Enhanced Base-Delta Compression with Memory Pooling

Electrical & Computer ENGINEERING

Aditya Bhandaru, Gennady Pekhimenko, Onur Mutlu

Carnegie Mellon

Overview

- Base-Delta Compression [Pekhimenko et. al., PACT'12] proposes a promising technique for increasing on chip cache capacity using **compression**.
- **B+Δ** offers good compression but incurs an **additional access latency**.
- **B+Δ** suffers poor compressibility when adjacent data in memory have large value ranges.

- **Observation**: Traditional compilers and memory-allocators are unaware of **B+Δ** cache compression in hardware.
- **Key Idea**: Arrange data in memory to optimize B+Δ compressibility.
- **Solution**: Recent literature on Memory Pooling, Data Splitting [Curial et. al., ISMM'08] and related work seems promising.

Mechanisms

Basic Splitting-Pooling Example (64-bit)

Simple struct (a node perhaps)

FLAG (1B)	VAL (4B)	POINTER (8B)

In memory layout (high range in adjacent values)

...	0	100	0x77..0	1	108	0x77..4	0	93	0x77..C	...

After split-pool allocation (much lower range)

...	0	1	0	...	0x77..0	0x77..4	0x77..C	...	100	108	93	...

After **B+Δ** compression (huge space savings)

...	B0	+0	+1	+0	...	B0x77..0	+0x0	+0x4	+0xC	...	B100	+0	+8	-7	...

Proof of Concept Methodology

- To test the affect of splitting and pooling on **B+Δ** compression, we manually restructured programs for optimal data layout. (Later: implement pointer transformations in compiler)
- For this project, we focused on pointer based algorithms for benchmarks (**bisort** and **llu** – an apprx. for *Health*)

Effect of Pooling on B+Δ Compression Block Types

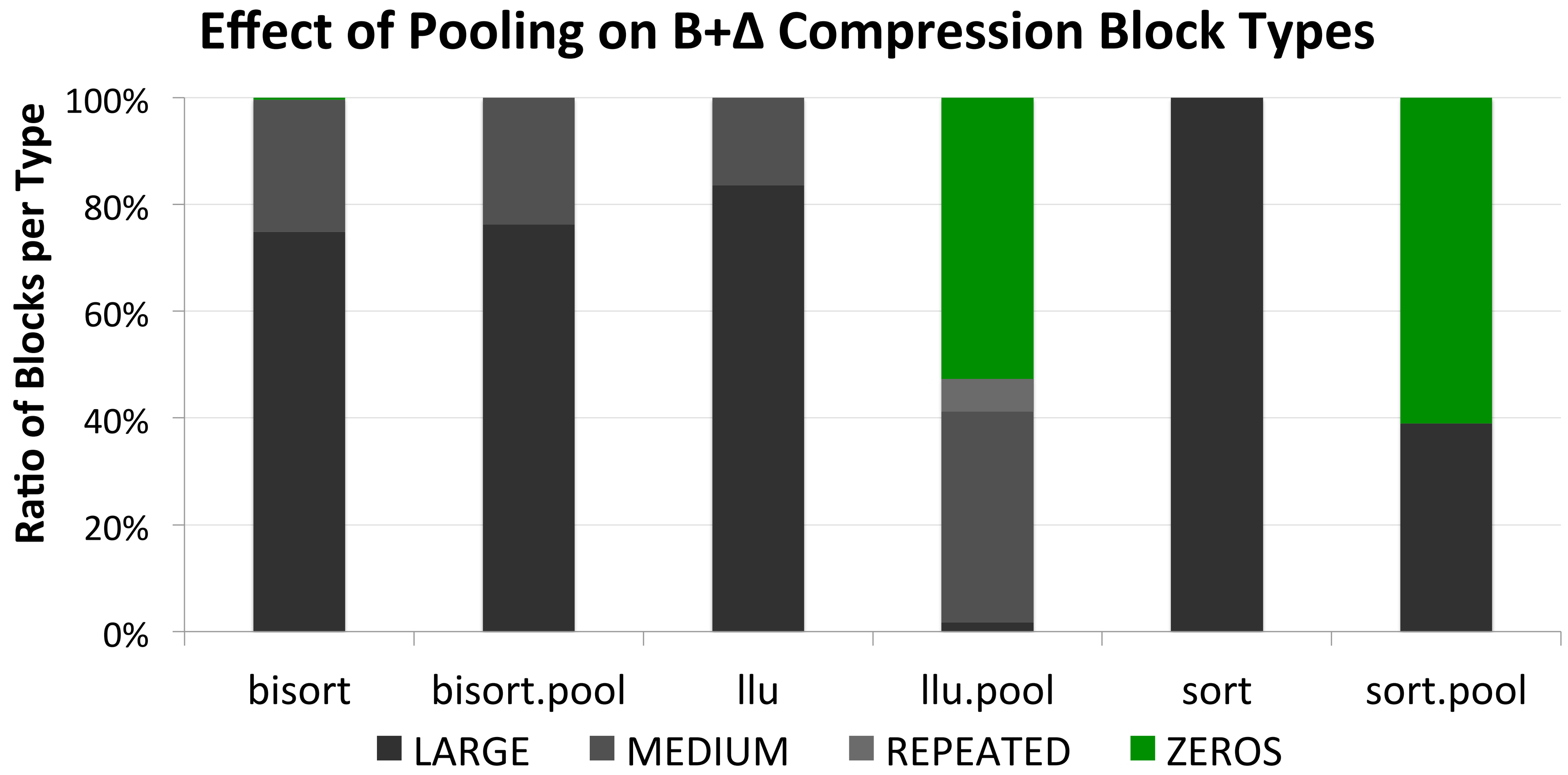


Figure 1. Each column shows the ratio of block-types for B+D compression with and without splitting and pooling. Notice the large increase in 1-byte all-zero blocks, and general decrease of large, uncompressed blocks. *(1 MB, 16-way, 32-BiB BΔ-Cache)*

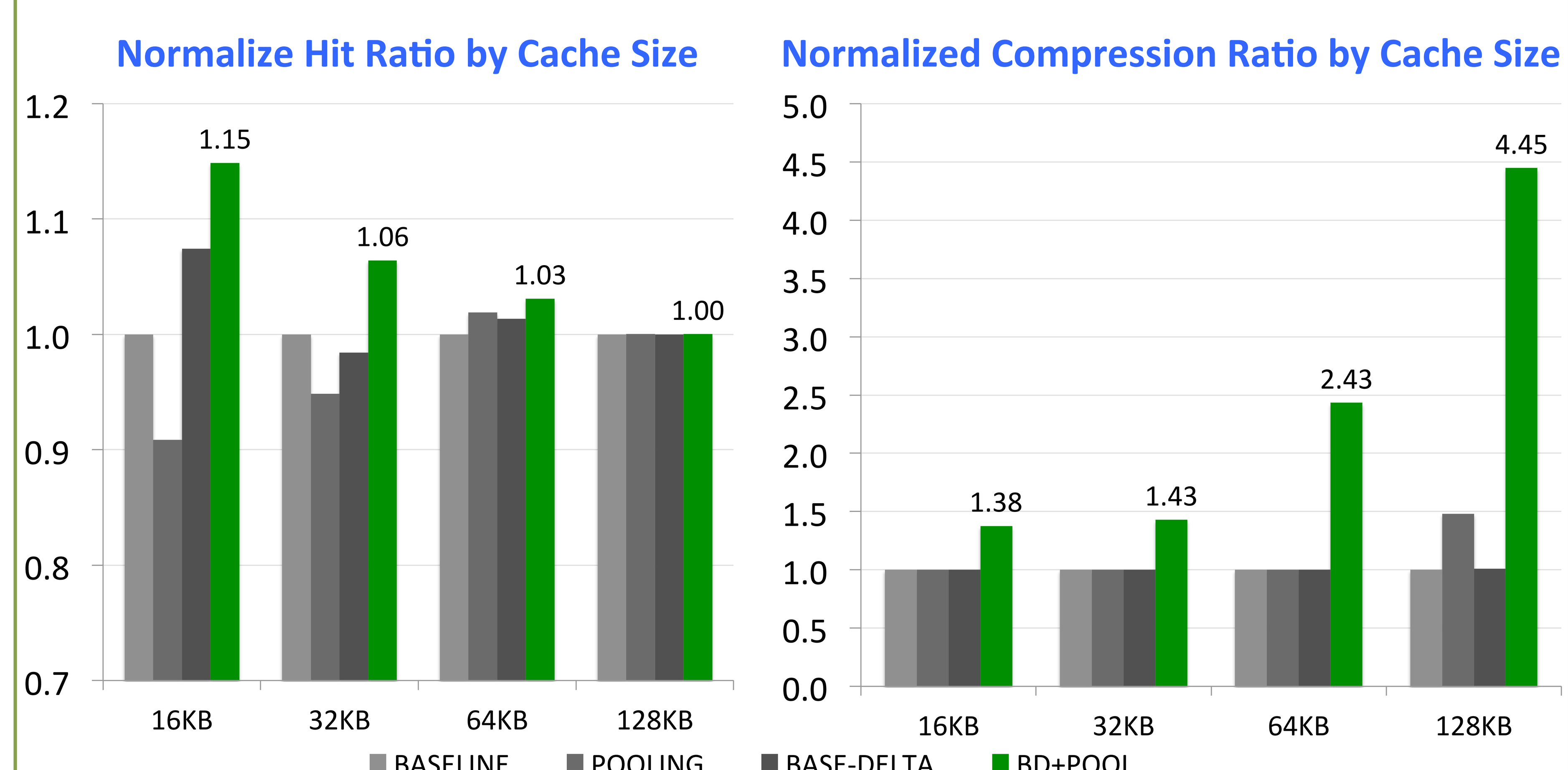
Motivation

Problem: Can we mitigate low compressibility cases for **B+Δ** compression?

- Increase viability for B+Δ implementation in hardware, and justify the extra access latency.
- Proposals like Memory Pooling and Data Splitting **already improve locality and reduce value range** in adjacent data values.
- But they have not yet been applied to **B+Δ**!

Results

Results for LLU micro-benchmark (working set ~117kb)



- **Improvement** in hit% from fewer evictions (more space)
- **B+Δ** alone reaches cache capacity for sizes **<** working set.
- **BΔ+POOL** still comes up with space savings!

- **Compression Ratio: 2.6x avg** (over LLU, TreeSort, ArraySort) **1.93x** over just *Split-Pool*, **2.47x** over only *B+Δ*
- **Hit Rate: 8% avg. increase over micro-benchmarks.**

- Pointer based algorithms had poor locality.
- Expect multithreaded apps benefit from compression too.

Conclusions

BΔ-POOL: Strong improvement over baseline, pooling and base-delta

- Just proof of concept*
- Makes single base version of **BΔI** more viable.

*Recall that splitting and pooling was done by hand. *Safely* splitting-pooling in the compiler is not always possible.

Further Work

- Implement pointer transformations in LLVM.
- Run benchmarks on cycle accurate BΔ-simulator.
- Incorporate data from standard benchmarks.
- Multithreaded environments.
- Interaction with non-traditional LRU policies.

See CARP by Huberty et. al.