Accelerating Matrix Processing with GPUs

(Invited Paper)

Nicholas Malaya, Shuai Che, Joseph L. Greathouse, René van Oostrum, and Michael J. Schulte
AMD Research
Advanced Micro Devices, Inc.
7171 Southwest Pkwy, Austin, TX 78735
Email: Michael.Schulte@amd.com

I. Summary of Work

Matrix operations are common and expensive computations in a variety of applications. They occur frequently in high-performance computing [1], [2], graphics [3], graph processing [4], [5], and machine learning [6], [7] applications.

This paper discusses how to map a variety of important matrix computations, including sparse matrix-vector multiplication (SpMV), sparse triangle solve (SpTS), graph processing, and dense matrix-matrix multiplication, to GPUs. Since many emerging systems will use heterogeneous architectures (e.g. CPUs and GPUs) to attain the desired performance targets under strict power constraints [8], this paper discusses implications and future research for matrix processing with heterogeneous designs.

Conclusions common to the matrix operations discussed in this paper are: (1) Future algorithms should be written to ensure that the essential computations fit into local memory, which may require direct programmer management. (2) Algorithms are needed that expose high levels of parallelism. (3) While the scale of computation is often sufficient to support algorithms with superior asymptotic order, additional considerations, such as memory capacity and bandwidth, must also be carefully managed. (4) Libraries should be used to provide portable performance.

II. Sparse Matrix-Vector Multiplication (SpMV)

SpMV is a linear algebra primitive that multiplies a matrix stored in a sparse format by a dense vector, resulting in another dense vector: $y = Ax$. SpMV is used in a wide range of applications, from iterative solvers to machine learning and graph analytics [9], [10]. The algorithm used to calculate SpMV is heavily dependent on the data structure used to hold the sparse matrix. Perhaps the most popular format is compressed sparse row (CSR), since it compresses most matrices well, and CPU-based algorithms that use CSR are simple to write and perform well.

Other algorithms (e.g. sparse matrix-dense matrix multiplication) rely on CSR for their own high-performance implementations [11], and a great deal of existing code written for CPUs uses CSR. Recent works have found ways to achieve high GPU-based SpMV performance while leaving the CSR data structure untouched [12], [13]. Since these algorithms do not change the CSR structures, other algorithms can use the matrices without requiring further transformations.

For example, CSR-Adaptive [12], [13] records the number of non-zero values in each row when the matrix is first created. It then groups together adjacent rows into blocks such that each has roughly the same number of non-zero values. Unlike previous CSR-based SpMV algorithms, which use one thread per row or which send a fixed number of rows to each SIMD, CSR-Adaptive changes the number of rows each SIMD unit operates on depending on the number of non-zero values in each row.

A block of matrix rows may contain many short rows or a small number of very long rows, and the algorithm changes the way it calculates the result for each row depending on how many elements it is operating on. CSR-Adaptive loads the contiguous rows into on-chip scratchpad memory with a technique that does not cause memory divergence. After this, one SIMD thread can be used to calculate each row, since on-chip scratchpads do not suffer from the same memory divergence problems as DRAM systems. Alternately, for the case of an extremely long row, multiple threads can be used to calculate parts of that row, generating more parallelism.

Modern GPU SpMV algorithms that use CSR are now some of the highest-performing implementations since they solve the problems of memory divergence and lack of parallelism. Implementations of CSR-Adaptive, for instance, are now at the core of open source sparse BLAS libraries such as ViennaCL [14] and clSPARSE [15]. While current GPU SpMV implementations can reach up to 95% efficiency for many input matrices, some inputs see much worse performance. This primarily occurs when the vector inputs do not cache well. For instance, very wide rows require a large number of vector memory accesses, which can displace useful data from caches and cause memory bank and channel conflicts.

Solving these issues is the primary remaining research problem in this domain. Cache bypassing when vector inputs do not cache well could raise cache hit rates. In addition, scheduling work for long rows to other processors (e.g. CPUs), so that they do not cause GPU cache or memory conflicts, may lead to further improvements.

III. Sparse Triangle Solve (SpTS) on GPUs

Triangular solves appear (with slight variations) in a broad range of numerical linear algebra algorithms, such as direct methods, preconditioned iterative methods, and least squares problems. In HPC, most of the HPCG benchmark's execution time is spent in algorithms like SpTS [16], [17].

Triangle solve corresponds to solving an equation $A\,\mathbf{x} = \mathbf{b}$ for the values of the vector $\mathbf{x}$. Consider a dense lower triangular 3×3 matrix solve:

$$\begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

The first two rows are therefore, $x_1 = b_1/a_{11}$ and, $x_2 = (b_2 - a_{21}x_1)/a_{22}$.

The challenge of accelerating this problem for sparse matricies on GPUs are: (1) the extra work needed to *find data dependencies* between matrix rows and (2) a general *lack of parallelism* due to these dependencies. Unless the only non-zero values exist on the diagonal, some serialization must occur. Determining whether two rows can run in parallel (i.e. if the second row has no non-zero values in the column with the first row's diagonal) is difficult when the matrix is stored in the CSR format.

Existing solutions generally try to find level sets, which require an expensive pre-processing analysis that is typically many times slower than the solution step itself [18]. Other work uses graph coloring [19], [17], which reorders rows and breaks some transitive dependencies. While this may accelerate SpTS-style algorithms, it can result in more

1063-6889/17 $31.00 © 2017 IEEE
DOI 10.1109/ARITH.2017.14

iterations, as it is (in essence) solving a different set of equations than the original problem.

Liu *et al.* recently showed an SpTS algorithm that does not require any pre-processed data dependency analysis [20]. Their algorithm requires the matrix to be in the compressed sparse column (CSC) format, which is a transposed CSR. With the CSC format, threads can know which subsequent rows rely on their result, meaning they can "push" results forward to dependent threads. Threads that have been fed work are actively computing, while the others spin-loop waiting for data. The principle limitation of this algorithm is that it requires a matrix transpose (from CSR to CSC and potentially back to CSR).

Operations such as SpTS will be increasingly important for future HPC systems. With billions of concurrent threads, these system have the potential to be seriously bottlenecked by a lack of parallelism, and small serial regions can become an Amdahl's Law limit. In such systems, heterogeneous processors with cores that can quickly execute serial regions may be needed to maximize SpTS performance.

IV. Graph Processing

Prior research on GPU graph processing has taken a problem-specific approach by parallelizing and optimizing individual algorithms on the GPU [21], [22]. These advanced techniques usually are not intuitive for regular programmers because significant code restructuring is needed to fully use the GPU. A solution to this is the GPU library, *BelRed* [23]. This library contains a set of key building blocks common to many graph algorithms. Table I shows some sample BelRed functions. BelRed implements graph algorithms using linear algebra operations [24], [25]. However, optimizing these operations for the GPU and their proper use in implementing diverse graph processing applications has not been sufficiently studied by prior work.

TABLE I: Sample BelRed functions [23]

Functions	Description
$\vec{u} = SpMV(M, \vec{v})$	sparse-matrix vector multiplication
$\vec{u} = SpMinDotPlus(M, \vec{v})$	the $min.+$ operation
$\vec{u} = SegReduc\_Op(M)$	segmented reduction. $Op: +, \&, min...$
$U = SpGeMM(M, N)$	sparse-matrix and sparse-matrix multiply
$U = vOuterSum(\vec{v}, \vec{w})$	the outer-sum operation
$\vec{u} = vElemWise\_Op(\vec{v}, \vec{w})$	vector elem. wise. $Op: +, \&, min, .$
$U = SpElemWise\_Op(\vec{v}, \vec{w})$	sparse matrix elem. wise. $Op: +, \&, min, .$

CSR-Adaptive (see Section II) and Liu & Vinter's algorithm [11] are used to optimized sparse-matrix and vector, and sparse-matrix algorithms, respectively. Figure 1 shows an example using min.+ to

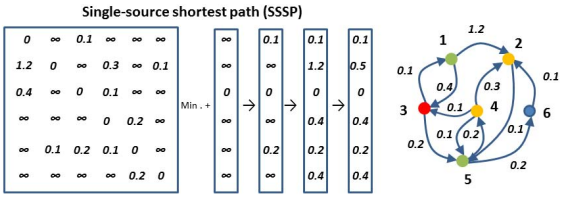

Fig. 1: An example of conducting $min.+$ repeatedly to implement single-source shortest path (SSSP) [25]. For an element (j, i) with value w, vertex i connects to vertex j with an edge weight of w.

solve single-source shortest path (SSSP). Various basic graph algorithms are implemented with BelRed including PageRank, Coloring, and Floyd-Warshall [23]. BelRed implementations are also provided for data analytic problems such as Ktruss and Jaccard Coefficients using the algorithms in [26].

BelRed was initially implemented in OpenCL; support is being added for HSA [27] and ROCm [28]. Future work will optimize BelRed for Exascale node architectures [29]. Parallelizing these operations and partitioning data across nodes without significant load imbalance is a challenge. Furthermore, efficient runtime techniques are needed to redistribute workloads dynamically at runtime. In addition, due to little data reuse and low arithmetic intensity, network communication is important. Finally, some graph algorithms launch a series of BelRed functions iteratively. They may exhibit interesting dependencies among each other where sequential execution is not required. Thus, it may be feasible to exploit more parallelism with asynchronous execution to improve the resource utilization.

V. Matrix-Matrix Multiply on GPUs

The product of two matrices is an important operation in machine learning, particularly in Deep Learning applications [30] and scientific computing [31], [32]. For dense square matrices with sizes of a few thousand, the Strassen-Winograd algorithm is the fastest matrix multiplication implementation [33], [34]. This is due to its superior asymptotic order: $\mathcal{O}(n^{2.807})$ versus $\mathcal{O}(n^3)$ for traditional convolutions.

The Strassen algorithm is a recursive approach that reduces the number of multiplication operations in a matrix multiply while increasing the number of addition operations. Consider the matrix multiplication of the 2×2 matrices A and B, i.e. $C = AB$,

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} * b_{12} + a_{12} * b_{22} \\ a_{21} * b_{11} + a_{21} * b_{21} & a_{21} * b_{12} + a_{22} * b_{22} \end{pmatrix}.$$

which requires 8 multiplies and 4 additions.

Consider instead the Strassen algorithm, which forms surrogate values,

$$\begin{aligned} P_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ P_2 &= (a_{21} + a_{22})(b_{11}) \\ P_3 &= (a_{11})(b_{12} - b_{22}) \\ P_4 &= (a_{22})(b_{21} - b_{11}) \\ P_5 &= (a_{11} + a_{12})(b_{22}) \\ P_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ P_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}$$

and then computes the matrix elements as $c_{11} = P_1 + P_4 - P_5 + P_7$, $c_{12} = P_3 + P_5$, $c_{21} = P_2 + P_4$, and $c_{22} = P_1 + P_3 - P_2 + P_6$. Counting these equations indicates that the Strassen algorithm has reduced the number of multiplies from eight to seven, at the cost of 18 additions. Notice also this algorithm can be applied to larger $2k \times 2k$ matrices. This only requires treating each element (e.g. a_{12}) as a separate $k \times k$ sub-matrix. For sufficiently large matrices, replacing $\mathcal{O}(n^3)$ matrix multiplications by $\mathcal{O}(n^2)$ matrix additions results in a speedup. Submatrices can be computed recursively as long as they are large enough to benefit from the surrogate values approach.

Given the importance of matrix-matrix multiplications for a variety of workloads, this will remain a critical algorithm for future supercomputers. A critical bottleneck in parallelizing Strassen's algorithm is the communication between the processors [35]. Effective and well-supported distributed memory dense linear algebra libraries are still largely unavailable, although some frameworks are emerging [36]. Some recent work has sought to combine Strassen-Winograd and sparse matrix computations [37] to accelerate convolutions, for instance, by making the neural network weights sparse. FFTs, which provide an alternative method to reduce computation while simultaneously requiring more memory capacity and bandwidth, are also commonly used [38].

REFERENCES

[1] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present and Future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003.

[2] J. Dongarra, M. A. Heroux, and P. Luszczek, "HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems," University of Tennessee Electrical Engineering and Computer Science Department, Knoxville, Tennessee, Tech. Rep. UT-EECS-15-736, November 2015.

[3] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL® Programming Guide*, 8th ed. Pearson Education, 2013.

[4] K. Bryan and T. Leise, "The $25,000,000,000 Eigenvector: The Linear Algebra behind Google," *SIAM Review*, vol. 48, no. 3, pp. 569–581, 2006.

[5] S. Che, B. M. Beckmann, and S. K. Reinhardt, "BelRed: Constructing GPGPU Graph Applications with Software Building Blocks," in *Proc. of the IEEE High Performance Extreme Computing Conf. (HPEC)*, 2014.

[6] O. Abdel-Hamid, A. r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional Neural Networks for Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533–1545, Oct 2014.

[7] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *CoRR*, vol. abs/1604.07316, 2016.

[8] T. Vijayaraghavan, Y. Eckert, G. H. Loh, M. Schulte, M. Ignatowski, I. Paul, B. Beckmann, S. Reinhardt, W. Brantley, J. Greathouse, O. Kayiran, M. Poremba, W. Huang, A. Karunanithi, G. Sadowski, V. Sridharan, S. Raasch, and M. Meswani, "Design and Analysis of an APU for Exascale Computing," in *Proceedings of The 23rd IEEE Symposium on High Performance Computer Architecture (HPCA)*, February 2017.

[9] E.-J. Im and K. Yelick, "Optimization of Sparse Matrix Kernels for Data Mining," in *Proc. of the Workshop on Text Mining*, 2001.

[10] J. R. Gilbert, S. Reinhardt, and V. B. Shah, "High-performance Graph Algorithms from Parallel Sparse Matrices," in *Proc. of the Int'l Workshop on Applied Parallel Computing*, 2006.

[11] W. Liu and B. Vinter, "An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data," in *Proc. of the Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2014.

[12] J. L. Greathouse and M. Daga, "Efficient Sparse Matrix-Vector Multiplication on GPUs using the CSR Storage Format ," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2014.

[13] M. Daga and J. L. Greathouse, "Structural Agnostic SpMV: Adapting CSR-Adaptive for Irregular Matrices," in *Proc. of the Int'l Conf. on High Performance Computing (HiPC)*, 2015.

[14] K. Rupp, F. Rudolf, and J. Weinbub, "ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs," in *Int'l Workshop on GPUs and Scientific Applications (GPUScA)*, 2010.

[15] J. L. Greathouse, K. Knox, J. Poła, K. Varaganti, and M. Daga, "clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library," in *Proc. fo the Int'l Workshop on OpenCL (IWOCL)*, 2016.

[16] K. Kumahata, K. Minami, and N. Maruyama, "High-performance Conjugate Gradient Performance Improvement on the K Computer," *Int. J. High Perform. Comput. Appl.*, vol. 30, no. 1, pp. 55–70, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1177/1094342015607950

[17] E. Phillips and M. Fatica, "A CUDA Implementation of the High Performance Conjugate Gradient Benchmark," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2014, pp. 68–84.

[18] M. Naumov, "Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU," *Nvidia white paper*, 2011.

[19] M. Naumov and P. Castonguay and J. Cohen, "Parallel Graph Coloring with Applications to the Incomplete-LU Factorization on the GPU," *Nvidia White Paper*, 2015.

[20] W. Liu, A. Li, J. Hogg, I. S. Duff, and B. Vinter, *A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves*. Cham: Springer International Publishing, 2016, pp. 617–630.

[21] D. Merrill, M. Garland, and A. Grimshaw, "Scalable GPU Graph Traversal," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Feb 2012.

[22] S. Che, B. Beckmann, S. Reinhardt, and K. Skadron, "Pannotia: Understanding Irregular GPGPU Graph Algorithms," in *Proceedings of the IEEE International Symposium on Workload Characterization*, Sept 2013.

[23] S. Che, B. M. Beckmann, and S. K. Reinhardt, "Programming GPGPU graph applications with linear algebra building blocks," *International Journal of Parallel Programming*, pp. 1–23, 2016.

[24] A. Buluc and J. R. Gilbert, "The Combinatorial BLAS: Design, Implementation, and Applications," *International Journal of High Performance Computing Applications*, vol. 25, no. 4, Nov. 2011.

[25] J. Kepner and J. Gilbert, *Graph Algorithms in the Language of Linear Algebra*. Society for Industrial and Applied Mathematics, Jan. 2011.

[26] D. Hutchison, J. Kepner, V. Gadepally, and B. Howe, "From NoSQL Accumulo to NewSQL Graphulo: Design and Utility of Graph Algorithms inside a BigTable Database," in *Proc. of the IEEE High Performance Extreme Computing Conf. (HPEC)*, Sept 2016.

[27] Heterogeneous System Architecture (HSA), "Web resource," http://hsafoundation.com/.

[28] ROCm, "Web resource," https://radeonopencompute.github.io/install.html.

[29] M. J. Schulte, M. Ignatowski, G. H. Loh, B. M. Beckmann, W. C. Brantley, S. Gurumurthi, N. Jayasena, I. Paul, S. K. Reinhardt, and G. Rodgers, *IEEE Micro*, vol. 35, no. 4, pp. 26–36, 2015.

[30] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[31] D. McDougall, N. Malaya, and R. D. Moser, *The Parallel C++ Statistical Library for Bayesian Inference: QUESO*. Cham: Springer International Publishing, 2016, pp. 1–38.

[32] M. Lee, N. Malaya, and R. D. Moser, "Petascale direct numerical simulation of turbulent channel flow on up to 786k cores," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 61:1–61:11. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503298

[33] A. R. Benson and G. Ballard, "A Framework for Practical Parallel Fast Matrix Multiplication," *CoRR*, vol. abs/1409.2908, 2014. [Online]. Available: http://arxiv.org/abs/1409.2908

[34] P.-W. Lai, H. Arafat, V. Elango, and P. Sadayappan, "Accelerating Strassen-Winograd's matrix multiplication algorithm on GPUs," in *High Performance Computing (HiPC), 2013 20th International Conference on*. IEEE, 2013, pp. 139–148.

[35] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-Optimal Parallel Algorithm for Strassen's Matrix Multiplication," *CoRR*, vol. abs/1202.3173, 2012. [Online]. Available: http://arxiv.org/abs/1202.3173

[36] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A New Framework for Distributed Memory Dense Matrix Computations," *ACM Trans. Math. Softw.*, vol. 39, no. 2, pp. 13:1–13:24, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2427023.2427030

[37] S. Li, J. Park, and P. T. P. Tang, "Enabling Sparse Winograd Convolution by Native Pruning," *arXiv preprint arXiv:1702.08597*, 2017.

[38] M. Mathieu, M. Henaff, and Y. LeCun, "Fast Training of Convolutional Networks Through FFTs," *arXiv preprint arXiv:1312.5851*, 2013.