A Novel Variable-Blocking Representation for Efficient Sparse Matrix-Vector Multiply on GPUs

Tuowen Zhao, Tharindu Rusira, Khalid Ahmad, Mary Hall
School of Computing, University of Utah

Introduction

Fillrate-guided block compressed sparse row (FBCSR) is a novel approach to improve the performance of sparse matrix-vector multiply (SpMV) on GPUs. Motivated by the observation that in finite element method, many of the matrices consist of dense blocks of different sizes and unaligned starting positions, FBCSR can identify and extract those local nonzero patterns that could improve the memory access and reduce intra-warp divergence of the corresponding SpMV kernels. As compared to other variable blocking methods such as unaligned block compressed sparse row, it relies on local patterns and can generate larger blocks well-suited for single instruction, multiple threads processing, while also tolerating a bounded number of generated zero-fillins. We present the SpMV performance results for FBCSR on two generations of Nvidia GPUs, which shows that FBCSR outperforms available alternatives for the matrices to which it is applicable.

Contribution

- A new matrix representation - Fillrate-guided block compressed sparse row(FBCSR)
  - Designed to exploit local sparsity patterns of matrices well suited for some matrices in FEM applications
  - Could exploit SIMT execution model and offers better memory coalescing potentials on GPUs by defining the shapes to be extracted
  - Offers tuning possibilities that could be used for computation vs. memory tradeoff
- Experiments of SpMV kernels using *The University of Florida Sparse Matrix Collection* on two generations of Nvidia GPUs and comparing againsts Cusp's CSR and Hybrid implementations.
- Up to 85% faster against CSR and up to 918% faster against Hybrid on specific matrices.

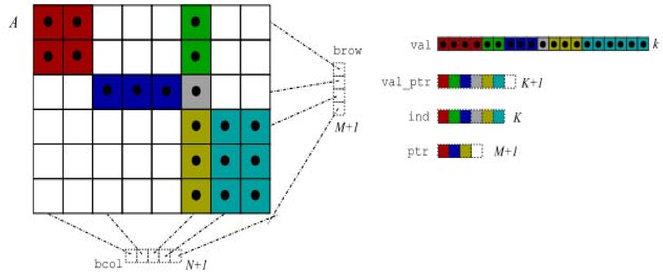
From CSR to UBCSR

Our new matrix representation is inspired by UBCSR[1], which tries to overcome some of the drawbacks of the BSR format and optimize for register usage on a CPU.

1. Using vector similarity to convert the matrix to VBR format. Split any two adjacent rows or columns if they are too dissimilar.

   VECTOR SIMILARITY = COMMON NON-ZERO/MAXIMUM NON-ZERO

2. Extract $r_i \times c_i$ blocks from the VBR format by taking as many adjacent $r_i \times c_i$ blocks as possible from any VBR blocks that is larger than $r_i \times c_i$ and put them in split A_i.
3. Rest of the nonzeros are put into later splits, or stored in CSR format in the end.



Strengths:

1. Variable blocking scheme improves fill-ins as standalone nonzeroes are more likely to be stored in CSR format instead of a blocked format.
2. Splitting the matrix using parameters present a simple heuristic allowing part of the matrix to be represented in a more suitable format.
3. Can use specialized kernels to improve the performance for different splits.

Weaknesses:

1. UBCSR requires VBR format as an intermediary
   a. Very small VBR blocks are restricting the size of UBCSR blocks and small blocks are inefficient for GPUs
   b. Row similarity is a global measure
2. No guarantee for fill-in ratio, might still generate some fill-ins
3. UBCSR shapes are limited to rectangles of different shapes. For example, they won't work on matrices that looks like a chess board.

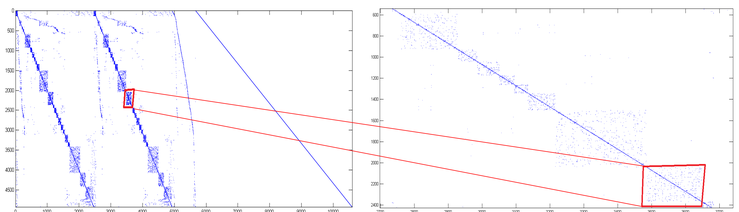
New Matrix Representation - FBCSR

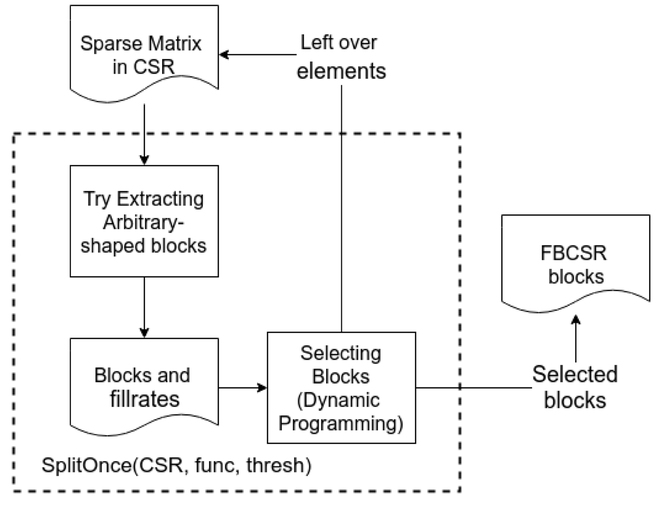
Fillrate-guided Block Compressed Sparse Row

1. Improves regularity in memory access pattern and reduce intra-warp divergence
2. Ensure that every block that extracted had a fillrate greater than a threshold

   `Fillrate = NumberOfNonZeros / TotalNumberOfElements`
3. Enabling arbitrarily shaped* boxes: row, column, zig-zag, diagonal, parallelogram …

   * Due to the Dynamic Programming algorithm, each row in a block shape is continuous and has the same number of elements.



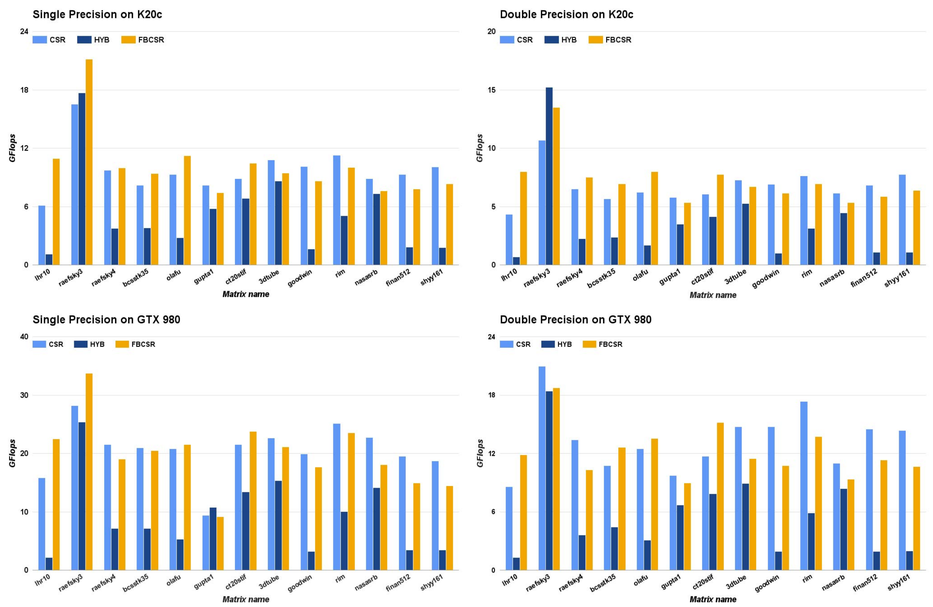


1. The `SplitOnce` algorithms is run multiple times to try different block shapes, and the remaining elements are put into CSR format.
2. Block shapes used: 32-element row, column, diagonal, and anti-diagonal, and also 32x32 square
3. Each block shape has its own threshold for extracting FBCSR blocks
4. Each block shape has its own optimized CUDA kernel

FBCSR - Analysis

1. The algorithm takes an arbitrary shape described by the shape function.
2. The extraction algorithm could ensure the overall fillrate is at least the threshold. This could ensure a bounded performance of the specific FBCSR kernel.
3. Let *NNZ* be the number of nonzeroes in a matrix, *n* be the number of rows of the matrix. The time and space complexity of the extraction algorithm are both O(NNZ+n), which is same as the input in CSR format.
4. Block shapes can be selected to better address different architectures.

Results



- The figures show the performance of SpMV kernels using our FBCSR implementation on two difference generations of Nvidia GPU. The K20c and 980 GTX. The CSR and Hybrid implementation are taken from Cusp[3].
- The matrices are taken from the *The University of Florida Sparse Matrix Collection*[2]. The graphs are showing the portion where the new format applies(have blocks in FBCSR), which is 13 out of the 31 matrices.
- Comparing to CSR format, we are able to achieve comparable and up to 85%(lhr10, double precision) better performance on K20c and up to 42%(lhr10, single precision) better performance on GTX 980.
- Comparing to Hybrid format, we are able to beat it in all but the raefsky3 matrix in double presision on the K20c, with up to 918% better performance(lhr10, single precision, GTX 980).
- We observed that the method exposed more tuning possibilities: Some of the kernels we choose are bandwidth bound on specific architecture and some are computation bound. We could leverage the threshold parameter to balance the usage of computation unit and memory unit.
- The performance of FBCSR is largely dictated by the gross number of local patterns exists in the matrices. The more local patterns exists in a matrix the higher the gain from adopting FBCSR.
- The increased bandwidth and computation power of future generations of GPU hardware will benefit this representation more as the coalesced memory access and reduced intra-warp divergence becomes more important.

Future Work

- Design better heuristics for choosing shapes and tuning the parameters of the representation.
- Apply this representation for computing SpMV on other architectures, such as applying smaller shapes and utilizing bitmasked registers[4] for computation on CPUs.
- Utilize compiler-based techniques in CHiLL[5] to improve data access patterns of other applications with sparse data using this representation.

References

[1] Vuduc, Richard Wilson. Automatic performance tuning of sparse matrix kernels. Diss. University of California, Berkeley, 2003.
[2] The University of Florida Sparse Matrix Collection, T. A. Davis and Y. Hu, ACM Transactions on Mathematical Software, Vol 38, Issue 1, 2011, pp 1:1 - 1:25. http://www.cise.ufl.edu/research/sparse/matrices.
[3] S. Dalton, N. Bell, L. Olson, and M. Garland, "Cusp: Generic parallel algorithms for sparse matrix and graph computations," 2014, version 0.5.0. [Online]. Available: http://cusplibrary.github.io/
[4] Buluc, Aydin, et al. "Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication." Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International. IEEE, 2011.
[5] Chen, Chun, Jacqueline Chame, and Mary Hall. CHiLL: A framework for composing high-level loop transformations. Technical Report 08-897, U. of Southern California, 2008.