Analyzing Software Cache Configuration for In-line Data Compression

SANSRITI RANJAN and JON C. CALHOUN (ADVISOR), Clemson University, USA

In order to compute on or analyze larger data sets, applications need access to large amounts of DRAM memory. To increase the size of memory requires a costly hardware upgrade. Compressing data structures stored in memory does not require hardware upgrades. Inline compression compresses and decompress data needed by the application as it moves out and into it's working set. Naive inline compression compresses and decompresses for each memory access which significantly hurts performance. Caching decompressed values in a software managed cache limits the number of compression/decompression operations. The structure of the cache impacts the performance of the application. In this poster, we build and utilize a compression cache simulator to analyze and simulate various cache configurations for an application. We evaluate both direct-mapped and set-associative caches on five HPC kernels. Results show that as the cache size increases, the hit rate increases. Further, we are able to tune the cache policy and improve the hit rate by 5.3 percent for two-way set associative caches and by 7 percent for four-way set associative caches.

CCS Concepts: • **Computer systems organization → Architectures**.

Additional Key Words and Phrases: data compression, software cache, hit rate, multi-dimensional data

ACM Reference Format:
Sansriti Ranjan and Jon C. Calhoun (Advisor). 2021. Analyzing Software Cache Configuration for In-line Data Compression. 1, 1 (October 2021), 3 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Large HPC applications need access to large amounts of DRAM memory[1]. While compression helps in reducing the storage requirements for the application, naive inline compression/decompression operation for each memory access hurts the performance of the application. Caching decompressed values in a software cache limits the number of compression operations. However, different applications require different caching mechanisms based on the computations for those applications. It is important to have a tool to simulate and analyze the appropriate cache configuration for an application. This enables optimum performance for the application. We built a compressed cache simulator that helps to analyze the performance of a cache based on different parameters and test on five HPC kernels.

This paper makes the following contributions:

- Built a cache simulator that tests five different applications and configures the appropriate cache structure for the application.
- Understand how varying the parameters cache size and cache policy impacts the performance of the application.
- Shows how spatial locality and random access patterns of applications impacts the cache configuration for the application.

Authors' address: Sansriti Ranjan, sansrir@clemson.edu; Jon C. Calhoun (Advisor), joncal@clemson.edu, Clemson University, 105 Sikes Hall, Clemson, South Carolina, USA, 29634.

2 BACKGROUND

Conventional software caches cannot fit multiple blocks of data in a single data entry or cache line. However, compressed caches using data compression are able to save space and fit large allocated arrays in cache blocks. Moreover, conventional caches fail to incorporate dimension of an application to store multi-dimensional arrays and improve performance. Here we incorporate the dimension of an application to generate the address trace before running the application with our simulator. This helps us store data in the appropriate cache line and analyze the performance for the cache configuration. The main applications implemented here are Matrix Multiplication, Jacobi, Fast Fourier Transform, Dijkstra and Sparse Matrix Vector Multiplication.

3 METHODOLOGY

Our Cache Simulator takes in compressed arrays from the five HPC applications and cache the blocks as required by the application to be able to calculate the cache hits and misses. This will enable us to compute the hit rate and assess the performance of the cache structure. Therefore, we first run the HPC applications and store the output comprising the compressed arrays and the blocks accessed in the application. Each of these arrays has N blocks, each block consisting of k elements. These arrays can be 1D, 2D or n-dimensions and will therefore have k^n elements for all N blocks. Moreover, using the address trace of the HPC application, we parse through the compressed arrays and for each block accessed, depending on a read/write action being performed, we load the array from the cache or store it. We simulate different cache structures (varying the cache size and cache policy) and analyze the performance.

4 EXPERIMENTAL RESULTS

The performance for different cache structures depends on the cache size, block size and cache policy. In our experiments, we tested the utility of a direct mapped, 2-way set associative and 4-way set associative cache. For each of these policies, we tested with cache sizes ranging from 0.5MB to 5MB for all the HPC kernels. Figure 1 shows the hit rate vs the cache size for the five kernels. We see that Matrix Multiplication has the highest hit rate and increases with an increase in cache size. On the other hand, we see that Dijkstra and SpMV applications' hit rate reduces with an increase in cache size. On further inspection, we noticed that low spatial locality along with aliasing leads to the degradation in performance. Therefore, an increase in cache size does not improve the hit rate. However, by increasing the cache policy as shown in Figure 2, the hit rate improves relatively for these applications. An improvement of 5.3 percent is seen on increasing the associativity to a two-way and 7 percent to a four-way for most of the applications while only 2 percent for Dijkstra and SpMV. An improvement in cache size improves the hit rate for all except Dijkstra and SpMV.

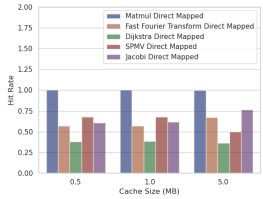

Fig. 1. Direct Mapped Cache

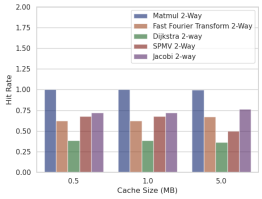

Fig. 2. Two-Way Set Associative Cache

5 CONCLUSION

The results show that increasing the cache size improves the hit rate and performance of an application provided it has high spatial locality and no random access patterns. Results further show that increasing the associativity or cache policy improves the hit rate further and can add to an improvement in the performance for applications with low spatial locality. Based on the application's access patterns and locality, the simulator helps to configure an optimum cache and gauge the improvement in performance of the application. Moreover, it helps to assess the trade-off between cache size and cache policy for applications with low spatial locality.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Jon C. Calhoun Donald Elmore. 2019. *Evaluating Lossy Compressors for Inline Compression, journal = "Supercomputing 2019".*