DR-BDI: Data Rearrangement for BDI Compression

Milestone 1 Report

Aditya Bhandaru

18-740 Computer Architecture, Mutlu Fall 2013

Overview

BDI compression shown to positively impact memory bandwidth and overall system performance.

Problem: Poor data compressibility or program insensitivity may not justify the access additional latency.

Observation: Compilers and memory allocators are microarchitecture unaware w.r.t BDI compression.

Idea: Create/transform applications to leverage the BDI microarchitecture using data rearrangement.

Background

- Memory wall
- Caching
  - The working set.
  - Conflict and capacity misses.
- Base-Delta-Immediate (BDI) compression.
  - Pekhimenko et. al. [cite]
  - Variable block size.
  - Increase cache capacity.

Motivation

- BDI has promise.
  - ~1.5 x cache capacity
  - 2.53 x less memory bandwidth
  - ~10% performance improvement (multicore)
- BDI must overcome these issues.
  - Latency of (de)compression (use in L2+).
  - Program not sensitive to cache size.
  - Poor compressibility (high data value range)

Idea

Rearrange the application data to place similar values adjacently in memory.

similar means:

- Same type ~ Static?
- Close values ~ Dynamic, profiling?

Increases likelihood of small deltas in cache lines -- higher compressibility.

Prior Work

- Dynamic Pool Allocation (pooling)
  - Small dedicated heap regions.
  - More flavors (split in compiler?)
- Forma
  - **Safe** splitting of aggregated data types
- Dynamic structure splitting
  - Runtime correctness check (page table/protection)

None of these specifically target BDI.

Initial Approach

Perform simple data data splitting and pooling.

Then look at bottlenecks in the BDI microarchitecture and optimize.

Drawbacks: As with prior work, we require changes to the compiler *and* the memory allocator.

Methodology

- Proof of concept.
  - Implement BDI, feed it memory traces, record hit-miss-evictions.
  - Manually rearrange allocation, record new traces. Look for improvements in hit-miss-evict data.
- If convincing, continue.
  - Generalize rearrangement algorithm (safe? correct?)
  - Automate rearrangement (script? compiler? memlib?)
  - Implement cycle accurate BDI.
  - Confirm actual performance improvements.
  - Run on benchmarks.

Progress

- Simple test suite (no rearrangement).
- Memory-trace recording infrastructure.
- Automated driver for input traces to cache stats output.
- BDI implemented in C++ (tricky-ish)

Coming Up

- Manually rearrange test programs.
- Gather data - hope for improved caching
- Hypothesis: definite improvement, but does it justify the drawbacks?

Acknowledgments

A thanks to Gennady Pekhimenko for directing me to many relevant papers with some overlapping fundamentals.

References

- Curial et. al., MPADS: Memory-Poolong-Assisted Data Splitting, ISMM’08
- Pekhimenko et. al., Base-Delta-Immediate Compression., PACT’12.
- Wang et. al., On Improving Heap Memory Layout by Dynamic Pool Allocation., CGO’12.
- Wang et. al., On the Fly Structure Splitting for Heap Object, ACM’12.