Evaluating the Impact of Dynamic Binary Translation Systems on Hardware Cache Performance

Arkaitz Ruiz-Alvarez
Kim Hazelwood
University of Virginia

IISWC’08
Dynamic Binary Translators

- Purpose: program analysis and modification
- Applications: security, program instrumentation, dynamic optimization
- DBTs: Dynamo, DynamoRIO, Pin, Strata, Valgrind
- DBTs translate and execute the binary image of a program
DBT Overview

- DBT changes the application’s:
  - code layout: JIT compiler + code cache
  - instructions:
    - DBT overhead + bookkeeping ins.
    - 1 original ins. → multiple traces
Our Motivation

- Icache effects:
  - Application speedup under Dynamo
  - Test this effect on current generation of DBTs

- What is the impact of DBTs on:
  - the instruction/trace cache?
  - the unified L2 cache?
  - the locality of the application?
  - other structures of the microarchitecture?
  - overall benchmark performance?
Pin and DynamoRIO

- DBTs that offer an *instrumentation* API
- Both use a *JIT compiler* and store translated code in a *code cache*
- We run them on:
  - Pentium 4: 32-bit with a hardware trace cache
  - Xeon Core 2: 64-bit with a hardware instruction cache
- We use SPEC 2006 INT benchmarks
Experimental Methodology

- We use hardware performance counters (PAPI and perfex) and simulation
- Measurements:
  - Running time: processor cycles
  - Instructions executed
  - L1 instruction/trace cache accesses, misses
  - L1 data cache accesses, misses
  - L2 unified cache accesses, misses
  - Branch prediction
  - Locality
- Graphs:
  - Error bars show variability across benchmark inputs
  - Benchmarks are ordered by Pin’s performance, fastest to slowest
Linux x86 32-bit Pentium 4 Benchmark Running Time

Processor cycles normalized to native run

- **Pin**
- **DynamoRIO**

Benchmark applications: mcf, libquan, bzip2, hmmer, astar, h264, sjeng, go, omnetp, gcc, xalanc, perl, INT
Execution Time

- Several benchmarks show near native performance:
  - DynamoRIO: mcf, libquantum, bzip2, astar and hmmr.
  - Pin: mcf, libquantum, and bzip2

- We analyze the performance of the instruction/trace cache and other structures of the microarchitecture.
Linux x86 32-bit Pentium 4 Trace Cache

Normalized Trace Cache Misses and Accesses

- Pin, Accesses
- Pin, Misses
- DynamoRIO, Accesses
- DynamoRIO, Misses

Graph shows normalized trace cache misses and accesses for various benchmarks on Linux x86 32-bit Pentium 4 processor.
Pentium 4 Trace Cache

- 2.5x more misses for Pin and 1.7x more for DynamoRIO
- Only *libquantum* and *hmmer* under DynamoRIO improve performance
- Performance can get worse:
  - If we add instrumentation
  - If space for the code cache is limited
- Misses are equally distributed over time for most of the benchmarks
Linux x86 64-bit Xeon Core 2 Instruction Fetch Unit Stalls

- Cycles With IFU Stalled

- Native
- Pin

- Graph showing the percentage of cycles with IFU stalled for various applications, with error bars indicating variability.
Xeon Core 2 Instruction Cache

- No single benchmark improves under Pin:
  - Normalized miss count
  - Cycles in which IFU is stalled
- Poor benchmark performance $\rightarrow$ poor instruction fetch unit performance
- Pentium 4 (12K-uop) vs. Xeon Core 2 (32 Kb) miss counts:
  - Both show performance degradation
  - Instruction cache degradation is lower on Xeon Core 2 than on Pentium 4
Instruction/Trace Cache

- Benchmarks with good performance may show a significant increase in trace/instruction cache misses
  - Poor code layout?
  - Bigger memory footprint?
  - Greater number of executed instructions?
Linux x86 64-bit Pentium Xeon Core 2 Pin L1 Data Cache

Normalized L1 Data Cache

L1D Accesses
L1D Misses
L1D Cycles miss outstanding

mcf  astar  bzip2  libquan  hmmr  omntp  gcc  sjeng  h264  go  xalanc  perl  INT
L1 Data Cache

- Pin does not incur additional overhead in many benchmarks
- Average miss count increase is 8.3%
- Average increase on cycles with outstanding misses is 2%
- L1 data effects:
  - More accesses to the data due to Pin’s data structures
  - Number of misses similar to native run
  - Pin’s accesses to data show very high locality
Linux x86 32-bit Pentium 4 Level 2 Cache

L2 Cache Accesses and Misses

- Pin, Accesses
- Pin, Misses
- DynamoRIO, Accesses
- DynamoRIO, Misses

Applications:
- mcf
- libquan
- bzip2
- hmmmer
- astar
- h264
- sjeng
- go
- omnetpp
- gcc
- xalanc
- perl
- INT
Level 2 Cache

- Pentium 4: 20 to 25% additional misses for DynamoRIO and Pin
- Xeon Core 2: 8% for Pin
- Level 2 unified cache effects:
  - Additional pressure on the level 2 cache due to greater number of trace cache misses
  - No dramatic increase in number of misses
  - Code layout may be improved by DBTs
Linux x86 32-bit Pentium 4 Instructions Executed

Normalized Instructions Executed

- Pin
- DynamoRIO

Normalized Instructions Executed vs. Applications:
- mcf
- libquann
- bzip2
- hmmer
- astar
- h264
- sjeng
- go
- omnetp
- gcc
- xalanc
- perl
- INT
Instructions Executed

- Main factor that affects performance
- Benchmarks that performs well under DBTs
  - Small binary image $\rightarrow$ less JIT compilation
  - Low # of indirect branches $\rightarrow$ no need to resolve indirect branches frequently
  - Long running time $\rightarrow$ amortize the compilation time by using a code cache
- Benchmarks that perform close to native require light intervention from the DBT (less compilation, less bookkeeping overhead).
Conclusions

- DBTs effect on the microarchitecture (compared to native execution):
  - Hardware trace caches show a significant performance degradation (170% to 248%)
  - Hardware instruction caches also show a negative impact
  - The level 1 data cache performs close to native
  - The level 2 cache shows a less dramatic increase in miss count (20% to 25%) than the L1 instrucion/trace cache

- In general, there are no *icache effects* for DBTs that focus on instrumentation

- The layout of the code cache is not responsible for poor cache performance

- Major factor affecting performance is the number of instructions executed