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
Salt Lake City, UT
{*ztuowen,tharindu,khalid,mhall*}*@cs.utah.edu*

***Abstract*—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 block 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.**

***Keywords*-Sparse Matrix Representation, Sparse Matrix Computations, SpMV, GPU**

I. Introduction

Sparse matrix-vector multiply (SpMV) is a fundamental building block of numerous HPC applications, such as the finite element method (FEM). In FEM, many of the matrices have intricate sparsity structures that consist of dense blocks of different sizes and unaligned starting positions. For such matrices, standard sparse representations (i.e., Coordinate, CSR, ...) fail to make use of these special structures and, as a result, sometimes produce inefficient SpMV kernels. Specialized matrix representations such as the unaligned block compressed sparse row (UBCSR) [1] are introduced to recognize these structures and extract corresponding dense blocks. These methods aim at using the sparsity structures to guide the block extraction that could help improve the performance of the corresponding SpMV kernels on CPUs.

We observe that for single instruction, multiple threads (SIMT) architectures such as GPUs, coalescing memory access and reducing divergence in computation are key points in improving application performance; meanwhile, a small amount of redundant computations introduced by zero-fillins can be tolerated. These characteristics mean that only some access patterns are efficient for GPUs, and most of them are multiples of the warp sizes. So, local pattern-driven extraction of these larger structures could potentially work better on GPUs. Above-mentioned UBCSR is not pattern based, and its utilization of variable block row (VBR) greatly restricts its flexibility in the shapes and sizes of the block generated. In addition, previous pattern-based methods such as the Pattern-based Representation (PBR) [2] uses histograms to recognize the recurring patterns. However, the PBR algorithm doesn't allow for unaligned block starting positions and fails to identify larger blocks.

Limitations of the previous representations prompted us to look for a more flexible representation that could identify larger local patterns of nonzeroes in sparse matrices and also guarantee the ratio of the generated fillins. We designed our representation, fillrate-guided block compressed sparse row (FBCSR), to improve on previous implementations by using a novel variable blocking scheme to capitalize on the GPU. Our representation can spot and extract particular patterns occurring in a sparse matrix and allow for unaligned block starting positions to make it suitable for FEM applications.

II. FBCSR

Assuming we have the original matrix in compressed sparse row (CSR), its conversion to FBCSR is guided by the *fillrate*. This parameter is inspired by the fact that we could tolerate some amount of zero-fillins. Let nnz be the number of original nonzeroes in the block and $nelems$ be the number of elements in a block then *fillrate* is defined on each block as fr in Equation 1.

$$fr = nnz/nelems \tag{1}$$

FBCSR relies on matrix splitting and considers different block shape configurations for a different split of the matrix. The block shapes chosen will lead to efficient SpMV kernels on the GPU and are common enough that they are present on many of the matrices. FBCSR uses the *SplitOnce* algorithm shown in Figure 1 to generate the matrix splits and converts them into the FBCSR formats.

SplitOnce is an iterative algorithm. In each iteration, a block shape described by *func* and the threshold of fillrate, *thresh*, are used to extract the blocks from the sparse matrix in CSR format. First, it scans through the whole matrix looking for all the blocks in the specific shapes that

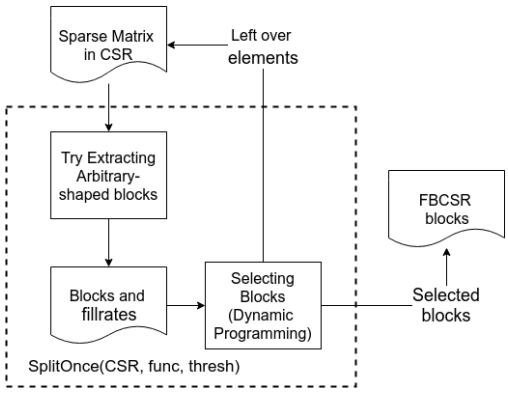

Figure 1. CSR-FBCSR conversion flowchart

have a fillrate larger than the threshold. Then, the dynamic programming algorithm will look at each row and select the blocks that could maximize the number of nonzeroes extracted from that row. A similar dynamic programming algorithm then performs the block row selection to maximize the number of nonzeroes extracted. Afterwards, all selected blocks are stored in FBCSR format for the split, and the remaining nonzeroes are stored in CSR format to be used as input of the next iteration.

The conversion from CSR to FBCSR has an overhead comparable to CSR. Assuming that each row of the block shape is continuous, let NNZ be the total number of nonzeroes in the original matrix, n be the number of rows, and E be the row span of the block shape. The scanning of the matrix into potential blocks and the block selection have a time complexity of $O(NNZ * E^2 + n)$. If we treat the block shapes as constants, the total time complexity is $O(NNZ + n)$, the same as CSR.

Other than input level of complexity, FBCSR will reduce the indexing overhead relative to the CSR format as our representation only needs one column reference for a block that consists of multiple elements. FBCSR also offeres several benefits over using fixed-size blocks (BCSR):

- Allow variable column and row starting position.
- Ensure that every extracted block has a fillrate,fr, greater than a threshold. This will ensure a bounded performance for those blocks in FBCSR.
- Enable arbitrarily shaped box described by the shape functions: row, column, diagonal, parallelogram ...
- Instead of using only one block shape, the heuristics in *SplitOnce* allow for combination and trade-off between different block shapes.

Comparing with UBCSR and PBR, FBCSR is able to identify and handle larger non-rectangular blocks while still can have varying block row and column starting positions.

III. Experiments & results

We tested FBCSR against CUSP's[3] CSR and Hybrid implementations of SpMV on two different generations of Nvidia GPUs, the K20c and 980 GTX. We tested our representation on some of the matrices from The University of Florida Matrix Collection[4] in both single and double precisions. The block shapes we used are 32-element column, row, diagonal, and anti-diagonal, and also 32×32 square. Out of the 31 matrices we tested, 13 of the matrices benefit from using FBCSR, such that some of the nonzeroes are in FBCSR format.

Compared to CSR format, we achieved comparable and up to 85% (lhr10, double precision) better performance on K20c and up to 42% (lhr10, single precision) better performance on GTX 980. Also, we are able to beat the Hybrid representation in all but the raefsky3 matrix in double precision on the K20c, with up to 918% better performance (lhr10, single precision, GTX 980). The increased bandwidth and computation power of future generations of GPU hardware will benefit this representation more as the coalesced memory access and the reduced intra-warp divergence become increasingly more important.

IV. Conclusion and Future Work

We observed that the FBCSR representation exposed more tuning possibilities: some of the kernels we choose are bandwidth bound on specific architectures and some are compute bound. We could leverage the threshold parameter to balance the usage of computation unit and memory unit and between introduced fillins and more coalescing. We could also potentially extends this model for computing SpMV on other architectures and incorporate compiler-based techniques in CHiLL[5] to automatically improve data access patterns of other application with sparse data.

References

[1] R. W. Vuduc and H.-J. Moon, "Fast sparse matrix-vector multiplication by exploiting variable block structure," in *International Conference on High Performance Computing and Communications*. Springer, 2005, pp. 807–816.

[2] M. Belgin, G. Back, and C. J. Ribbens, "Pattern-based sparse matrix representation for memory-efficient smvm kernels," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 100–109.

[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] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[5] C. Chen, J. Chame, and M. Hall, "Chill: A framework for composing high-level loop transformations," Citeseer, Tech. Rep., 2008.