Cache misses (hits) per memory access (rate).

- Individually for each cache level.
- Also for address translation caches.

- Stack (reuse) distance. Number of accesses to unique addresses between reuses of the same address.
- Average memory access time usually in clock cycles. $T_{avg} = p_{hit} \cdot T_{cache} + (1 - p_{hit}) \cdot T_{memory}$

Metric properties:

- Depends on many platform properties (timing, prefetching, replacement strategies).
- Can guide application specific optimizations (data layout modifications, tiling, compute to fetch ratio).

Allocation Behavior Metrics

Overview

Metrics characterize dynamic (heap) memory allocation patterns.

- Allocation rate, deallocation rate. Should be the same, on average.
- Live size. Total size of usable (reachable) memory.
- Object lifetime. What time elapses between object allocation and deallocation (becoming unreachable). Time unit is usually a byte allocated or an object allocated.
- Object size.
- $Avg\ live\ size = Avg\ object\ size \cdot Avg\ object\ lifetime$

> Look at the paper by Hertz et al.: Generating Object Lifetime Traces … doi:10.1145/1133651.1133654. Examine Figure 12 for a heap profile visualization of an example benchmark workload. The X axis shows time in bytes allocated. The Y axis shows position in heap sorted by object age. The lines are similar to map contours, fired off in constant X axis steps, and thus indicate the volume of live objects remaining from particular program execution phases.
> Look at the paper by Lengauer et al.: Accurate and Efficient Object Tracing … doi:10.1145/2668930.2688037.