Accelerating Attention Based Models via HW-SW Co-Design using Fine-Grained Sparsification.

Abhimanyu Rajeshkumar Bambhaniya*, Amir Yazdanbakhsh†, Suvinay Subramanian‡, Tushar Krishna*
*Georgia Institute of Technology. *abambhaniya3@gatech.edu,tushar@ece.gatech.edu*
†Google DeepMind. *ayazdan@google.com*
‡Google. *suvinay@google.com*

***Abstract*—This paper proposes FIne-Grained Sparsification (FIGS), a novel architecture for accelerating attention-based models using N:M structured sparsity. Existing hardware accelerators focus on optimizing compute to achieve ideal processing element (PE) utilization but ignore the implications of higher input bandwidth. FIGS overcomes this challenge by leveraging techniques like grouping and reusing input data to reduce required input bandwidth, achieving high PE utilization while minimizing on-chip interconnect area. The paper also proposes FIGS-Train, a sparsity training recipe that improves the accuracy of N:M structured sparse attention models.**

I. Introduction

Attention-based models have become increasingly popular in recent years due to their ability to focus on relevant information while processing input data. They are particularly effective in natural language processing (NLP) [3], [7], [10], image recognition [25], and speech recognition, code generation [4]. The attention mechanism allows these models to assign varying levels of importance to different parts of the input data, resulting in improved accuracy and efficiency.

Despite their effectiveness, attention-based models have become increasingly complex and computationally demanding, with some models containing billions of parameters [23]. This growing size of the models has led to longer training and inference times, limiting their applicability in practical settings. Various techniques, such as parallel processing, model compression, and low-precision arithmetic, have been proposed to overcome these challenges. However, these techniques have their limitations and often come at the cost of reduced accuracy [2], [5], [11]–[13], [18], [21], [22], [24], [28], [29].

Sparsity is an increasingly prevalent technique for accelerating attention-based models [6], [26] . Sparsity aims to reduce the number of parameters and computations required by the model by identifying and removing redundant or less important connections. N:M structured sparsity is a particularly interesting technique as it removes a fixed number of weights for each block of weights in the model [29]. This approach is more hardware-friendly and easier to implement in hardware accelerators than other sparsification techniques, such as random or magnitude-based pruning. By using this technique, it is possible to accelerate the execution of attention-based models while maintaining high accuracy levels [1], [20], [22].

Although N:M sparsity has been shown to be an effective technique for accelerating attention-based models, the existing hardware accelerators, such as S2TA [21], STA [20], and VEGETA [15], focus primarily on optimizing the compute to achieve ideal processing element (PE) utilization, while ignoring the implications of higher input bandwidth. Recent research [27] has shown that the required input bandwidth for N:M accelerators scales up with M, which can lead to significant challenges in designing hardware that can effectively support this higher bandwidth. Figure 1 shows the increase in the on-chip interconnect area as the bandwidth required increases [16], [17].

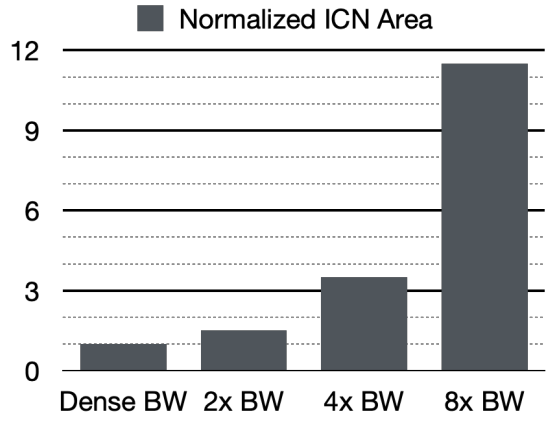

Fig. 1. **Growing cost of supporting higher on-chip bandwidth in 7nm.**

Thus we propose FIGS: FIne-Grained Sparsification, novel architectures that can improve PE efficiency without increasing the on-chip bandwidth requirement. Our architectures leverage techniques like grouping and reusing input data to reduce the required input bandwidth. By doing so, they can achieve high PE utilization while minimizing the on-chip interconnect area. We also, show a potentially promising training approach, FIGS-Train, that helps improve the accuracy of an N:M structured sparse attention model. We develop even more efficient accelerators for attention-based workloads by combining fine-grained sparsity with FIGS hardware architectures designed to improve PE efficiency.

To summarize, we make the following contributions:

- We propose a new hardware microarchitecture, FIGS, that helps accelerate N:M structured sparse models without increased input BW requirements.
- We propose a new sparsity training recipe, FIGS-Train,

that can be accelerated using N:M structured HW.
- We compare FIGS microarchitecture, with current SOTA N:M accelerators.
- We train attention-based models using FIGS-Train, achieving better accuracy than current state-of-the-art N:M structured sparsification techniques.

II. FIGS Architecture

A. Processing Elements(MAC PE)

Each processing element comprises of β multiplier units. The FIGS architecture works in a systolic array-inspired weight-stationary format, storing all sparse weights in a MAC PE register and metadata. We have bitmask as metadata, as that would result in the lowest overhead for the structured sparse case. The MAC PEs send out β metadata to its corresponding *Swap Reg*, and it gets the appropriate input activation input. Each multiplier and adder completes the MAC operation and forwards an output to the downstream MAC PE. We position β as a configurable parameter, as with newer technology nodes, it is possible to have fewer registers between PEs [14], [17]. This helps in reducing the area of the complete engine.

B. Swap Reg

Swap Reg acts as input staging unit. It takes in γ input words from the engine input or the previous swap reg. Each *Swap Reg* registers these γ input words in flops. The *Swap Reg*'s main task is to provide appropriate data to the MAC PEs. Using the meta-data provided by the MAC PEs, the *Swap Reg* would generate β corresponding output words. Multiple neighboring *Swap Regs* can also talk to each other and exchange their stored input words. Depending on the architectural parameter,α, a single *Swap Reg* takes all the input activation of the α *Swap Regs*, making a total of $\alpha \cdot \gamma$ input words. This means each *Swap Regs* has to select the β output activation words from $(\alpha + 1)\gamma$ inputs. Thus a bigger α/β/γ value results in a bigger *Swap Reg*.

C. The FIGS Engine

We build the complete FIGS engine using MAC PEs and *Swap Reg* as building blocks. A FIGS engine has $N_{Rows} = R$ and $N_{Cols} = C$. All the MAC PEs are accompanied by a *Swap Reg* at their input. Thus the whole FIGS engine has R * C MAC PEs and *Swap Regs*.

γ depends on the maximum allowable on-chip bandwidth capacity and required sparsity support. β will depend on the engine's required running frequency and the synthesis technology node. Theoretically, this can be any value between 1 and R. Finally, α depends on the maximum sparsity support required. We calculate $\alpha = \frac{M_{max}}{N_{min} \cdot \gamma}$.

Once, we have all the configuration parameters of the FIGS engine, we get an engine with the total number of MAC units = $R * C * \beta$. The engine takes in $C * \gamma$ words as input per cycle and generates $R * \beta$ output words per cycle.

D. Dataflow

With the full FIGS engine now described, we move on to how the FIGS accelerates flexible N:M sparsity without changing the input BW. In Figure 3, we show 2:4 sparse matrix with input BW twice of the dense architecture. Each row of A matrix with 2:4 structured sparsity is mapped to a single column of MAC PEs. Each column of MAC PEs takes β of rows of matrix A and generates β partial sum. To support 2:4 sparsification, with have need $\alpha = \frac{M}{N \cdot \gamma} = 1$.

In the current configuration, two neighboring *Swap Regs* can exchange the input activations. We can see this in action, when in row 1 of the engine, the First MAC PE, takes Element 3 as input based on the metadata. This element has been provided by the *Swap Reg* of row 2. In each cycle, the *Swap Reg* forwards $\gamma = 2$ input activation to the next *Swap Reg* but gives out $\beta = 4$ activations to the MAC PE. Using this mapping, we ensure, that PEs are fully utilized while allowing higher N:M sparsity, for the same γ. The full execution of this would take 'T' cycles.

Now, in order to execute 1:2 or 1:1 sparsity, we can still use the same mapping; just the *Swap Regs* would not need to swap data at that time.

III. FIGS-Train

A. The Recipe

Now that we understand the abilities of the FIGS accelerator, we propose FIGS-Train, a specialized training recipe capable of efficient sparsification. Figure 4 shows the training schedule for FIGS-Train. The intuition is that by keeping the same number of non-zeros in the row, we keep reducing the block size. It is important to understand each progressive step would be a subset of the previous sparsification. For example, in the figure, we 1:4 $\in$ 2:8 $\in$ 4:16 $\in$ 8:32. Thus, this approach helps the gradients gather locally before pruning which helps achieve better accuracy.

B. Training methodology and results

We applied this technique for weight sparsification at various locations in two attention-based models(ViT [9] and Swin V2 [19]. We trained these models on imagenet-1k [8] dataset with various amounts of sparsity. Table I summarizes the results for the 2 models at different sparsity levels. We compare the results with SR-STE [29], the current SOTA technique of N:M structured sparsification.

We found the two feed-forward layers in each layer are the most robust. Hence we always sparsified the weights in those layers. Next, we also pruned the weights of Query and Key matrices. We found pruning the weights of the Value matrix had the biggest impact on the network accuracy; Hence we did not prune the Value layer's weights.

IV. Evaluations

A. Experimental Setup

We convert the feed-forward layer of the encoder block of 2 attention-based models to GEMM operations. We analyze

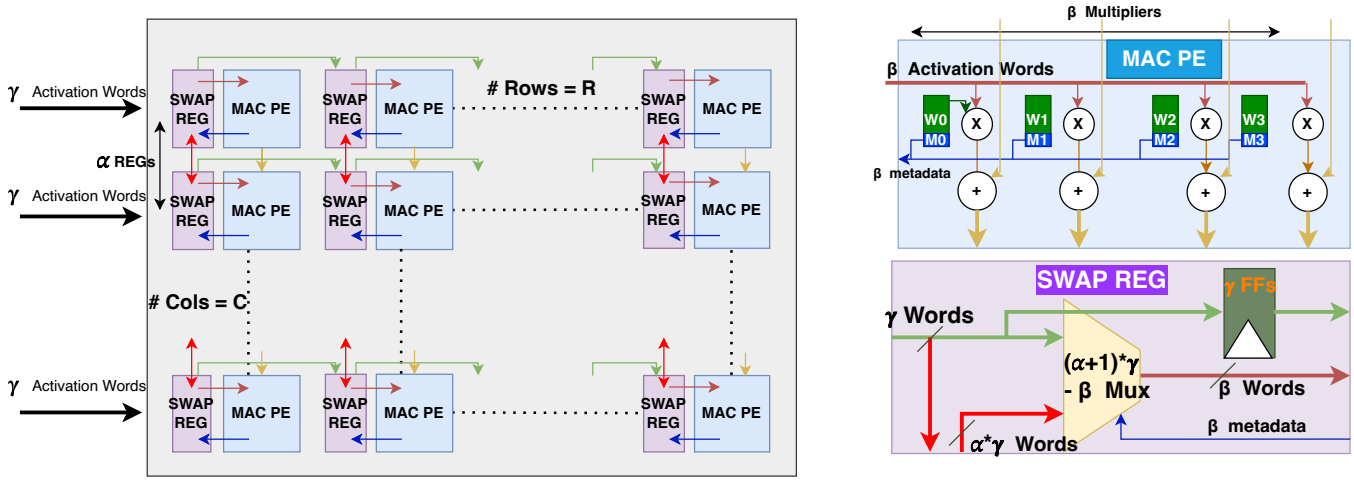

Fig. 2. **FIGS Micro architecture design for $\beta = 4$ and $\alpha = 2$.**

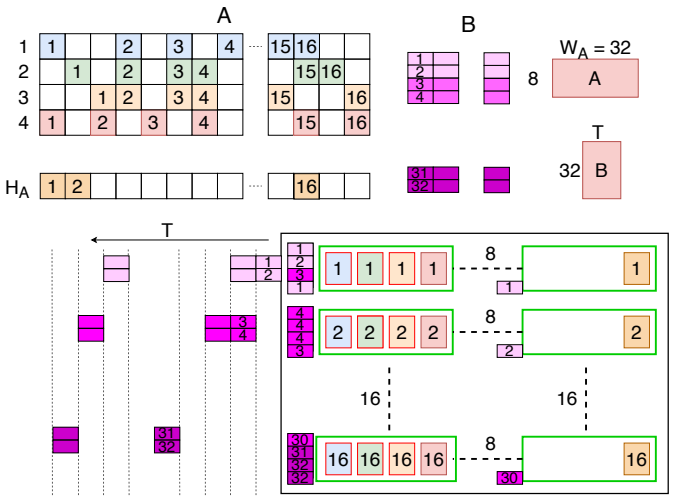

Fig. 3. **Dataflow for runing 2:4 structured sparse matrix multiplication with #Cols = 16, #Rows = 8, $\gamma = 2$, $\beta = 4$ and $\alpha = 1$. Note this other design SOTA design requires a minimum $\gamma = 4$.**

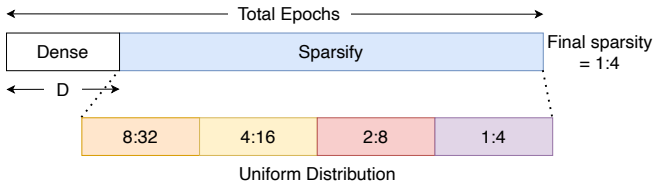

Fig. 4. **Breakdown of training epochs for FIGS-train recipe.**

the training time for the layer when trained with the FIGS-Train recipe. We compare the runtime of these operations on a systolic array, SOTA N:M accelerators like STA/Vegeta, and compare them with three configurations of FIGS. We assume the same number of MACS = 512 MACs for all architectures.

TABLE I
SPARSITY TRAINING RESULTS WITH FIGS. FF MEANS SPARSITY IS PRESENT IN THE FEED-FORWARD LAYERS. QK MEANS THE SPARSITY IS PRESENT IN THE QUERY AND KEY WEIGHTS.

Model	Sparsity	ViT	Swin V2
Dense	/	76.369	83.45
SR-STE	1:8(FF)	77.869	81.437
FIGS	1:8(FF)	**78.175**	**81.466**
SR-STE	1:8(FF + QK)		81.218
FIGS	1:8(FF + QK)		**81.438**

B. Performance Analysis

Figure 5 shows the runtime of the SWIN and VIT Feed-Forward layers on the 5 accelerators. We normalized the runtime using the longest runtime (runtime for a dense systolic accelerator). We first observe that a systolic accelerator with no acceleration ability for sparse operations results in the highest runtime. STA/VEGETA is a state-of-the-art architecture for accelerating N:M structured sparsity, but these are bottlenecked by the input bandwidth to the compute array. We assume 4x the dense input bandwidth for all architectures. With this, STA/VEGETA can only accelerate layers with 1:4 or higher amount of sparsity. Thus for ViT, it can accelerate only the last sparsification phase with 1:4 sparsity, while for SWIN, it can accelerate the last 2 sparsification phases with 2:4 and 1:2 sparsity. Compared to these, FIGS architectures with the same input BW perform much better. FIGS($\alpha = 1$) can accelerate the sparsity ratio upto 2:8, FIGS($\alpha = 2$) can accelerate the sparsity ratio upto 4:16, and FIGS($\alpha = 3$) can accelerate the sparsity ratio upto 8:32.

Hence, we observe that all FIGS configurations perform much better than existing SOTA accelerators. For ViT with 1:4 sparsity, FIGS with ($\alpha = 3$) achieve 4.4x speedup over S2TA and Vegata, while 2.41x speedup for Swin v2 with 1:2 sparsity.

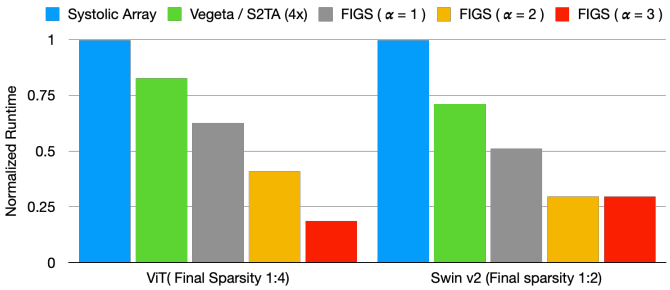

Fig. 5. **Runtime Comparison of fine-grained runtime sparsification.**

V. Future Work

Some of the potential directions we are considering for this work are:-

- Our proposed FIGS technique provides a promising direction for accelerating attention-based models with N:M structured sparsity. One possible direction for future work is to investigate the efficacy of FIGS-Train in larger models and models other than those for image classification.
- We would also explore how to club this technique with other existing sparsification techniques like block sparsity, butterfly sparsity, etc.
- We would also like to do an area-energy-performance analysis of FIGS uarch to cap its ability and view its feasibility in realistic implementations.

VI. Conclusion

In this work, we proposed FIne-Grained Sparsification (FIGS) techniques to accelerate attention-based models without increasing the on-chip bandwidth requirement. We showed that our proposed FIGS microarchitecture can achieve high PE utilization while minimizing the on-chip interconnect area. Our proposed FIGS-Train training recipe also showed promising results in improving the accuracy of N:M structured sparse attention models. We compared the performance of our proposed FIGS microarchitecture with state-of-the-art N:M accelerators and showed that it outperforms existing approaches.

References

[1] "Tensorfloat-32 in the a100 gpu accelerates ai training, hpc up to 20x," 2019, https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

[2] M. Behnke and K. Heafield, "Losing heads in the lottery: Pruning transformer attention in neural machine translation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 2664–2674. [Online]. Available: https://aclanthology.org/2020.emnlp-main.211

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021.

[5] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin, "The lottery ticket hypothesis for pre-trained bert networks," 2020.

[6] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[7] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[10] EMNLP, "Emnlp 2017 second conference on machine translation (wmt17)," https://www.statmt.org/wmt17/, 2017.

[11] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2943–2952.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[13] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1398–1406.

[14] Intel, "Presentation deck: Intel architecture day 2021," 2021, https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf.

[15] G. Jeong, S. Damani, A. R. Bambhaniya, E. Qin, C. J. Hughes, S. Subramoney, H. Kim, and T. Krishna, "Vegeta: Vertically-integrated extensions for sparse/dense gemm tile acceleration on cpus," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 259–272.

[16] N. P. Jouppi, , C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[17] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten lessons from three generations shaped google's tpuv4i : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[18] S.-C. Kao, A. Yazdanbakhsh, S. Subramanian, S. Agrawal, U. Evci, and T. Krishna, "Training recipe for n: M structured sparsity with decaying pruning mask," *arXiv preprint arXiv:2209.07617*, 2022.

[19] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, "Swin transformer v2: Scaling up capacity and resolution," 2022.

[20] Z.-G. Liu, P. N. Whatmough, and M. Mattina, "Systolic tensor array: An efficient structured-sparse gemm accelerator for mobile cnn inference," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 34–37, 2020.

[21] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 573–586.

[22] NVidia, "Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt."

[Online]. Available: https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

[23] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," *arXiv preprint arXiv:2202.05924*, 2022.

[24] W. Sun, A. Zhou, S. Stuijk, R. Wijnhoven, A. O. Nelson, hongsheng Li, and H. Corporaal, "Dominosearch: Find layer-wise fine-grained n:m sparse schemes from dense neural networks," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[25] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[26] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, apr 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3530811

[27] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical approach to sparse tensor accelerator modeling," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1377–1395.

[28] Y. Zhang, M. Lin, Z. Lin, Y. Luo, K. Li, F. Chao, Y. Wu, and R. Ji, "Learning best combination for efficient n: M sparsity," *arXiv preprint arXiv:2206.06662*, 2022.

[29] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li, "Learning n:m fine-grained structured sparse neural networks from scratch," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=K9bw7vqp_s