Exploiting Gustavson's Algorithm to Accelerate Sparse Matrix Multiplication Extended Abstract

Guowei Zhang[†] Nithya Attaluri[†] Joel Emer[†*] Daniel Sanchez[†]
[†]MIT [*]NVIDIA
{zhanggw, nsattaluri, emer, sanchez}@csail.mit.edu

1. Motivation

Scientific and machine learning applications are increasingly computing on sparse data, i.e., data where a large fraction of values are zeros. In this work, we focus on accelerating sparse matrix-sparse matrix multiplication (SPMSPM), a key kernel that lies at the heart of many sparse algorithms, like sparse deep neural networks [5, 10], sparse linear and tensor algebra [8, 13], graph analytics [3, 7], and scientific simulations [1].

SPMSPM has two key characteristics that make it challenging to accelerate. First, SPMSPM is *bottlenecked by memory traffic and data movement*: it requires far fewer arithmetic operations per input element than dense matrix multiplication, and its inputs and outputs typically use a *compressed* representation that omits zeros but is more complicated to traverse, requiring irregular and indirect accesses. Thus, to be effective, accelerators *must minimize data movement*, rather than compute operations. Second, SPMSPM has a *rich algorithmic diversity*: it admits a wide range of *dataflows* (i.e., computation schedules) with different tradeoffs, and some dataflows have asymptotically worse performance on particular inputs. Thus, accelerators must achieve efficiency though specialization while avoiding the inefficiencies of using a suboptimal SPMSPM dataflow.

2. Limitations of the State of the Art

Prior work has proposed SPMSPM accelerators that greatly improve performance over CPUs and GPUs. And yet, these accelerators have focused on one of two SPMSPM dataflows, *inner-product* [6, 12] or *outer-product* [9, 15], which suffer from poor input or output reuse, leading to high traffic that limits speedups. *Gustavson's algorithm* [4] does not suffer from these problems but features irregular memory access patterns that are a poor match to prior accelerators.

In SPMSPM, inputs and output matrices are encoded in a compressed sparse representation, where each row or column is encoded as a *fiber*, a list of coordinates and nonzero values sorted by coordinate. Compressed sparse data structures avoid encoding frequent zero values, but can only efficiently be traversed in a particular order, and require indirect accesses to locate each fiber. SPMSPM requires accessing variable-sized fibers and intersecting or combining them. These operations are inefficient on CPUs and GPUs.

Accelerators like UCNN [6] and SIGMA [12] implement *inner-product* SPMSPM. Inner-product produces the output matrix one element at a time, by intersecting rows and columns of the input matrices. Inner-product maximizes output reuse but sacrifices reuse of the input matrices, and is inefficient with very sparse matrices, as it is dominated by the cost of intersections. Intersections are inefficient because all of the elements of the rows and columns must be traversed, but few row and column elements have a matching coordinate and contribute to the output. Thus, most of the intersection is ineffectual.

By contrast, accelerators like OuterSPACE [9] and SpArch [15] implement *outer-product* SPMSPM. Outer-product computes the output one partial matrix at a time, by traversing each row and column of the input matrices once and producing a partial output matrix that includes all of their contributions. Outer-product achieves good reuse of the input matrices and avoids inner-product's ineffectual intersections, at the cost of producing many large partial output matrices that must be combined (merged) to produce the final output. Outer-product accelerators are thus burdened with significant partial output matrix traffic and the complexity associated with combining those partial output matrices.

Accelerators include multiple optimizations over these basic dataflows to mitigate their inefficiencies, such as tiling and preprocessing the input matrices. Nonetheless, they are hampered by the fundamental drawbacks of inner- or outer-product, and incur order-of-magnitude traffic inefficiencies. For example, Fig. 3 in the paper shows that these accelerators incur 14–1045× more traffic than needed on two representative matrices. Since SPMSPM is memory-bound on accelerators, these traffic overheads translate to performance degradations.

3. Key Insights

The key insight in this paper is to leverage *Gustavson's algorithm* [4], an alternative SPMSPM dataflow passed over by previous hardware designs, to build a far more efficient and versatile accelerator. Gustavson computes the output matrix one row of a time, by traversing rows of A and linearly combining (i.e., scaling and reducing) the rows of B for which the row of A has nonzero values. Gustavson is more efficient because it avoids the extremes of inner- and outer-product dataflows. While Gustavson does not get as much reuse of a single value as inner- or outer-product dataflows, it gets reuse of modestly sized rows. Unlike outer-product, Gustavson requires combining partial output *rows* rather than partial output matrices, a simpler operation on much smaller intermediates that more easily fit on-chip; and unlike inner-product, Gustavson avoids

ineffectual intersections and poor input reuse.

Leveraging Gustavson in SPMSPM acceleration requires new architectural support to adapt to its characteristics, namely its irregular reuse. Gustavson also has qualitatively different benefits in hardware than in software: while nearly every high-performance CPU and GPU SPMSPM implementation is a variant of Gustavson's, software versions leverage Gustavson mainly to reduce the cost of merging fibers, which is their dominant overhead. By contrast, Gustavson's key benefit in accelerators is reducing memory traffic.

Specifically, we present GAMMA, the Gustavson-Algorithm Matrix-Multiplication Accelerator. GAMMA combines three key features to exploit Gustavson's algorithm:

1. GAMMA uses simple processing elements (PEs) that linearly combine sparse input rows to produce each output row. PEs implement high-radix mergers that combine many input rows (e.g., 64 in our design) in a single pass, reducing work and memory accesses. Instead of expensive high-throughput mergers as in prior work [15], GAMMA uses simple one-output-per-cycle mergers that take 27× less area. GAMMA then relies on Gustavson's row-level parallelism to achieve high throughput efficiently, using tens of PEs to perform many combinations in parallel. Thus, GAMMA concurrently processes *thousands* of compressed sparse *fibers*, variable-sized rows from inputs or partial outputs.
2. GAMMA uses a novel storage structure, FIBERCACHE, to efficiently buffer the thousands of fibers required by the PEs. FIBERCACHE is organized as a cache to capture Gustavson's irregular reuse patterns. However, FIBERCACHE is managed explicitly [11], like a large collection of buffers, to proactively fetch missing fibers ahead of time and avoid PE stalls. This saves megabytes of dedicated on-chip buffers.
3. GAMMA dynamically schedules work among PEs to ensure high utilization and minimize memory traffic despite the irregular nature of Gustavson's algorithm.

GAMMA outperforms prior accelerators on a wide range of inputs. However, GAMMA still incurs more traffic than needed on some inputs. To address this issue, we propose a preprocessing technique that combines row reordering and tiling of some rows of one matrix input. Preprocessing further improves GAMMA's performance and avoids pathologies across the full range of inputs.

4. Main Artifacts

- We present GAMMA, a practical implementation of an efficient accelerator that implements Gustavson's algorithm for SPMSPM. This includes the design of PEs that leverage a simple high-radix merger to efficiently perform the key computation needed by Gustavson's algorithm and the design of a novel cache-like structure that performs efficient data orchestration of sparse fibers.
- We present data preprocessing techniques that reorder the rows of the matrices and also selectively tile rows of the matrices to improve the efficiency of GAMMA.
- We synthesize GAMMA to project its hardware costs and compare them to prior accelerators.
- We evaluate GAMMA's performance and memory efficiency with detailed simulation on a wide range of sparse matrices.

5. Key Results and Contributions

- We show that Gustavson's dataflow is often more efficient than the other two SPMSPM dataflows. But it is challenging to implement in accelerators due to its use of less regular access patterns than previously implemented dataflows.
- We build GAMMA, a novel SPMSPM accelerator to implement a dataflow inspired by Gustavson's algorithm. The accelerator combines specialized PEs, that efficiently compute linear combinations of rows, a novel cache-like structure to capture Gustavson's irregular reuse, and dynamic task scheduling to achieve high utilization despite irregularity.
- We propose preprocessing techniques that boost GAMMA's effectiveness by generating a reference pattern that reduces memory traffic.
- We evaluate GAMMA under a broad range of matrices, showing large performance gains and memory traffic reductions over prior systems. Compared to the state-of-the-art accelerator, SpArch, GAMMA improves performance by 2.1× with a smaller area budget. GAMMA reduces total DRAM traffic by 2.2× on average and by up to 13×, reduces non-compulsory DRAM traffic by 13× on average, and achieves nearly full DRAM bandwidth utilization. GAMMA delivers larger benefits over OuterSPACE and a state-of-the-art software SPMSPM. Moreover, GAMMA is effective on a much broader range of sparse matrices.

6. Why ASPLOS

Accelerators are a common topic for ASPLOS, and this paper is particularly well suited because it melds insights and techniques from software and hardware in different ways. First, we focus on exploiting a dataflow that has been underexplored in hardware despite being widely used in software implementations. Our work shows that, with the right architectural support, Gustavson's algorithmic advantages confer significant benefits in hardware, although these benefits are different from software. Our work aligns software and hardware implementations around a common algorithm. Second, this work accelerates SPMSPM, which is widely used in machine learning, graph analytics, and linear algebra. By providing hardware support for a more efficient and versatile dataflow, GAMMA can simplify the development of programming stacks in these domains, such as compilers and optimizations for sparse data [2, 8, 14]. Finally, this work illustrates the effectiveness of hardware-software co-design by exploring specific approaches to preprocessing of matrices that both reorder rows and selectively tile some rows to improve efficiency.

References

[1] A Canning, G Galli, F Mauri, A De Vita, and R Car. O (n) tight-binding molecular dynamics on massively parallel computers: an orbital decomposition approach. *Computer Physics Communications*, 94(2-3), 1996.

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX symposium on Operating Systems Design and Implementation (OSDI-13)*, 2018.

[3] John R Gilbert, Steve Reinhardt, and Viral B Shah. High-performance graph algorithms from parallel sparse matrices. In *International Workshop on Applied Parallel Computing*, 2006.

[4] Fred G Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software (TOMS)*, 4(3), 1978.

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

[6] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. In *Proceedings of the 45th annual International Symposium on Computer Architecture (ISCA-45)*, 2018.

[7] Jeremy Kepner, David Bader, Aydın Buluç, John Gilbert, Timothy Mattson, and Henning Meyerhenke. Graphs, matrices, and the GraphBLAS: Seven good reasons. *Procedia Computer Science*, 51, 2015.

[8] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. The tensor algebra compiler. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2018.

[9] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. Outerspace: An outer product based sparse matrix multiplication accelerator. In *Proceedings of the 24th IEEE international symposium on High Performance Computer Architecture (HPCA-24)*, 2018.

[10] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th annual International Symposium on Computer Architecture (ISCA-44)*, 2017.

[11] Michael Pellauer, Yakun Sophia Shao, Jason Clemons, Neal Crago, Kartik Hegde, Rangharajan Venkatesan, Stephen W Keckler, Christopher W Fletcher, and Joel Emer. Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration. In *Proceedings of the 24th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XXIV)*, 2019.

[12] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *Proceedings of the 26th IEEE international symposium on High Performance Computer Architecture (HPCA-26)*, 2020.

[13] Ichitaro Yamazaki and Xiaoye S Li. On techniques to improve robustness and scalability of a parallel hybrid linear solver. In *International Conference on High Performance Computing for Computational Science*, 2010.

[14] Yunming Zhang, Mengjiao Yang, Riyadh Baghdadi, Shoaib Kamil, Julian Shun, and Saman Amarasinghe. Graphit: A high-performance graph dsl. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2018.

[15] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. Sparch: Efficient architecture for sparse matrix multiplication. In *Proceedings of the 26th IEEE international symposium on High Performance Computer Architecture (HPCA-26)*, 2020.