Multi-level Blocking Optimization for Fast Sparse Matrix Vector Multiplication on GPUs

[Extended Abstract]

Yusuke Nagasaka
Tokyo Institute of Technology
Tokyo, Japan
nagasaka.y.aa@m.titech.ac.jp

Akira Nukada
Tokyo Institute of Technology
Tokyo, Japan
nukada@smg.is.titech.ac.jp

Satoshi Matsuoka
Tokyo Institute of Technology
Tokyo, Japan
matsu@is.titech.ac.jp

1. SPARSE MATRIX FORMATS

Design motivation of sparse matrix format is data compression and reducing calculation. The popular sparse matrix formats have been CSR (Compressed Sparse Row) and COO (Coordinated). More recently, in order to achieve better performance in SpMV computations, various sparse matrix formats have been proposed for specific types of matrices or architectures. ELLPACK and JDS formats are suitable for many-core processors such as GPUs, since these formats allow for continuous streaming memory access. SELL-C-σ [1] was recently proposed to improve ELLPACK. SELL-C-σ sorts every σ rows by the number of non-zero elements per row, and then every C rows are converted to ELLPACK. The number of zeros filled in the ELLPACK format is largely reduced. Segmented SELL-C-σ and Non-Uniformly Segmented SELL-C-σ [2] were proposed to improve cache hit ratio due to random memory access to the input vector. The matrix is segmented along the column and each sub-matrix is converted to SELL-C-σ. While the formats improve the cache hit ratio, they generate additional contiguous memory access. BCCOO format in yaSpMV library [3] focuses on reducing memory access. BCCOO divides the matrix into blocks to reduce the access to column index, and store in COO format. In order to find best parameter such as block size, yaSpMV library provides auto-parameter tuning scheme. Although BCCOO formats improved the performance of SpMV, the problem about high memory bandwidth still remains because BCCOO is based on COO format, which require large amount of memory access.

2. PROPOSAL

We propose the state of the art sparse matrix format reducing memory access by compression of column index to improve SpMV performance on GPU. Adaptive Multi-level Blocking (AMB) format has three blocking optimization techniques. First level is column-wise segmentation limiting width to 64k columns in order to improve the locality of the access to the input vector. Second level is row-wise slice by converting each sub-matrix into SELL-C-σ. The slice which does not have non-zero element is removed and each column index is represented in 16-bit integer (`unsigned short`). Third level is blocking optimization technique. Blocking the column index to each row reduces not only memory usage of column index, but also divergence during iterations by unrolling. The block size is set from 1 to 8. Although the large block size may generate more zero filling to the array of value data, the memory usage of column index will be significantly reduced. In SpMV computation, we initialize the output vector first, and add the result of each row using `atomicAdd` (in case of single precision).

3. EVALUATION

We compare the performance of our approach optimized for SpMV computations to existing formats. All experiments are done in single precision. For our evaluation, we utilize the matrices from the University of Florida Sparse Matrix Collection. All selected matrices are positive definite and their row sizes are more than 64k. Our evaluations have been done on NVIDIA's Tesla K20X GPU installed on TSUBAME-KFC. The GPU has 6 GBytes of device memory, and its memory bandwidth is up to 250 GByte/sec. GPU codes are implemented in CUDA 7.0.

Our AMB format achieves significant speedup of x2.83 on maximum and x1.75 on average compared to NVIDIA's cuSparse (we use CSR, HYBRID and BSR format), and x1.38 on maximum and x1.08 on average compared to yaSpMV. We also evaluated the amount of memory access except the access to input vector. Since all the information about matrix is read once, we estimate the amount of memory access by counting the size of arrays. We also estimated the access to output vector. For the matrix which has small number of non-zero elements, the cost of atomicAdd is not negligible. We postulate one atomicAdd operation requires two memory accesses. As the evaluation, our AMB format reduces memory access 33% on maximum and 10% on average compared to the SpMV library and existing sparse formats.

4. CONCLUSIONS

We propose the novel sparse matrix formats which are designed for reducing the memory access in SpMV computation. AMB format achieves x1.38 on maximum and x1.06 on

average performance gain compared to existing fast SpMV libraries and other sparse matrix formats. For the future work, we devise auto-parameter tuning mechanism for AMB format.

5. REFERENCES

[1] Moritz Kreutzer et al. A unified sparse matrix data format for modern processors with wide SIMD units. *CoRR*, abs/1307.6209, 2013.

[2] Y. Nagasaka, A. Nukada, and S. Matsuoka. Cache-aware Sparse Matrix Formats for Kepler GPU. In *20th IEEE International Conference on Parallel and Distributed Systems*, 2014.

[3] S. Yan, C. Li, Y. Zhang, and H. Zhou. yaSpMV: Yet Another SpMV Framework on GPUs. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, 2014.