Analyzing the Performance of ZFP Compressed Arrays on HPC Kernels

Pavlo Triantafyllides
Holcombe Department of Electrical and Computer Engineering
Clemson University
Clemson, South Carolina, USA
ptriant@clemson.edu

Jon C. Calhoun (Advisor)
Holcombe Department of Electrical and Computer Engineering
Clemson University
Clemson, South Carolina, USA
jonccal@clemson.edu

ABSTRACT

Per-node memory capacity limits the maximal problem size of HPC applications. Naïve data compression alleviates the memory impact, but requires full decompression before the data is accessed. ZFP compressed arrays reduce the memory footprint, by independently compressing data in fixed sized blocks. Thus, decompressing individual blocks and caching them enables random access and a reduction in decompressions on the critical path. The performance of ZFP compressed arrays is dependent on several key variables: software cache size, cache policy, and compression rate. In this poster, we explore the sensitivity of these ZFP parameters on runtime performance for the matrix-matrix multiplication algorithm. Results show that selection of cache size, policy, and rate yields 8% performance improvement over the default ZFP configuration.

ACM Reference Format:
Pavlo Triantafyllides and Jon C. Calhoun (Advisor). 2019. Analyzing the Performance of ZFP Compressed Arrays on HPC Kernels. In *Proceedings of Supercomputing '19: The International conference for High Performance Computing, Networking, Storage, and Analysis (SC '19).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Increasing the maximal problem size of HPC applications is limited by per-node memory availability. Data compression reduces the memory footprint occupied by HPC applications, but requires full decompression to access the data. Compressed arrays allow partial decompression of compressed data, requiring only the decompression of accessed elements. We study ZFP lossy-compressed arrays [1] because they are the current state-of-the-art and are backed by a software cache designed to reduce the occurrence of compression and decompression operations on the critical path. Our STREAM [2] results (shown on poster) yield a 0.01 speedup when using ZFP arrays over C arrays, providing motivation for this study. This poster analyzes the performance of ZFP compressed arrays on sequential standard and tiled Matrix-Matrix Multiplication (MMM). This poster makes the following contributions:

- Looks at how the performance changes in response to variations in ZFP rate, software cache size, and software cache policy.
- Achieves a performance improvement of 8% using optimal cache size selection for Matrix-Matrix Multiplication.
- Identifies the optimal configuration for Matrix-Matrix Multiplication and Tiled Matrix-Matrix Multiplication.

2 ZFP COMPRESSED ARRAYS

Lossy compressed arrays are a d-dimensional data structure that uses lossy compression to reduce the memory footprint, while still supporting random access. ZFP compressed arrays [1] are the current state of the art lossy compressed array. ZFP arrays utilize fixed-rate blocking in order to support read and write random access. In order to maximize reuse of decompressed elements, ZFP utilizes a user-configurable software cache. The software cache can be initialized to a user specified size and is configured as a direct-mapped cache, a two-way skew-associative cache, or a two-way skew-associative cache with a simple hash function. When unspecified, the cache algorithm defaults to a direct-mapped cache that fits two layers of decompressed blocks. The cache size is specified in bytes and must be greater than or equal to the size of one 4^d block.

3 EXPERIMENTAL RESULTS

3.1 Testing Methodology

We run our experiments on Clemson University's Palmetto Cluster. The nodes we use have Intel Xeon Gold 6148 CPUs. The experiments we run are STREAM Version 5.1.0 [2] and standard and 32×32 tiled matrix-matrix multiplication (MMM). All experiments are run using C++ ZFP lossy compressed arrays and are compiled using GCC 4.8.1. The experiments consist of parameter sweeps on a 4GB stream of doubles and nxn matrices of size $n = 256, 384, 512, 1024$. The parameters varied are cache size, cache policy, and ZFP rate.

3.2 Impact of Software Cache Size

To study the impact of cache size on MMM performance, we run a paramater sweep across ZFP rates $4, 8, 16, 32$ and cache sizes between 2^8 and 2^{16} on standard and 32×32 tiled MMM using 2^d ZFP arrays as matrices. Figure 1 shows the performance of standard MMM with matrices of size 1024×1024 for each specified rate and cache size, where the default cache size for this problem is represented as the black dashed line. Figure 1 shows that an 8% improvement in performance is achieved by selecting a 20kB cache over the default 32kB cache for this problem. Figure 2 shows

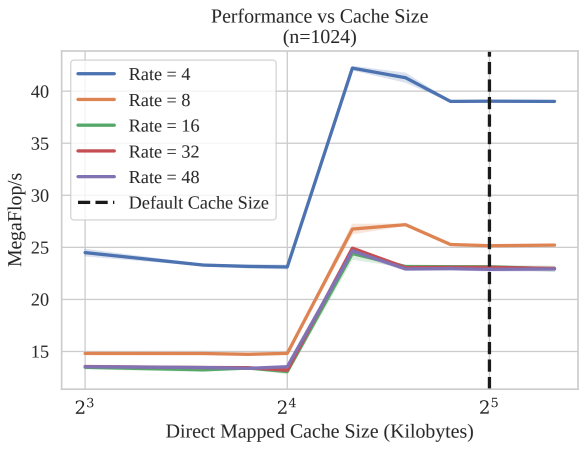

Figure 1: Standard Matrix-Matrix Multiplication Performance in response to changes in cache size.

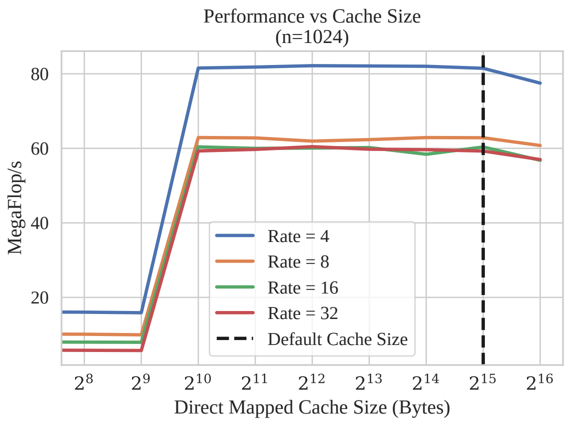

Figure 2: $32x32$ **Tiled Matrix-Matrix Multiplication Performance in response to changes in cache size.**

this data for the tiled MMM problem, where performance is up to 82.3 MegaFlop/s once the cache size exceeds the tile size, a 90% improvement over the peak standard MMM performance. For both of these problems, the smallest values for rate perform the best, indicating that reducing the rate improves performance. These results show that 20kB is the optimal cache size for 1024×1024 sequential MMM and the tile size is the optimal cache size for tiled MMM.

3.3 Impact of Software Cache Policy

We examine the impact of cache policy on MMM performance by running parameter sweeps on standard and tiled MMM for nxn matrices of size $n = 256, 384, 512, 1024$; rates $4, 8, 16, 32$; the default cache size; and each cache policy: direct-mapped, two-way skew-associative, and two-way skew-associative with simple hash. Figure 3 shows the speedup for each rate and matrix size using a

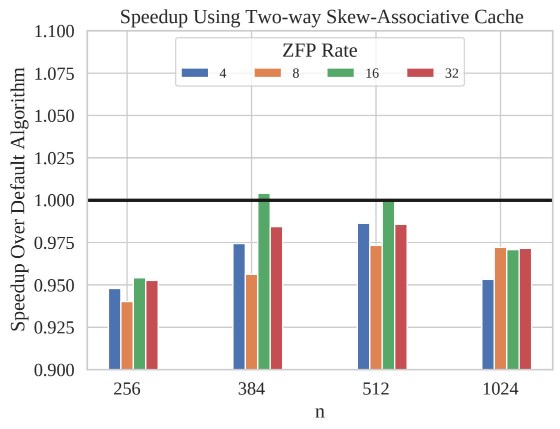

Figure 3: Matrix-Matrix Multiplication Speedup using Two-way Skew Associative Cache

two-way skew-associative cache over the default, direct-mapped, cache. This figure shows universal slowdown using a two-way skew-associative cache for all matrix sizes and rates other than rate 16, which provides negligible speedup for matrix sizes 384×384 and 512×512. These results show that the direct-mapped cache is the optimal cache policy for sequential MMM and tiled MMM (shown on poster).

4 CONCLUSION AND FUTURE WORK

This poster shows the performance variability of MMM using ZFP arrays in response to change in rate, cache size, and cache policy. In most cases, smaller rates yield better performance, cache size selection is dependent on problem and tile size, and the direct-mapped cache performs the best. For MMM with 1024×1024 matrices and a rate of 4, we achieve an 8% speedup by selecting 20kB cache over the default 32kB. We show that tiling performs better than standard MMM, but performance does not improve once the cache size exceeds the tile size. Finally, we show that the default cache policy performs better than the two-way skew associative cache for this problem. We will explore why this is true, and what problems perform best with alternative configurations in future work.

ACKNOWLEDGMENTS

Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197.

REFERENCES

[1] P. Lindstrom. 2014. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (Dec 2014), 2674–2683. https://doi.org/10.1109/TVCG.2014.2346458
[2] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.