Implementation and Evaluation of GBDI Memory Compression Algorithm for Different Workloads

Sunil Venkatesh Rao
Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
snlvrao@vt.edu

***Abstract*—In this paper I have implemented and tested the Global Base Delta Immediate (GBDI) memory compression algorithm that was proposed in the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA). The correctness of my algorithm is tested by comparing my compression ratio against the results of the literature. I have then evaluated the compression ratio for different C and Java workloads.**

***Index Terms*—GBDI, memory, compression, benchmark**

I. Introduction

GBDI is a compression algorithm that builds upon the existing Base Delta Immediate (BDI) algorithm. The literature [1] offers substantially higher bandwidth of 1.5x and compression ratios of about 2.3x for SPEC2017 workloads. Unlike BDI, which uses separate bases for each of the memory blocks, baseline GBDI uses global bases that are established during a data analysis phase. Since there is offline analysis prior to online analysis, it is classified under the family of statistical compression techniques. Many optimizations need to be done along with the baseline GBDI to get good compression ratios.

II. Motivation

Most compression techniques are data specific and cannot be generalized. Some techniques are better for one type of data than others. Naturally, this means that to obtain good compression ratios, we have to understand the data patterns for different workloads to make the compression algorithm better. Since the literature has bench-marked for SPEC2017 workloads and found established good compression ratios, the natural question is how it performs against other type of workloads.

III. GBDI Architecture

Fig. 1 gives the overview of the GBDI algorithm. There are a total of 5 steps in the design of the compression engine.

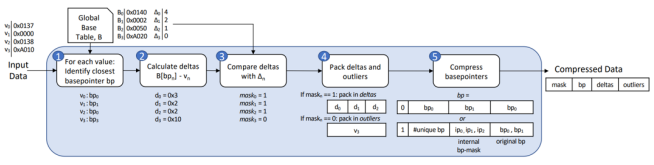

Fig. 1. Overview of the main steps in GBDI compression.

A. Global Base Table

Initially, all zero pages are removed from the memory dump as they are de-duplicated in memory and consume no bandwidth. 200k values are read for initial data analysis phase. Histogram binning is used to determine the global bases. N bins are chosen such that all 200k values are classified into one of the N equally spaced bins. Among the N bins, B bins are chosen that contain the most values and stored as global bases in a global base table. Empirical study from the literature show that a value of $B = 2048$ works the best. It is found that choosing a value of N is critical in getting good compression ratio. According to experimental analysis, $N = 2^{28}$ is found to yield the best compression ratio. A sensitive analysis is shown in the results section.

B. Closest Base Pointer and Delta Computation

For the online phase, individual memory blocks of 64B are considered. For each of the 16 words in a memory block, the closest base pointer and its corresponding delta are computed. Delta is computed as described in (1).

$$\Delta = ClosestBaseValue - Data \tag{1}$$

A maximum delta is established for each of the global bases by considering the maximum distance to the closest base values. It is important to note that an upper bound of $16 - \log_2(B)$ is chosen for delta to improve compression. This corresponds to 5 signed bits and therefore the delta values range from -16 to +15. It should also be noted that the closest base pointer is the index (0 to 2047) of the base table corresponding to the closest base value.

C. Comparing Deltas

The deltas of each of the 16 words are compared with the maximum delta corresponding to their closest base pointers. A mask is established for each word where mask bit is set to 1 if the delta is lesser than the corresponding base's max delta. Else, mask bit is set to 0. Thus, each compressed memory block needs to have a 16-bit mask array to indicate whether the corresponding word is compressed or not.

D. Pack Deltas and Outliers

If mask bit is 1, the value is considered as an inlier and is compressed using the closest base pointer and delta as shown in Fig. 1. If mask bit is 0, the value is left uncompressed and the full 32-bit value is stored in the compressed block. Since an upper bound of $16 - \log_2(B)$ is chosen to represent delta, each word can be compressed to $deltasize + ptrsize = 16 - \log_2(B) + \log_2(B) = 16$ bits.

E. Base Pointer Compression

Combinations of several optimization techniques are used to compress the size of base pointer array.

- If all the words inside a block are the same, the compressed block stores only one word along with its format encoder as represented by Fig. 2.
- $\#unique\_bp$ is determined for each block and if $\#unique\_bp \leq 4$ then the number is stored first followed by an internal base pointer mask for each of the inliers. Finally, the unique base pointers are also stored. However, if $\#unique\_bp > 4$, each of the base pointers are simply concatenated.
- Irrespective of above optimizations, each of the base pointers are also Huffman-encoded [2] if it is found to be beneficial. This greatly reduces the bits used from a constant 11 bits per inlier to variable bits and improves compression ratio. However, it is found that Huffman encoding quickly becomes ineffective with increasing number of bases. It could be seen that the Huffman encoder is effective for a maximum of 150 bases after which the number of bits in the encoded number exceeds the original number. A sensitive analysis is shown in the results section.
- Exploiting intra-block bases is a technique suggested in the paper which combines BDI with GBDI on a per block basis to select the best compression method for that block. This, however, shows negligible improvement according to the paper and has not been implemented.
- Prefix 0b10 identifies the special case where there are no outliers, while prefix 0b11 reflects the fixed delta format.

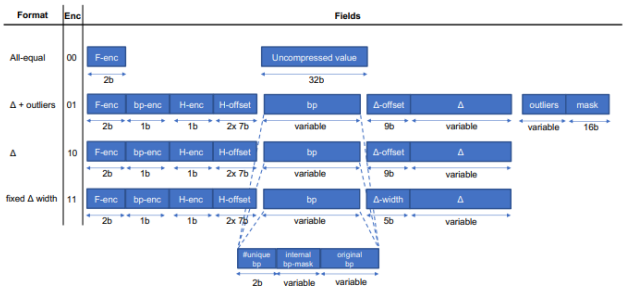

Fig. 2. GBDI compression formats.

IV. Experiment and Results

A. Histogram Binning

As portrayed in Fig. 3 below, an analysis of the histogram binning of 200k values of a SPEC2017 memory dump shows that majority of the values were concentrated at value zero.

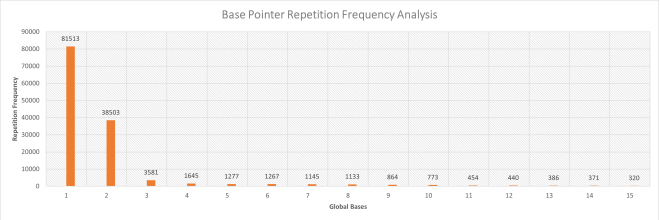

Fig. 3. Histogram binning of SPEC2017 memory dump.

B. Huffman Encoding

Table I shows the effectiveness of Huffman encoding. It can be seen that a base pointer 0 which took 11 bits originally, can now be represented using only 1 bit. A maximum of 150 base pointers out of a total of 2047 can be Huffman encoded before it becomes ineffective.

TABLE I
Huffman Encoding

Base Pointer	*Huffman Code*	*Bits Used*
0	1	1
1	01	2
2	00101	5
3	000111	6
4	000010	6
5	000001	6
6	0011110	7
7	0011101	7
8	0010001	7
9	0001100	7

C. N vs Compression Ratio

Fig. 4 represents a sensitivity analysis of the effect of number of bins on compression ratio. It can be seen that compression ratio is not exactly linear. This is because the size of meta data plays a big role in compressed data format and more the number of inliers need not always indicate the best compression ratio although it is the usual trend. It can be seen from Table II, the inlier count only starts to get larger than the outlier count at $N = 26$ bits. When $N = 15$ bits, the outlier data corresponds to 125MB of the original 179MB dump size whereas when $N = 26$ bits, outlier data only consumes 77MB in the compressed data block.

D. Expected vs Obtained CR

Fig. 5 shows the comparison of expected results as per the literature against what was actually obtained. It may be seen that results are quite close thus setting good grounds to test the algorithm for other C and Java workloads.

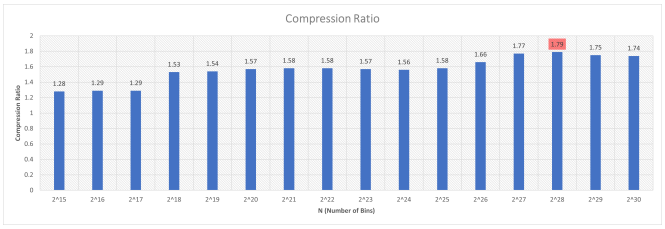

Fig. 4. Effect of number of bins on compression ratio of a SPEC2017 memory dump.

TABLE II
HUFFMAN ENCODING

No. of Bits	*Inlier Count*	*Outlier Count*
15	13978822	32949818 (125MB)
21	22088104	24840536
22	22278696	24649944
23	21754691	25173949
24	21221966	25706674
25	21973207	24955433
26	23874782	23053858[a]
27	26299537	20629103
28	26567826	20360814 (77MB)
29	26054602	20874038

[a] Inlier count is greater than outlier count.

E. Compression Ratios for Other Workloads

Fig. 6 shows the compression ratios for other C and Java workloads.

V. CONCLUSION

The average compression ratio for the Java workloads come to around 1.89x which is slightly lower than the average for SPEC2017 benchmark of 2.3x. However, there is good consistency in the results and the variations are little. It can therefore be inferred that the spatial locality of data plays an important role in data compression and that the data patterns in the different workloads tested could roughly be the same.

ACKNOWLEDGMENT

I take this opportunity to thank my professor, Dr. Xun Jian, for guiding me throughout the project. His lectures in Computer Architecture course were very informative and part reason for this successful project.

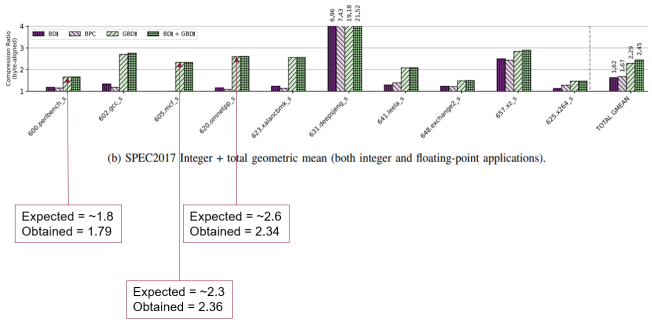

Fig. 5. Expected vs obtained CR for SPEC2017 memory dumps.

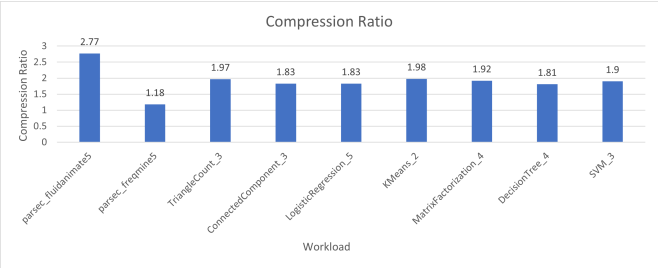

Fig. 6. Compression ratios for different workloads.

REFERENCES

[1] A. Angerd, A. Arelakis, V. Spiliopoulos, E. Sintorn and P. Stenström, "GBDI: Going Beyond Base-Delta-Immediate Compression with Global Bases," 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Korea, Republic of, 2022, pp. 1115-1127, doi: 10.1109/HPCA53966.2022.00085.

[2] D. A. Huffman, "A method for the construction of minimum-redundancy codes," Proceedings of the IRE, vol. 40, no. 9, pp. 1098–1101, 1952.