Register-based Implementation of the Sparse General Matrix-Matrix Multiplication on GPUs

Junhong Liu*†, Xin He*†, Weifeng Liu‡, Guangming Tan*†
*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
†University of Chinese Academy of Sciences
‡Department of Computer Science, Norwegian University of Science and Technology
liujunhong@ncic.ac.cn,hexin2016@ict.ac.cn,weifeng.liu@ntnu.no,tgm@ict.ac.cn

Abstract

General sparse matrix-matrix multiplication (SpGEMM) is an essential building block in a number of applications. In our work, we fully utilize GPU registers and shared memory to implement an efficient and load balanced SpGEMM in comparison with the existing implementations.

CCS Concepts • **Computing methodologies** → **Parallel algorithms**;

1 Introduction

The motivation of this paper is to fully utilize GPU registers and shared memory to implement an efficient and load balanced SpGEMM. This work chooses the vertical-merge approach proposed in the paper [3] as an early baseline and the N-to-M product-thread binding strategy [4] to achieve the goal. In Table 1, we list the memory use, nonzero-to-thread mapping, and computing methods of the existing libraries and ours on GPUs. As Table 1 shows, our library is the first to use register and shared memory to implement SpGEMM. The other libraries are either shared memory and global memory or register and global memory.

2 The Proposed SpGEMM Algorithms

The matrix-matrix product $C = AB$ can be split into the basic computation work unit, i.e., the vector-matrix product $c = aB$ that computes one output row, where c and a are the corresponding rows of C and A. Different with all the existing libraries, in this paper we devise an adaptive N-to-M product-thread binding vertical merge method, where N represents the number of products and M means the number of threads, to compute the value of the output matrix, which leads to the sorted column indices of the output c and makes the most use of the register and shared memory resources.

PPoPP '18, February 24–28, 2018, Vienna, Austria

ACM ISBN 978-1-4503-4982-6/18/02.
https://doi.org/10.1145/3178487.3178529

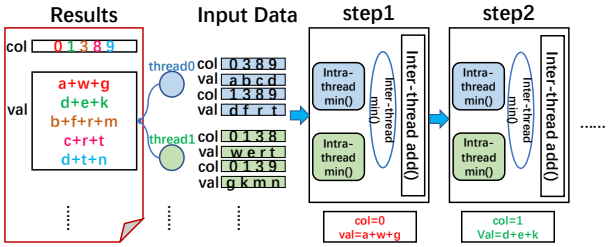


Figure 1. An example showing the proposed reg-spgemm and N-to-M design

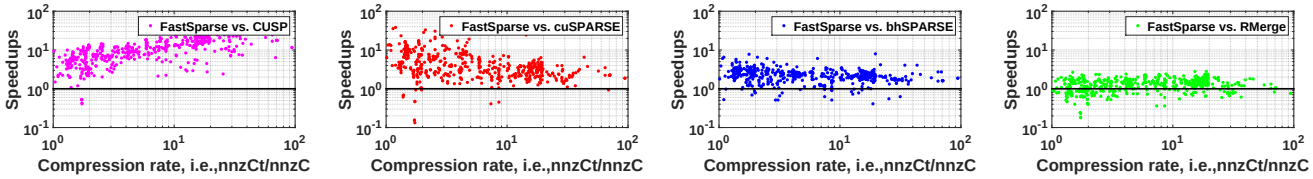
Based on this approach, we first propose the register-based SpGEMM algorithm (reg-spgemm) for the short rows of A so that the threads within a warp are sufficient to handle the corresponding intermediate products. Reg-spgemm relies on the warp shuffle instruction at the register level and the N-to-M product-thread binding scheme. Figure 1 shows an example of our method. It can be seen that the example needs to merge four rows of matrix B, which is implemented by two threads of one warp. In step 1, the $intra-thread\ min()$ is the operation that gets the minimum column index of two rows of matrix B within each thread. While the $inter-thread\ min()$ is the operation that gets the minimum column index of four rows of matrix B across threads, which is implemented by using the reduction of warp-level shuffle instructions. Then, by using the $inter-thread\ add()$ operation that is also implemented by leveraging the warp-level shuffle instructions, the first output element of vector c, i.e., $a + w + g$ is obtained. Also, with the same operations, in step 2, the second output element of vector c, i.e., $d + e + k$ is obtained. Actually, the steps are in one *for loop* until all four rows of matrix B are added up and the results of one row of the output matrix C are directly stored to global memory from registers.

When the row number of matrix B is large, the reg-spgemm will be insufficient, because the warp size of current Nvidia GPU is 32. Then the shared memory-based SpGEMM algorithm (smem-spgemm) is implemented to handle the long rows of A via multiple warps performing the reg-spgemm algorithm. At this time, the results of Figure 1 are not stored to global memory, but shared memory for the next merge operations. Through these operations we guarantee the effective utilization of both registers and shared memory.

The reg-spegemm and smem-spgemm algorithm is combined with our binning scheme for the rows of matrix A

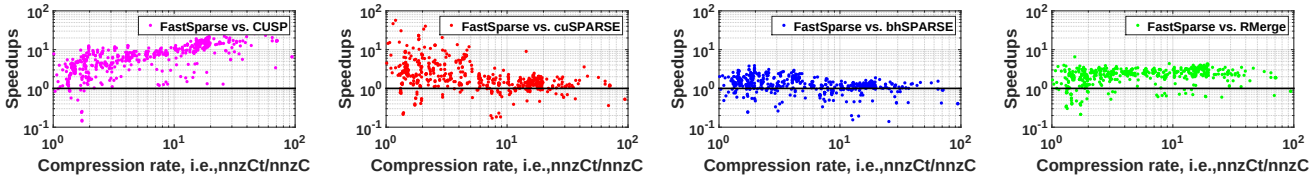
Table 1. Comparison between different SpGEMM libraries

	intermediate space allocation	global load balancing	nonzero-to-thread mapping	nonzero compression	memory use
CUSP [1]	upper bound	row+all sort	1-to-1	seg. sum	smem + gmem
cuSPARSE	precise	row	1-to-1	hash	smem + gmem
bhSPARSE [5]	progressive	row+bin/cta	1-to-1	seg. sum + hori. merge	smem
RMerge [3]	precise	row	1-to-1	vert. merge	reg. + gmem
FastSparse (ours)	precise	row+bin/warp	N-to-M	vert. merge	reg. + smem



(a) vs. CUSP (b) vs. cuSPARSE (c) vs. bhSPARSE (d) vs. RMerge

Figure 2. Performance comparison for double data on an Nvidia K40m (Kepler).



(a) vs.CUSP (b) vs. cuSPARSE (c) vs. bhSPARSE (d) vs. RMerge

Figure 3. Performance comparison for double data on an Nvidia Titan X (Pascal).

which are grouped into different categories. Then the rows in various bins adopt different parameters of reg-spgemm and smem-spgemm algorithms. We call the library of performing SpGEMM of ours as FastSparse.

3 Performance Evaluation and Conclusion

We use Nvidia K40m (Kepler) and Titan X (Pascal) GPUs for comparing the performance of our algorithm and several existing methods (CUSP [1], cuSPARSE, bhSPARSE [5] and RMerge [3]) that compute $C = A^2$ in double precision. The CUDA versions are 7.0 and 8.0 on K40m and Titan X, respectively. The selected benchmark suite includes 956 square sparse matrices with $100k \leq nnz \leq 200M$ from the SuiteSparse Matrix Collection [2].

The relative speedups are shown in Figures 2 and 3. It can be seen that the performance of our FastSparse is in general superior to the four existing libraries. Specifically, on K40m, our approach delivers a harmonic average speedup of 6.57x (up to 31.56x), 2.48x (up to 38.38x), 1.97x (up to 7.90x), and 1.12x (up to 2.82x) over CUSP, cuSPARSE, bhSPARSE and RMerge, respectively. On Titan X, the speedups are 3.75x (up to 25.76x), 1.16x (up to 56.48x), 1.07x (up to 3.82x), and 1.78x (up to 6.50x), respectively.

Acknowledgement

We would like to express our gratitude to all reviewer's constructive comments for helping us polish this paper. This work is supported by the National Key Research and Development Program of China (2016YFB0200300, 2016YFB0201305, 2016YFB0200504, 2016YFB0200803), National Natural Science Foundation of China, under grant no. (61521092, 91430218, 31327901, 61472395, 61432018) and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie project (752321).

References

[1] N. Bell, S. Dalton, and L. N. Olson. 2012. Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods. *SIAM Journal on Scientific Computing* 34, 4 (2012), C123–C152.

[2] T. A. Davis and Y. Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.* 38, 1 (2011), 1:1–1:25.

[3] F. Gremse, A. Höfter, L. O. Schwen, F. Kiessling, and U. Naumann. 2015. GPU-Accelerated Sparse Matrix-Matrix Multiplication by Iterative Row Merging. *SIAM Journal on Scientific Computing* 37, 1 (2015), C54–C71.

[4] K. Hou, W. Liu, H. Wang, and W. Feng. 2017. Fast Segmented Sort on GPUs. In *Proceedings of the International Conference on Supercomputing (ICS '17)*. 12:1–12:10.

[5] W. Liu and B. Vinter. 2015. A Framework for General Sparse Matrix-matrix Multiplication on GPUs and Heterogeneous Processors. *J. Parallel Distrib. Comput.* 85, C (Nov. 2015), 47–61.