SIGMICRO: G: Efficient Sparse Matrix Kernels based on Adaptive Workload-Balancing and Parallel-Reduction

Guyue Huang (Advisors: Guohao Dai, Yu Wang (Tsinghua University), Yufei Ding and Yuan Xie (UCSB))
University of California, Santa Barbara
guyue@ucsb.edu

ABSTRACT

Sparse matrix-vector and matrix-matrix multiplication (SpMV and SpMM) are fundamental in both conventional (graph analytics, scientific computing) and emerging (sparse DNN, GNN) domains. Workload-balancing and parallel-reduction are widely-used design principles for efficient SpMV. However, prior work fails to resolve how to *implement* and *adaptively use* the two principles for SpMV/MM. To overcome this obstacle, we first complete the implementation space with optimizations by filling three missing pieces in prior work, including: (1) We show that workload-balancing and parallel-reduction can be combined through a segment-reduction algorithm implemented with SIMD-shuffle primitives. (2) We show that parallel-reduction can be implemented in SpMM through loading the dense-matrix rows with vector memory operations. (3) We show that vectorized loading of sparse rows, being a part of the benefit of parallel-reduction, can co-exist with sequential-reduction in SpMM through temporally caching sparse-matrix elements in the shared memory. In terms of adaptive use, we analyze how the benefit of two principles change with two characteristics from the input data space: the diverse sparsity pattern and dense-matrix width. We find the benefit of the two principles fades along with the increased total workload, i.e. the increased dense-matrix width. We also identify, for SpMV and SpMM, different sparse-matrix features that impact workload-balancing effectiveness. Our design consistently exceeds cuSPARSE by 1.07-1.57× on different GPUs and dense matrix width, and the kernel selection rules involve 5-12% performance loss compared with optimal choices. Our kernel is being integrated into popular graph learning frameworks [1, 2] to accelerate GNN training.

1 PROBLEM AND MOTIVATION

Efficient basic sparse-matrix primitives can benefit a variety of applications. The sparse matrix multiplication $Y_{M\times N} = A_{M\times K}X_{K\times N}$ where A is sparse and X, Y are dense, is referred to as Sparse Matrix-Vector product (SpMV, when $N = 1$) or Sparse Matrix-Matrix product (SpMM, when $N > 1$). SpMV and SpMM are fundamental components to a wide range of problem domains. SpMV is used in graph analytics and scientific computing [3, 4]. SpMM is used in iterative algorithms for sparse matrix factorization [5]. Recent advances in sparse NN, promising higher computational efficiency than dense models, rely on fast SpMV/MM kernels to demonstrate speedup in practice [6]. SpMM is also a core operation in graph neural networks (GNNs) [7, 8]. Accelerating SpMV/MM on GPUs, the dominating HPC hardware in presence, can potentially boost the performance of many aforementioned applications.

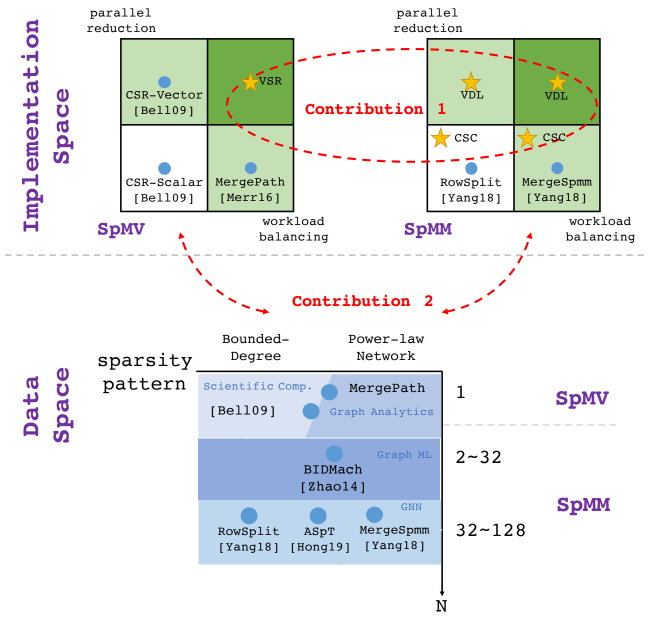

Figure 1: Target problem and contributions of this work. We consider two design principles, workload-balancing and parallel-reduction. First, we complete the implementation space with highly-optimized kernels. Second, we map a variety of SpMV/MM problems to implementations adaptively.

Workload-balancing and **parallel-reduction** are two design principles extensively studied for SpMV. Workload-balancing prevents the kernel from being bottlenecked by the most work-intensive thread, either through row-binning [6, 9], or through fine-grained segmentation of arithmetic and memory operations [10](Figure. 2(b)). Parallel-reduction increases the resource occupancy and bandwidth utilization compared to a sequential-reduction, through performing inner-product with vectorized element-wise multiplication and SIMD merge-tree [11](Figure. 2(c)). Prior art, however, leaves some missing pieces in the implementation and adaptive use of the two principles.

In terms of implementation, we lack the guidance to combine the two principles and extend them from SpMV to SpMM. Firstly, the combination of workload-balancing and parallel-reduction is not covered by prior work. The combination leads to a segment-reduction operation, whose existing solution is through tiled sequential scan, with limited resource occupancy. Secondly, how to apply parallel-reduction in SpMM is missed in previous work. Previous work on SpMM [12, 13] all apply sequential-reduction,

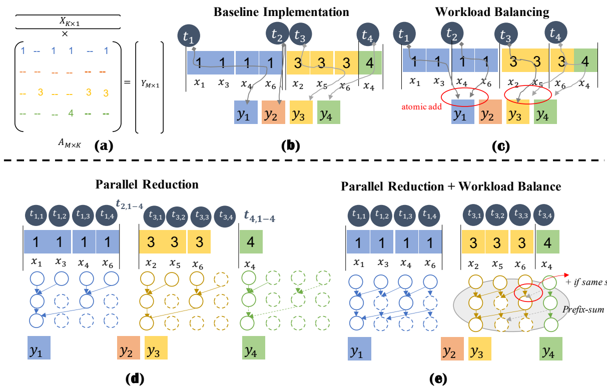

Figure 2: Illustration of workload-balancing and parallel-reduction. We combine the two principles with a vectorized segment-reduction algorithm with SIMD-shuffle primitives as shown in (e).

and target $N \geq 32$ scenarios. Given the effectiveness of parallel-reduction in SpMV [11], we naturally ask if parallel-reduction can bring similar benefit to SpMM with small N. However, prior work does not cover how to perform parallel-reduction in SpMM. We propose efficient implementations for these missing pieces, as marked in Figure. 1.

In terms of adaptive use of workload-balancing and parallel-reduction based on input characteristics, prior work fails to solve two problems: firstly, how to selectively apply the two principles based on the **sparsity pattern**, and secondly, how the answer to the first question changes with the **dense-matrix width** N. Many prior researches apply work-balancing not always but selectively. Still, comprehensive analysis of how sparsity features and N affect this choice is missing in prior work. Parallel-reduction is very effective for SpMV [11], but not applied to SpMM designs [6, 12–14]. The transition point from parallel-reduction to sequential-reduction is also not discussed in previous work.

Rethinking the two design principles for SpMV/MM, we make the following contributions:

- **Complete space with optimizations**. We fill the gaps of practicing workload-balancing and parallel-reduction in SpMV/MM with three novel optimizations: vectorized segment reduction, vector-type dense-row loading, and coalesced sparse-row caching. (Section. 2.1)
- **Heuristics from data to implementation**. We draw insights from evaluating the two principles on a large benchmark of sparse matrices, and to a range of N from 1 up to 128. We analyze why the benefit of two principles fades as N increases, and provide low-cost rules to selectively apply them. (Section. 2.2)
- **Comprehensive experimental results**. We extensively evaluate our method on three GPUs. Our approach outperforms cuSPARSE [15] by **1.07-1.52**×. The kernel selection strategy demonstrates an average 5-12% performance loss, compared to 68% in minimum if always picking one design. (Section. 3)

2 APPROACH AND UNIQUENESS

2.1 Implementing Two Principles

In this part, we present three optimizations for best practice of workload-balancing and parallel-reduction in SpMV/MM.

2.1.1 *Vectorized Segment Reduction (VSR).*

Motivation: The combination of workload-balancing parallel-reduction can effectively handle *short rows* in the sparse matrix. As shown in Figure. 2(d), in CSR-Vector SpMV, the de-facto practice of parallel-reduction, when the number of non-zeros in a row is smaller than the number of threads in a GPU warp (similar to a SIMD thread bundle), parallel-reduction performs many wasted operations. On the contrary, warps processing long rows bear heavy workload and can bottleneck the execution. To solve this, we apply the workload-balancing principle by assigning to each warp a fixed number of non-zeros instead of a certain row, as in Figure. 2(e).

Approach: After applying workload-balancing, the elements assigned to each warp can cross the boundary of rows. Hence, instead of the pure merge-tree reduction in CSR-Vector, we need to perform a segment reduction operation referring to the row-indices of each element. We implement our vectorized segment reduction (VSR) algorithm by simulating a prefix-sum network, but the reduction operation is to *add if the indices of two elements match*. Finally, we make each thread compare their indices with their right-side neighbor, to detect if they are the start of a segment and must dump out their results.

Results: [1] We compare the VSR design with three baselines: SpMV without workload-balancing or parallel-reduction, and with each single optimization. When tested on the SuiteSparse [16] benchmark, VSR-based SpMV exceeds the other three kernels (baseline and two optimizations individually) on 40.8% of the matrices.

2.1.2 *Vector-type Dense-row Loading (VDL).*

Motivation: This is an optimization on the straightforward implementation of parallel-reduction SpMM. Based on parallel-reduction SpMV, parallel-reduction SpMM can be completed with N-times SpMV on every column of the dense matrix. However, this straightforward implementation suffers from inefficient memory operations. In parallel-reduction SpMV, threads in a warp load dense-vector elements according to positions of the non-zeros in the sparse row. The addresses of target dense elements are not contiguous, leading to poor locality.

Approach: We explore the insight that each sparse-matrix element $A[i,k]$ can be multiplied with all elements in a dense-matrix row, i.e. $X[k,n]$ for $1 \geq n \leq N$. Our design is shown in Figure. 3. We make each thread load multiple dense-matrix elements using the vector-type memory operations, i.e. loading with type *float2/float4*. Take *float2* as an example, each thread holding a sparse non-zero $A[i,k]$ will load $X[k,0], X[k,1]$ together in one instruction, multiply $A[i,k]$ with both, and accumulate the results on two different partial-sums. This design can increase the amount of effective data per request, at least the size of *float2/float4*.

Results: We set $N = 2$ and compare *float2*-loading VDL against performing two SpMVs. We synthesize 27 matrices with the R-MAT generator [17] using various size, sparsity and distribution

[1] All ablation study in this section is conducted on RTX3090. We present overall results for three GPUs in Section. 3.

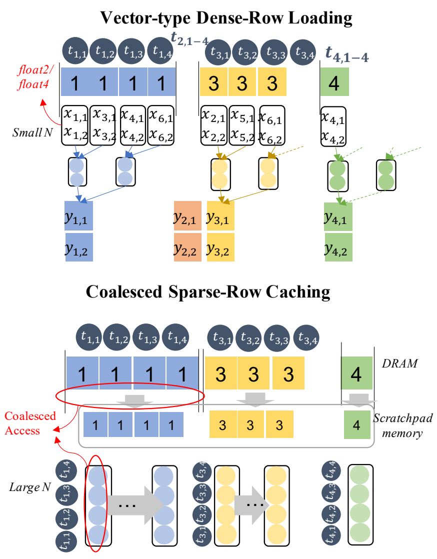

Figure 3: Illustration of VDL and CSC.

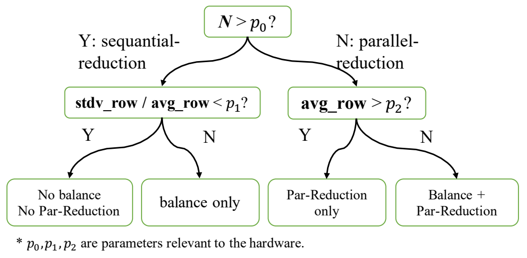

Figure 4: Adaptive strategy to select kernels.

parameters. On this micro benchmark, VDL performs 1.89× better than the two-SpMV solution.

2.1.3 Coalesced Sparse-row Caching (CSC). **Motivation:** Observe there are two benefits of parallel-reduction against sequential-reduction. Firstly, loading the sparse elements is more efficient. Threads in a warp load non-zeros in a sparse row that are stored contiguously, which is an ideal data access pattern on GPUs. Secondly, arithmetic operations are performed by more threads to exploit parallel resources. SpMM has plenty of parallelism, making the parallelized arithmetic operation less necessary. However, we would like to keep the benefit of efficient load operations.

Approach: We implement vectorized loading of sparse rows under a sequential-reduction scheme by exploiting the shared memory. The shared memory is a scratchpad memory accessible by all threads within the same warp. We first load the non-zeros in one row in a coalesced way, i.e. load *warp_size* non-zeros with one instruction. We store these elements into the shared memory. Next, we make parallel threads iterate over the cached elements, and compute on different columns of the dense matrix, as in Figure. 3.

Results: We set $N = 128$ and compare CSC against SpMM implementation with pure sequential-reduction. On the micro benchmark described in the previous part, CSC brings average 1.20× speedup.

2.2 Adaptive Two Principles To Problems

The previous part introduces our novel methods to optimize workload-balancing and parallel-reduction for SpMV and SpMM. This part introduces our second contribution, which is a kernel selection strategy for various inputs, i.e. the sparsity pattern and N. We derive our selection algorithm from the following three insights:

- **Insight 1**: Parallel-reduction is necessary for efficiently loading the sparse matrix, but suffers from poor memory locality in loading the dense matrix when N gets larger.
- **Insight 2**: Workload-balancing is necessary for matrices with skewed non-zero distribution, but involves extra overhead for well-balanced matrices.
- **Insight 3**: A common benefit of two principles, the increased parallelism and resource occupancy, becomes unnecessary when the total amount of work is large, either because of large size of the sparse matrix, or because of large N.

Insight 1 states how N affects whether to use parallel-reduction. As discussed in our VDL optimization in the previous part, parallel-reduction benefits from efficient loading of the sparse matrix, while exploiting locality in the dense matrix when non-zeros are clustered. However, when N increases, parallel-reduction suffers from poor locality when loading elements from the dense matrix if N is large. On the contrary, sequential reduction benefits from a friendly access pattern to the dense matrix. Enhanced with our CRC optimization, sequential-reduction outperforms parallel-reduction by a large margin due to efficient memory operations.

Insight 2 states how the sparsity pattern affects whether to use workload-balancing. Intuitively, if a sparse matrix has severely imbalanced non-zero distribution, parallelizing different rows to different threads results in imbalanced workloads, which is why workload-balancing is necessary. Statistical metrics such as standard deviation can be used to capture this row-wise imbalance.

Insight 3 further states how N and the sparse matrix size affects the effectiveness of two principles. A benefit of parallel-reduction is to parallelize operations and data loading compared with sequantial version, but this becomes unnecessary when the workload is heavy and the resource occupancy is high. GPU has a limited amount of computation resources, and when the total workload is large, GPU performs works with multiple waves of threads. In this case, the workload imbalance is less serious a problem, since new threads will occupy the resource of the early-finishing threads.

Our kernel selection strategy follows the three insights, and is shown in Figure. 4. We make decisions with the following steps: First, we consider N to choose between parallel- or sequential-reduction, following insight 1. We choose parallel-reduction for SpMV, and also SpMM with $N \leq 4$, when parallel-reduction can benefit from the vector-type data loading we propose in Section 2.2. When $N > 4$, we apply sequential reduction. Next, we decide whether to apply workload balancing according to sparse matrix features. According to insight 1, workload-balancing benefits matrices with imbalanced non-zero distribution in different rows. Hence, a high standard-deviation of the row-length, *stdv_row*, is a positive signal for us to apply workload-balancing. According to insight

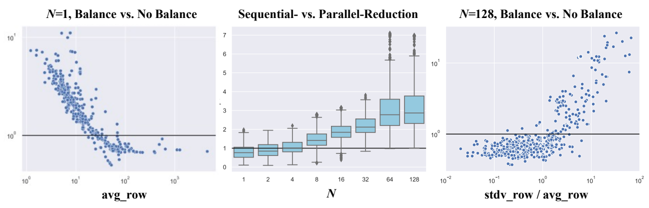

Figure 5: Validation of our adaptive strategy. Middle: we observe parallel-reduction is beneficial only when N is small. Left: When $N = 1$, how workload-balancing benefit correlates with the average row-length grows. Right: When $N = 128$, how workload-balancing benefit correlates with both the variance and average of row-lengths.

3, if the total amount of work is large, workload-imbalance becomes less serious. Thereby, a large mean row-length *avg_row* is a negative signal because it indicates a large number of non-zeros and amount of work. We combine the two signals and use *stdv_row*/*avg_row* as the metric and empirically decide the threshold. Finally, for parallel-reduction kernels, we observe that a large *avg_row* greatly benefits the imbalanced parallel-reduction because short rows cause the idle discussed in our VSR optimization. Thereby, for parallel-reduction cases, we use *avg_row* to decide whether to apply workload-balancing. This completes our kernel selection strategy.

3 RESULTS

3.1 Kernel Performance

Platform: We conduct experiments on three GPUs: Nvidia Tesla V100 (Volta architecture 7.0 compute capability), Nvidia RTX 2080 (Turing architecture 7.5 compute capability), Nvidia RTX 3090 (Ampere architecture 8.6 compute capability).

We use SuiteSparse matrix collection [16] to benchmark the kernel performance. We compare with two baselines: cuSPARSE in CUDA toolkit version 11.2, and ASpT [13], the state-of-the-art SpMM implementation. We test N from 1 up to 128. There are two ways to use our kernels: either to profile and select the best implementation off-line, or use our adaptive strategy for online selection. In most HPC and GNN applications, the sparse matrix can be profiled statically to select out the best kernel for iterative algorithms. We show the results of both approaches in Figure. 6.

The "ours" in Figure. 6 denotes the best of four implementations. For SpMV, our kernel is **1.14×, 1.07×, 1.11×** against cuSPARSE on three GPUs respectively. For SpMM, our kernel exceeds cuSPARSE by **1.26-1.41×, 1.09-1.44× 1.22-1.57×** on three GPUs respectively. Against ASpT [13] on settings they support, we achieve **1.21× 1.14×, 1.16×** when $N = 32$ and **1.18×,1.14×,1.06×** when $N = 128$.

3.2 Adaptive Strategy

The "ours with rule-based" in Figure. 6 denotes the performance of kernels selected by our adaptive strategy, as stated in Section. 2.2. Compared with the optimal choice under different N, the kernel selected by our rules is **6%-22%, 1%-8%, 7%-12%** slower. Compared with the last four bars, which denote the four implementations, our strategy consistently outperforms the solution of always choosing the same kernel. In addition, we can observe that the best option among four individual kernels changes along with N. Thereby, when averaged among all N, the best single-kernel solution involves **68%, 86%, 76%** performance loss, while our kernel selection rules involve only **12%, 5%, 10%** performance loss.

4 RELATED WORK

Categorized by optimization methods:

Workload-balancing: The workload-balancing is achieved by row-binning, i.e. coarsely grouping sparse-matrix rows into buckets with similar total workload [6, 9], or merge-path, i.e. fine-grained segmentation of arithmetic and memory operations to ensure balancing [10]. Yang *et al.* [12] extend MergePath to MergeSpmm, and in addition propose RowSplit which omits workload-balancing. The authors select between MergeSpmm and RowSplit according to an average length of sparse-matrix rows. We extend this approach to SpMM with arbitrary N, and improve the selection heuristics according to profiles on a more comprehensive benchmark.

Parallel-reduction: Bell & Garland [11] proposed the CSR-vector SpMV algorithm to load and compute on sparse elements in a vectorized fashion, in contrast to CSR-Scalar [11] which uses sequential reduction. CSR-Stream [9] exploits the parallel loading but sequential reduction of segments. Yang *et al.* [12] combines the vectorized loading with sequential reduction through a SIMD-shuffle primitive in CUDA. We greatly improved their implementation through exploiting the shared memory, as detailed in our prior publication [14].

Specialized sparse format: Compressed formats like ELL, block-CSR, HYB [15] improves data access efficiency but at the cost of padded zeros and wasted computation. Specialized formats are also used to mark clustered elements and expose chances for data re-use [13]. Format-dedicated optimizations are orthogonal to the design space we explore in this paper.

Categorized by application, the most-related prior work includes SpMM optimizations for graph neural networks [8] and data analytics [18]. SpMV acceleration is also studied for scientific computing [13] and graph analytics [3]. SpMV and SpMM, as alternatives to GEMV, GEMM in sparse DNN, is widely studies on GPU [19, 20], CPU [21] and accelerators [22, 23]. Our work brings significant benefit to GNN and data analytics (matrix factorization), while also bring improvements the long-studied SpMV problem in scientific and graph domains.

5 CONCLUSION

We extend the principles and practice of workload-balancing and parallel-reduction to a wide range of SpMV/MM problems. We propose three optimizations: VSR effectively combines the two techniques in SpMV, VDL and CSC optimize memory operation efficiency for SpMM with small and large N respectively. Altogether, we provide highly optimized implementations of two design principles in SpMV/MM. The second main contribution is strategies to selectively apply the two techniques based on low-cost metrics such as row-length average and deviation. Optimal choices with our optimizations exceed the cuSPARSE library by **1.07-1.57×** on different problems and GPUs. If the off-line profile is prohibited,

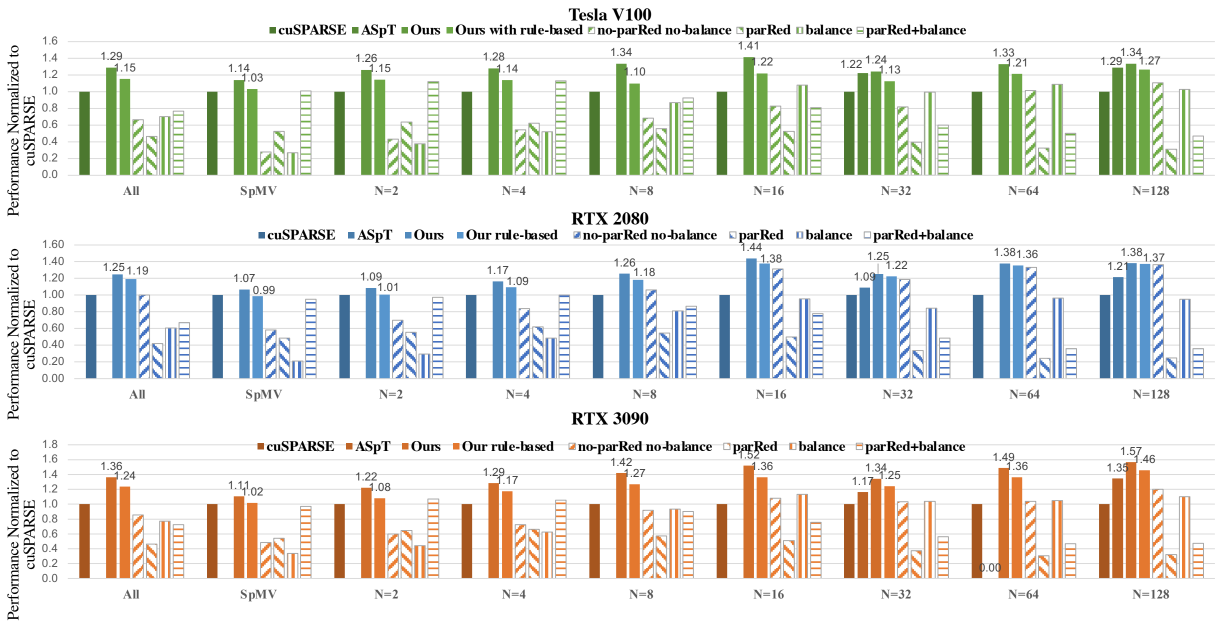

Figure 6: Kernel performance compared against cuSPARSE [15] and ASpT [13], tested on SuiteSparse benchmark [4].

our selection strategy involves only **5-12%** performance loss on average, compared with a minimum 68% if using a single kernel. We are collaborating with CogDL [2], a popular graph learning framework to apply our research.

REFERENCES

[1] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[2] Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Peng Zhang, Guohao Dai, et al. Cogdl: An extensive toolkit for deep learning on graphs. *arXiv preprint arXiv:2103.00959*, 2021.

[3] Xintian Yang, Srinivasan Parthasarathy, and Ponnuswamy Sadayappan. Fast sparse matrix-vector multiplication on gpus: Implications for graph mining. *arXiv preprint arXiv:1103.2405*, 2011.

[4] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, USA, 2nd edition, 2003.

[5] Wencong Xiao, Jilong Xue, Youshan Miao, Zhen Li, Cheng Chen, Ming Wu, Wei Li, and Lidong Zhou. Tux2: Distributed graph computation for machine learning. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 669–682, USA, 2017. USENIX Association.

[6] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020.

[7] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[8] Yuwei Hu, Zihao Ye, Minjie Wang, Jiali Yu, Da Zheng, Mu Li, Zheng Zhang, Zhiru Zhang, and Yida Wang. Featgraph: A flexible and efficient backend for graph neural network systems. *arXiv preprint arXiv:2008.11359*, 2020.

[9] Joseph L Greathouse and Mayank Daga. Efficient sparse matrix-vector multiplication on gpus using the csr storage format. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 769–780. IEEE, 2014.

[10] Duane Merrill and Michael Garland. Merge-based parallel sparse matrix-vector multiplication. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 678–689. IEEE, 2016.

[11] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the conference on high performance computing networking, storage and analysis*, pages 1–11, 2009.

[12] Carl Yang, Aydın Buluç, and John D Owens. Design principles for sparse matrix multiplication on the gpu. In *European Conference on Parallel Processing (Euro-Par)*, pages 672–687. Springer, 2018.

[13] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P Sadayappan. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, pages 300–314, 2019.

[14] Guyue Huang, Guohao Dai, Yu Wang, and Huazhong Yang. Ge-spmm: general-purpose sparse matrix-matrix multiplication on gpus for graph neural networks. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2020.

[15] Basic linear algebra for sparse matrices on nvidia gpus. =https://developer.nvidia.com/cusparse.

[16] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. 38(1), 2011.

[17] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446. SIAM, 2004.

[18] Changwan Hong, Aravind Sukumaran-Rajam, Bortik Bandyopadhyay, Jinsung Kim, Süreyya Emre Kurt, Israt Nisa, Shivani Sabhlok, Ümit V Çatalyürek, Srinivasan Parthasarathy, and P Sadayappan. Efficient sparse-matrix multi-vector product on gpus. In *International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 66–79, 2018.

[19] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. *arXiv preprint arXiv:2006.10901*, 2020.

[20] Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 31–42, 2020.

[21] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14629–14638, 2020.

[22] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. In *International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016.

[23] 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 *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70. IEEE, 2020.