CSR&RV: an Efficient Value Compression Format for Sparse Matrix-Vector Multiplication

Junjun Yan, Xinhai Chen and Jie Liu

August 30, 2022

CSR&RV: An Efficient Value Compression Format for Sparse Matrix-vector Multiplication

Junjun Yan[1,2], Xinhai Chen[1,2] ⋆, and Jie Liu[1,2] ✉

[1] Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, 410073 China
[2] Laboratory of Software Engineering for Complex System, National University of Defense Technology, Changsha, 410073 China
`{yanjunjun,chenxinhai16,liujie}@nudt.edu.cn`

Abstract. Sparse Matrix-Vector Multiplication (SpMV) plays a critical role in many areas of science and engineering applications. The storage space of value array in general real sparse matrices accounts for costly. However, the existing compressed formats cannot balance the compressed rate and computational speed. To address this issue, we propose an efficient value compression format implemented by AVX512 instructions called Compressed Sparse Row and Repetition Value (CSR&RV). This format stores each different value once and uses the indexes array to store the position of values, which reduces the storage space by compressing the value array. We conduct a series of experiments on an Intel Xeon processor and compare it with five other formats in 30 real-world matrices. Experimental results show that CSR&RV can achieve a speedup up to 3.86× (1.66× on average) and a speedup up to 12.42× (3.12× on average) for single-core and multi-core throughput, respectively. Meanwhile, our format can reduce the memory space by 48.57% on average.

Keywords: Sparse Matrix-Vector Multiplication · Value Compression · Storage Format · AVX512.

1 Introduction

Sparse Matrix-Vector Multiplication (SpMV) is a kernel operation in many vital fields, such as parallel computing, scientific computation, and machine learning [1, 2]. The expression of SpMV is $Y \leftarrow A*X$, where A is a sparse matrix and both X and Y are dense vectors. There are some classic sparse matrix storage formats have been proposed. For example, the Coordinates (COO) and the Compressed Sparse Row (CSR) [1]. The former uses the triple form $(row, col, value)$ to store all nonzero elements (nnz) while the latter compresses the row array to the row_ptr array, which only stores the start and end index of each row.

Recent researches suggest that certain new features have arisen in the modern architecture of CPUs, for example, the increased number of cores and threads, the enhanced capacity of caches, and the improvement of Single Instruction Multi

⋆ Junjun Yan and Xinhai Chen contributed equally to this work.

Data (SIMD) units[8]. There is a growing body of literature that redesigns and optimizes the classic formats by using those new features [2]. We generalize into two primary parts: first, taking full advantage of SIMD instructions for vectorization and using blocking algorithms to improve the data locality [8, 10, 2]; second, compressing the storage space to reduce the memory access [4, 5, 7], which depends on the feature of matrices [1].

In some applications, the sparse matrices have numerous repetitive values, which instructs us to compress the repeated elements in the value array to reduce memory access times and improve the SpMV efficiency [2]. Until recently, there are only litter studies about compression and optimization of the value array: Kourtis et al. [5] proposed the index and value compression approach to saving the memory of the value array but still not efficient in modern CPUs. Grigoras et al. [4] use the same idea in FPGA but also do not suit CPUs architecture. Therefore, it is necessary to design a new format to solve those problems.

This paper proposes an efficient value compression storage format named CSR&RV. The proposed format only stores the repetitive values once and compresses the original value array to a non-repetition value array and an index array. For each nnz, it uses the index array to store the position of the value array and loads the value indirectly. This operation compresses the storage space of the value array because the number of non-repetition values is much less than the number of nnz. What's more, we can further compress the storage space of the index array by using uint8 and uint16 to store the indexes (index compression).

We compare the CSR&RV format to five other formats with different evaluating criteria. The experimental results show that CSR&RV has the highest throughput and the lowest memory space overhead. Compared to the MKL-CSR [9], the proposed format can get an average of 1.66× and 3.12× speedup in single-core and multi-core throughput, respectively. Compared to other state-of-art formats [9, 8, 10, 6], it achieves an average of 1.36× and 1.86× speedup in single-core and multi-core throughput, respectively. With index compression, our format reduces an average of 48.57% (maximum of 58.13%) memory space on matrices saved in CSR format.

2 The Compressed Sparse Row and Repetition Value Format

2.1 CSR&RV Representation

The CSR-based storage format is not efficient enough in the matrices have extensive repetitive elements. Because SpMV will access the repetitive values in the value array many times, which leads to the precious memory bandwidth wasted. To overcome this disadvantage, the proposed format, as shown in Fig. 1, compresses the repetitive values in *csr_ vals*. In CSR&RV, the *csrv_ values* only stores each unequal value once. And for each nnz, we use an index array called *csrv_ vals_ idx* to point to the location of the value array. Therefore, the memory access for values indirectly uses *csrv_ vals_ idx* as indexes. Because the number of non-repetition values is much small than nnz, the *csr_ vals* can be compressed to a same-length array *csrv_ vals_ idx* and a negligible array *csrv_ values*. By changing the type of *csrv_ vals_ idx* (e.g. uint8), we can save the space further.

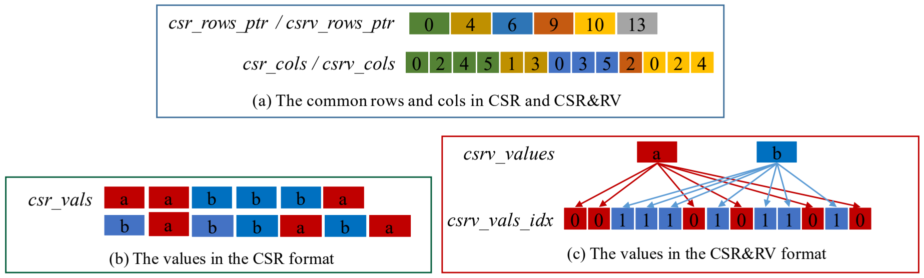

Fig. 1. The CSR&RV format.

2.2 SpMV Algorithm

Fig. 2 shows the SpMV method implemented by AVX512 instructions. The proposed method vectorize the data by rows which means the elements in the same rows will be packed together into several vectors. There are five main steps to accomplish once floating multiply and add operation, which is a fundamental operation in SpMV (the type of *val_ idx* array is uint8 as an instance):

(1) Loading eight indexes to the *_ val_ idx_ uint8* vector.
(2) Converting the indexes vector from uint8 to int32.
(3) Loading the columns from the *csrv_ cols* array.
(4) Gathering the vector operand *_ x* and *_ val*.
(5) Executing once floating multiply and add operation in the FMA units.

Looping above five steps until the computation of one row is accomplished. After that, we can use the *reduce_ add* instruction to sum the *_ y* vector horizontally and write back the result to the *y* array corresponding to the row rank. The write position in *y* is prefetched before writing back. Repeating the procession until all the rows are computed and stored to *y*.

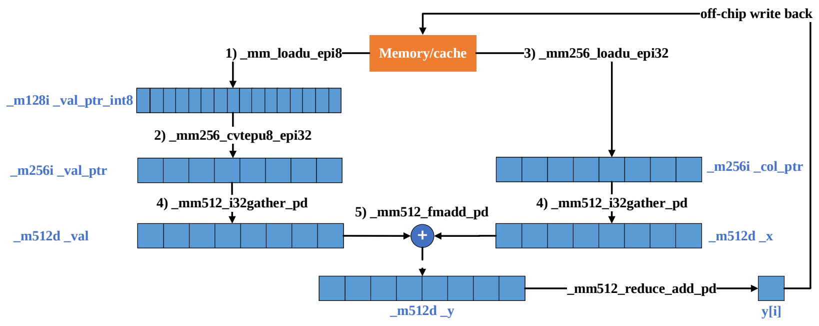

Fig. 2. The data flow in the SpMV algorithm.

3 Experimental Results

We compare our work with state-of-art open-source sparse matrix storage formats and SpMV algorithms. All the formats are implemented with the AVX512 instructions. The following 5 formats are compared: MKL-CSR [9], MKL-OPT (using the *mkl_ sparse_ optimize* function based on MKL-CSR), CSR5 [8], CVR [10], SPV8 [6]. We use 30 real-word sparse matrices downloaded from the SuiteSparse Matrix Collection [3]. Each matrix runs 1000 iterations and uses the single-iteration average time to evaluate. We run SpMV in double precision and record the run-time in different formats and threads (increased from 1 to 48). When evaluating the multi-core performance, we use the best run-time among all run-times. Normally, the maximal thread numbers (48) can get the best results.

3.1 Performance Comparison

Fig. 3(a) presents the single-core performance of different formats. Compared to the MKL-CSR, CSR&RV can achieve an average of 1.66× and a maximum of 3.86× speedup. Fig. 3(b) provides the multi-core performance. Compared to the best state-of-art formats, the proposed format can attain an average of 1.85× and a maximum of 7.92× speedup. These results suggest that the CSR&RV is better than other formats and the multi-core performance is better than the single-core. The possible reason for the phenomenon is that the optimization of arithmetic instructions can mainly influence the single-core performance because the bandwidth ability for one thread is enough. However, when considering multi-core, the increasing of thread numbers will gradually limit the bandwidth, which leads to the memory-access ability being the major influence and reflected by the storage space of formats.

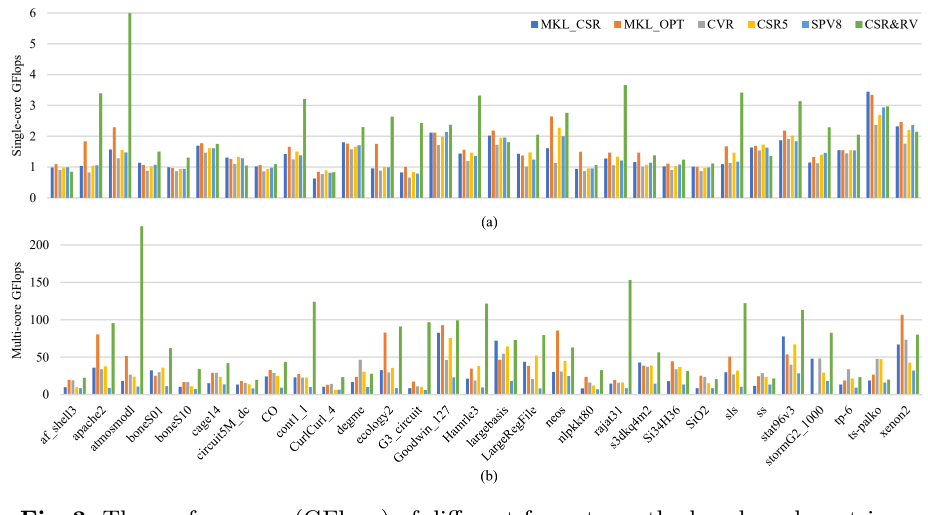

Fig. 3. The performance (GFlops) of different formats on the benchmark matrices.

3.2 Memory Overhead

Cause the memory space in the baseline formats is similar to the CSR format, we only compare our format with CSR. Fig. 4 shows the memory overhead of different benchmark matrices. The memory space token by CSR&RV is smaller than CSR in all matrices. This article calculated the memory-reduction rate as the reduced size of CSR&RV divided by the original size of CSR. Compared with CSR, the memory space in the CSR&RV format is reduced by 48.57% on average and 58.13% on maximum. By reducing the memory overhead, we can reduce the memory-access bandwidth, which makes SpMV more efficient.

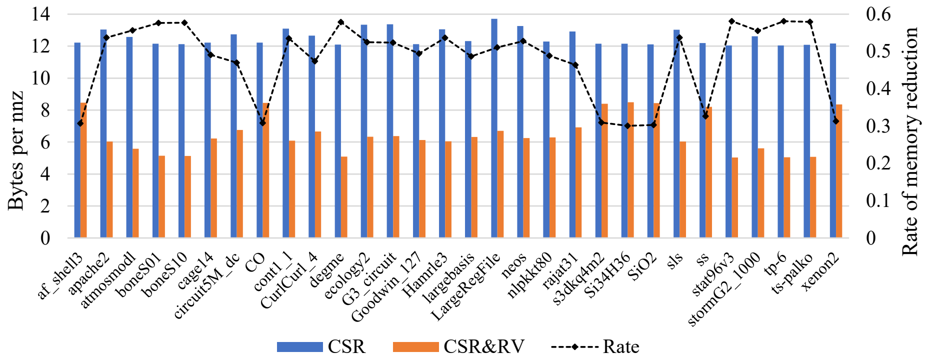

Fig. 4. Memory reduction in the CSR&RV format comparing to CSR.

3.3 Pre-processing

For the purpose of evaluating the practicability, this paper arranges all the tested matrices in the CSR format and measures the converting time from CSR to corresponding formats (except the MKL-CSR), which is called pre-processing in serval articles [7, 6, 10]. The pre-processing overhead is calculated as once single-core converting time divided by once single-core SpMV time. Table 1 shows that the pre-processing overhead in CSR&RV is the second lowest and only CVR is faster than it. This result indicates our format has the capability for actual use.

Table 1. The average processing overhead in different formats.

	CSR&RV	CSR5	MKL-OPT	SPV8	CVR
Processing overhead	3×	5×	10×	8×	2×

4 Conclusion

This paper proposes an efficient value compressed format named CSR&RV. The main idea of the proposed format is compressing the value array in CSR format to reduce the memory space and using AVX512 to improve the SpMV efficiency. We conduct a series of experiments on an Intel Xeon CPU and compare it with five state-of-art formats in 30 real-world sparse matrices. The experimental results show that CSR&RV can achieve the best throughput both in single-core and multi-core. Meanwhile, CSR&RV can reduce an average of about 50% (maximum of near 60%) of memory space compared to CSR. Moreover, this format has the advantage of being program-friendly and having low pre-processing overhead, which shows the potential to be employed in real-world applications. However, the CSR&RV format is mainly suited for sparse double-precision real matrices with many repetitive values. In future work, we will further extend our format to improve generality.

Acknowledgements This research work was supported in part by the National Key Research and Development Program of China (2021YFB0300101).

References

1. Barrett, R.: Templates for the solution of linear systems: building blocks for iterative methods. Software, environments, tools, SIAM, Society for Industrial and Applied Mathematics (1994)
2. Chen, X., Xie, P., Chi, L., Liu, J., Gong, C.: An efficient SIMD compression format for sparse matrix-vector multiplication. Concurr. Comput. Pract. Exp. **30**(23) (2018)
3. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011)
4. Grigoras, P., Burovskiy, P., Hung, E., Luk, W.: Accelerating SpMV on FPGAs by compressing nonzero values. In: 23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, Vancouver, BC, Canada, May 2-6, 2015. pp. 64–67. IEEE Computer Society (2015)
5. Kourtis, K., Goumas, G.I., Koziris, N.: Improving the performance of multithreaded sparse matrix-vector multiplication using index and value compression. In: 2008 International Conference on Parallel Processing, ICPP 2008, September 8-12, 2008, Portland, Oregon, USA. pp. 511–519. IEEE Computer Society (2008)
6. Li, C., Xia, T., Zhao, W., Zheng, N., Ren, P.: SpV8: Pursuing optimal vectorization and regular computation pattern in SpMV. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 661–666. IEEE (2021)
7. Li, Y., Xie, P., Chen, X., Liu, J., Yang, B., Li, S., Gong, C., Gan, X., Xu, H.: VBSF: a new storage format for SIMD sparse matrix-vector multiplication on modern processors. J. Supercomput. **76**(3), 2063–2081 (2020)
8. Liu, W., Vinter, B.: CSR5: an efficient storage format for cross-platform sparse matrix-vector multiplication. In: Proceedings of the 29th ACM on International Conference on Supercomputing, ICS'15, Irvine, CA, USA, June 08 - 11, 2015. pp. 339–350. ACM (2015)
9. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y.: Intel Math Kernel Library, pp. 167–188. Springer International Publishing, Cham (2014)
10. Xie, B., Zhan, J., Liu, X., Gao, W., Jia, Z., He, X., Zhang, L.: CVR: efficient vectorization of SpMV on x86 processors. In: Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018, Vösendorf / Vienna, Austria, February 24-28, 2018. pp. 149–162. ACM (2018)