

ECE 552: Final Project Report

Micro-architectural Support for Packed Sparse Matrix Multiplication on Hybrid Systolic Arrays

David Raibaut

Abstract—This project pulls inspiration from two novel architectural methods for increasing NPU efficiency: Hybrid Systolic Arrays (HSA) [1] and Sparse-TPU [2]. HSA provides a way of reducing chip area by combining matrix and vector multiplies into a single systolic array. Sparse-TPU provides hardware support for sparse-packed matrices thus increasing performance and reducing area. I developed unit tests and benchmarks for the components to ensure functional correctness, compare HSA, sparse-tpu and their novel combination with more traditional methods and analyze the performance and power efficiency impact.

Index Terms—Systolic Array, Energy Efficient Computing, Hardware Accelerator, LLM inference.

I. INTRODUCTION AND PROBLEM STATEMENT

User demand for Large Language Model (LLM) inference has grown rapidly throughout the past years [3]. On-edge inference has emerged as a powerful alternative to cloud-based solutions, thanks to its lower cost, decreased latency and heightened security with comparable performance for daily tasks. However, the limited memory capacity, energy constraints and sequential query model pose significant challenges for successfully deploying LLMs on edge, where user-facing applications require minimal latency (dictated by the decode stage [1]) and a reduced energy footprint (dictated by external memory accesses (EMA)) [1].

A proposed solution is the use of sparse-matrix LLMs [2], such as SparseLLM [4], SparseGPT [5] and KMZ [6]. These models compress larger trained models to reduce their number of parameters and hence reduce the memory requirements for the model and the number of operations to produce their computation by making their composing matrices sparser.

Other proposals to enhance edge inference efficiency include the use of Hybrid Systolic Arrays (HSA) [1]. These units function as an intermediate point between Matrix Processing Units (MPU, used during pre-fill stage) and Vector Processing Units (VPU, used during decode stage), thus eliminating the need for both. This has been shown to effectively reduce the area and thus energy consumption and maintain comparable latency.

In this project I plan to explore a novel HSA, by including hardware support for packed sparse matrices, essentially exploring the performance-efficiency tradeoff exposit by a combination of two previous methods. I will also implement sparse-matrix support on clock-gated MPUs and VPUs to show the advantage provided by the HSA architecture, and conversely non-sparse matrix supported MPUs, VPUs and

HSAs to show the advantage provided by the HSA structure independently.

II. RELATED WORK

There have been multiple proposals for on-edge LLM inference without the use of dedicated hardware accelerators, focusing mostly on CPU optimisation [7], [8], [9], efficient model compression and quantization [10], [11] and even custom frameworks for edge-specific model design [12], [13].

Even then, due to the tight energy and performance efficiency constraints on edge devices, particularly mobile devices such as phones or wearables, most of these techniques incur large tradeoffs, such as requiring models to be very small [7] ($\sim 1\text{B}$ parameters), or finding significant performance degradation at sub-4-bit quantization [10].

Thus, given the rise in user-demand [3] for low-latency high-performing LLMs, the most likely path forward seems to be dedicated hardware accelerators [14] optimised for this specific workload.

In this project I will mostly refer to the HSA [1] and Sparse-TPU [2] as they are the two most relevant sources due to their closeness with the proposed implementation. Other notable areas of research include heterogeneous systems [15], [16], as a way of attempting to reduce the memory bandwidth bottleneck at the cost of architectural flexibility, compressed KV-cache implementations [17] at the cost of added latency and KV cache reuse optimisations [18] at the cost of decreased accuracy.

III. RESEARCH QUESTIONS

- How can existing HSA hardware be adapted to support packed sparse matrix multiplication?
- How much performance and energy gains does packed sparse matrices provide in practice?
- How much performance and energy gains does HSAs provide in practice, what is the performance loss/tradeoff?
- How much performance and energy gains does the novel packed sparse matrix HSAs provide in comparison with the baseline (VPU and MPU combo)?

IV. PROPOSED SOLUTIONS

The proposed solution is that of a HSA structure with added hardware support for sparse-packed matrices. The use of the HSA structure allows for a lower chip area (since separate MPUs and VPUs are not required for the pre-fill and decode

stages respectively) thus leading to lower leakage power and overall energy use. Hardware-support for sparse-packed matrices increase performance by significantly reducing the number of operations required to be computed (i.e. the number of MAC units), at the cost of some accuracy (due to the packing), and thus reducing overall SA latency. Moreover, by clock-gating the individual MAC units, the dynamic power of the whole chip can be reduced thus decreasing energy use too.

In this project I set out to explore whether this novel idea of a HSA with support for sparse-packed matrices (referred to throughout as SHSA) truly leads to the performance and energy improvements over single HSA or single sparse-packing, how much each of these individual improvements contributes and what the accuracy tradeoff is.

A. Vector, Matrix and Hybrid Dataflows in Detail

The VPU performs Matrix-Vector Multiplication (MVM) during the decode stage, and the MPU performs Matrix-Matrix Multiplication (MMM) during the pre-fill stage. There are many dataflows possible for these different operations, all functionally equivalent, but with different tradeoffs. The two most salient type of dataflows are:

- Output-stationary (OS): each MAC is responsible of holding the matrix entry at its position, where the weight and activation data ‘flow’ over the architecture
- Weight-stationary (WS aka HSA-style): each MAC is pre-loaded with its weight value and the activation and partial sums flow over the MAC array, with the final answer being received in waves at the end of the MAC array.

Because a HSA unit should be able to operate both MVM and MMM operations, the microarchitecture for the MPU and VPU units should be similar enough that they could be hybridised.

I have decided to implement all my units in WS mode for the following reasons:

- 1) Minimise EMA during decode stage. In output stationary, each cycle weights are streamed leading to longer latencies or larger bandwidth, as during the decode stage I use $\mathcal{O}(1)$ activations but $\mathcal{O}(N)$ weights. In a WS model I can reuse the loaded weights across operations or pipeline their loading and save bandwidth.
- 2) OS would mean $\mathcal{O}(N^2)$ MACs. Although this would be fine during pre-fill, there would be significant under-utilisation of resources during the decode stage, where at most $\mathcal{O}(N)$ values are being calculated. *Note:* This is also true to some extent for the WS model, but MAC units can be gated trivially and moreover I have found a way to pipeline the vector multiplications in a weight-stationary model to incur negligible under-utilisation (see figure 1).
- 3) OS would require significant differences in the dataflow and gating style in MVM and MMM modes for the HSA unit leading to more complex control circuitry reducing the area and energy advantages that the hybrid approach provides.

Thus I have opted for a WS dataflow model. My proposed method has the systolic ‘wavefront’ move diagonally for MMM units and horizontally for MVM units (note: there is

also a way to eliminate weight-loading wavefronts if we can reuse weights across operations). This is what has allowed me to pipeline the weight-loading, since at any given instance only units on the wavefront (i.e. a column for MVM or a diagonal for MMM) are being operated. See figure 1 for a graphical representation.

B. Packed vs non-packed dataflows

The original STPU paper [2] proposes a more flexible approach to handling sparse-matrix packing than traditional fixed-packed 2:1 methods. However, in doing this, significant complexity is introduced for the sake of maintaining the flexibility of a variable sparsity factor which does not map well onto the hybrid architecture.

Moreover, the paper introduces sparse-matrix packing for the decode stage, since MMM operations introduce significant structural hazards in the systolic data flow, which is why I have adapted my roadmap to eliminate the sparse-MPU unit, as implementing such a unit is well beyond the scope of this project.

Due to the increased complexity of the original STPU paper, I have proceeded to create my own more simplified approach for adapting VPUs and HSA (in MVM mode). This novel method, although incurring a slight loss in accuracy, allows for either a $\sim 2x$ gain in performance or a $\sim 2x$ gain in energy saving, with only a slight increase in die area.

The crux of the packed (sparse) vs non-packed microarchitectures is as follows. Suppose we are working with a sparse model (i.e. weights have been pruned during training) with the following weights matrix:

$$W = \begin{bmatrix} 5 & 0 & 1 & 7 \\ 0 & 4 & 0 & 1 \\ 0 & 2 & 5 & 0 \\ 4 & 0 & 0 & 0 \end{bmatrix}$$

On a WS model, any 0-weight entry would map to an essentially idle MAC unit. To reduce the sparsity, we pack the matrix through column merging: Each element’s row is kept invariant and elements must be tagged with the column they came from. This allows for correction of the severe underutilisation of MAC units. However, I noticed two key things:

- Reducing EMA is our priority. Since most models are low-bit quantized ($Q < 8$), adding tags increases EMA if $Q < \log_2 N$. Since weight matrices are typically large ($N \sim 2^{10} - 2^{14}$ for small models), including tags guarantees added EMA and latency.
- Since each MAC now may need to keep track of values from multiple different columns and make sure they match with the activation value, MAC complexity increases.

So, my simplification is to merge only adjacent columns, and discard the lower-valued weight arbitrarily. Since we only merge adjacent columns, columns now need to be tagged solely with the parity of their column index (one bit) and

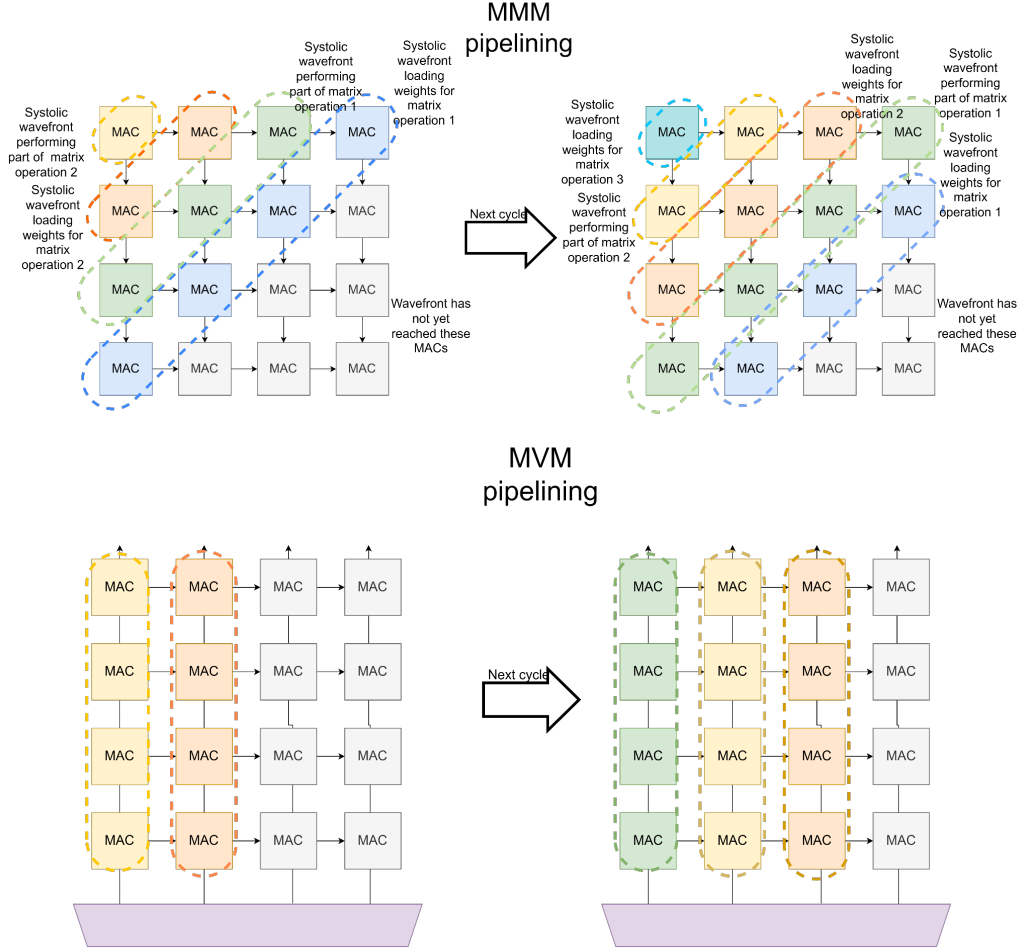


Fig. 1: The pipelining model I came up with to mitigate the under-utilisation. Shown is the MMM and MVM modes. The arrows show the flow of partial sums (can be either horizontal or vertical, but I typically take it to be horizontal) and partial sums (taken here to be left to right). The ‘master’ circuit driving this MPU/VPU is responsible of keeping track and reading the outputs for each of the different cycles when they are ready

the need for additional memory elements in each MAC is eliminated. For example, the matrix above would become:

$$W = \begin{bmatrix} 5, 0 & 7, 1 \\ 4, 1 & 1, 1 \\ 2, 1 & 5, 0 \\ 4, 0 & 0, 0 \end{bmatrix}$$

The only added tradeoffs we incur with my simplification is the following:

- Slight reduction in accuracy from sparse packing: for example, in above matrix, row 1’s column 3’s value had to be dropped. See section V.D for how this is calculated.
- 1 additional EMA per cycle, as each column now needs 2 activation values, which can be achieved with an extra read port to the activation memory, which is negligible in comparison with the $\mathcal{O}(N)$ accesses for weights.
- 1 additional input port and mux in each of the MACs: the additional input port is for the second activation value, and the mux will use the column parity tag as a select bit so that the appropriate activation value is used for

the multiplication. This is far less than the added MAC complexity from the STPU paper.

V. EVALUATION

A. Methodology

The hardware units are implemented in a C++ software simulator (custom-built from scratch, cycle accurate) as well as SystemVerilog [19] modules, with Xilinx Vivado to synthesize them. They will be tested on the AMD Artix A7 FPGA with 63400 LUTs, 126800 FFs, and 4860Kb of SRAM. Given the memory and logic slice constraints on the Artix A7, I originally estimate I would be able to fit a 32 by 32 SA unit, along with the accompanying driving logic, which per my experimentation was in fact correct. However, due to the lackluster computing power on the laboratory PCs, synthesising a 32 by 32 SA unit along with accompanying logic would take around 1h30min, which would have severely harmed the development time. So I settled for an 8x8 SA unit.

Aside from the C++ and SystemVerilog testbenches, a proof-of-concept of only one unit (the VPU) was implemented to a fully working status with a custom UART transmission

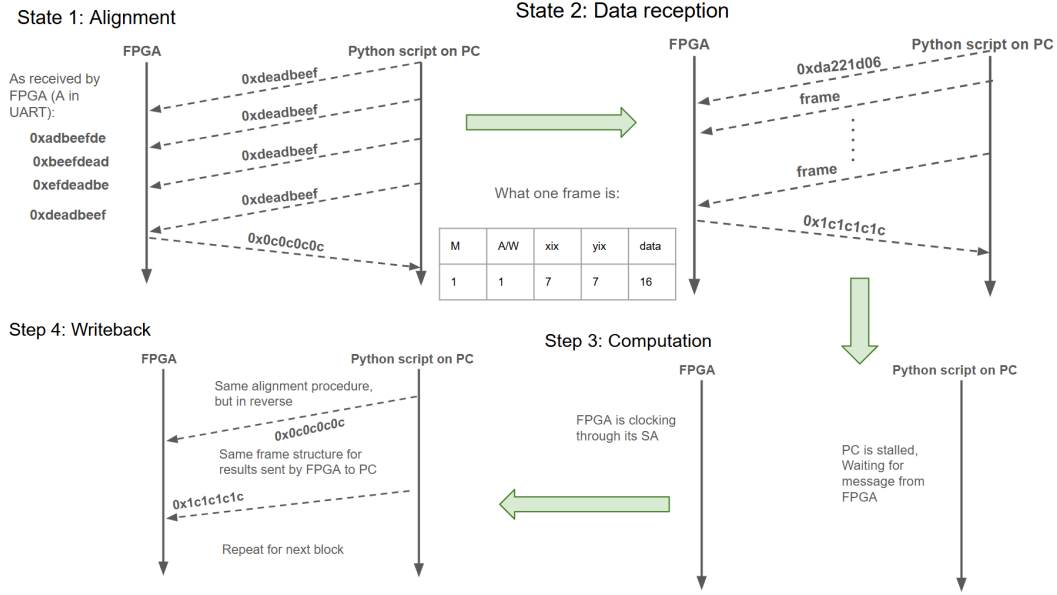


Fig. 2: FSM states for the UART protocol and communication between host PC and FPGA over UART Serial

protocol as a coprocessor to the host PC (an accompanying python driver was written), so that microbenchmarks consisting of randomly generated matrices could be run and verified. Figure 2 shows a diagrammatic overview of the transmission protocol and figure 3 shows a block diagram of the whole stack.

The proof-of-concept can be easily extrapolated for other units.

Figure 4 shows an 2x2 MAC circuit of the HVPU implementation in verilog as generated by Vivado, and figure 5 shows such a unit correctly performing the operation:

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 13 \end{bmatrix}$$

As you can see, this unit correctly implements the pipelined weight loading that was previously mentioned. *Note:* The unit size (i.e. rows and columns of MAC units) and bus bit width was parametrised in Verilog.

You can access the GitHub repo for this project at this link: <https://github.com/DR-Reg/ECE-552-project>. Note that the verilog sources lie under FPGA_impl/FPGA_impl.src/sources_1/new/

B. Functional Validation

- Unit testing:* Test benches in SystemVerilog are provided for each of the aforementioned units (GitHub). All units with implemented test benches function correctly.
- Software implementation:* The C++ implementation itself has been made cycle accurate and has been functionally validated and traced by hand for all units that were implemented. Due to the large scope of the project, some of the initial units had to be skipped.
- Proof of concept device:* A python program working with the custom driver generates 100 random numpy matrices and computes the multiplication on the host PC as well as

through the FPGA via the transmission protocol. All runs were successful with only a negligible amount (less than 10% of the runs, each consisting of 100 transmissions, so less than 0.1% overall) having issues due to transmission errors.

C. Physical Characteristics

Using the Vivado Synthesis tool, the following utilisation (area), power and clock frequency characteristics were found:

1) *Clock Frequency:* All units were able to meet timing constraints for a 100MHz clock. Due to the cross-domain clocking, a negative worst negative slack was provided by Vivado, but since all the critical paths were across clock domains, these can be effectively ignored. All models had essentially an equivalent critical path which consisted of the distance between latches across MACs in the SA within the fast clock domain.

2) *Power:* Figures 6, 7, 8, 9, 10 show the power data for the main units when an 8x8, INT16 quantised model is synthesised.

The VPU shows the highest on-chip power (0.252W), with 23% going to DSP slices. This makes sense since the column broadcasting model of the VPU implies that a larger count of DSP slices must be utilised at a given time. Meanwhile, the rest of the units display a lower, equal power consumption (0.199W), which is surprising, since a lower power-consumption would be expected for the sparse HSA unit than the regular HSA unit. My best explanation for this is that my verilog has some strange mistake which leads the implementation to infer an over-simplification of the circuit when it is attached to the transmission circuit. This over-optimisation from the synthesiser leads to parts of the logic to be deleted for the MPU, SVPU and HSA models, thus why the power consumption is so low for all of them, and why the DSP slices are not even displayed on the graphs.

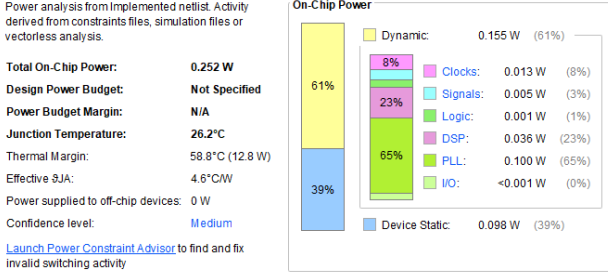


Fig. 7: VPU unit power analysis

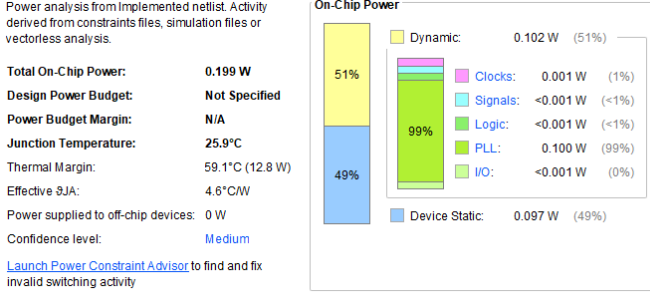


Fig. 8: HSA unit power analysis

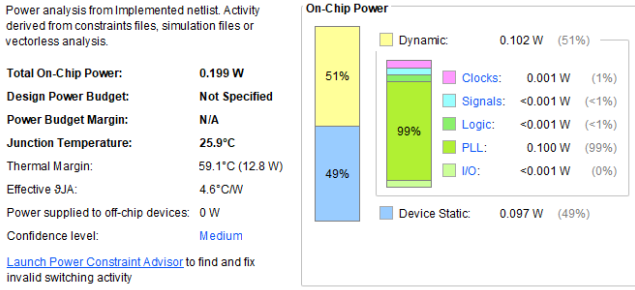


Fig. 9: SVPU unit power analysis

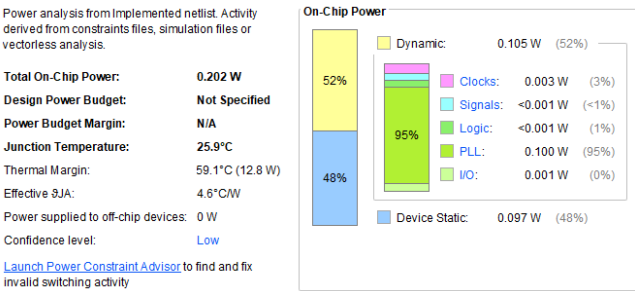


Fig. 10: SHSA unit power analysis

confirmed through C++/verilog testbenches), I was unable to catch these errors during the functional validation.

D. Performance Impact

Performance impact was compared across multiple metrics. Baselines for performance are the bare MPU and VPU units.

1) *Accuracy for sparse-packing*: Accuracy was calculated according to the following formula. Given the result from the

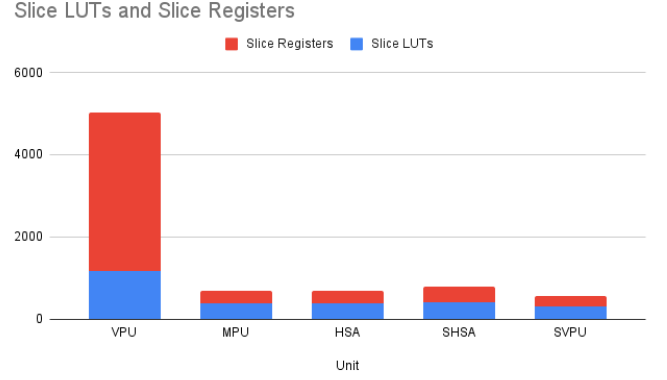


Fig. 11: Utilisation composition and total counts for each of the units

original matrices $C = A \times B$ and the sparse-packed, quantised $C' = A' \times B'$, we extract the element wise percentage difference (matrix P):

$$p_{ij} = \frac{|c_{ij} - c'_{ij}|}{c_{ij}} \times 100$$

Then the metrics P_{25}, P_{50}, P_{75} are computed, corresponding the lower quartile, median and upper quartile across all entries p_{ij} . This is an accurate way of measuring the percentage difference as it takes into account all of the elements in the matrix simultaneously, and also gives a sense of the spread across them.

Figure 12 shows the percent difference of a sparse-packed quantised, a non-sparse-packed-quantised, and a sparse-packed non-quantised against the original matrix computation (the 0% difference would be the baseline of no quantisation or packing). This was generated from the first FFN BERT transformer model, with 100 random activation inputs being tested. As is visible accuracy is very poor for all three different settings.

There are a few reasons for why the pruned (packed-sparse) INT16 model has a median at 200% difference. Firstly, we see a pruned (packed-sparse) FP32 already has this poor performance. This is because the pruner was created to prune in a tile-based manner as opposed to a column-based manner.

Moreover, there is severe problems with the quantisation scheme also producing highly wrong values. I believe that this is because the activations were not being equally quantised causing a disparity between them and erroneous data points.

2) *Raw Performance (cycles)*: Figure 13 shows absolute performance in cycles for each unit and 14 shows the relative performance. These were computed through statical analysis after the C++ implementation was finished, since the number of cycles it takes is completely deterministic. Essentially, it can be easily shown that an N-wide MPU unit takes $2N - 1$ cycles, and a VPU unit takes N cycles. The sparse versions of these take half of these for the computation cycles.

Similarly, the number of load-store cycles are computed statically, with MPU taking $2N^2$ and the VPU taking $N^2 + N$. The sparse units each take $\frac{B}{B+1}$ number of external memory accesses, where B is the bit width. As is visible on the graphs, as N increases the EMAs (which already dominated

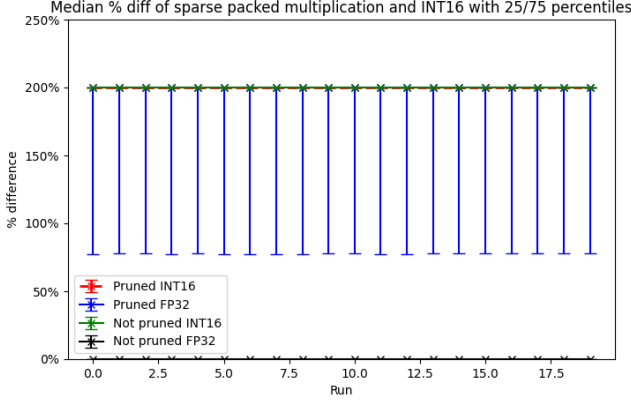


Fig. 12: SHSA unit power analysis

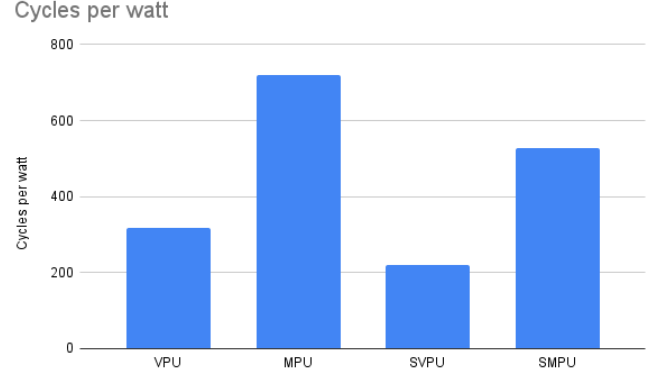
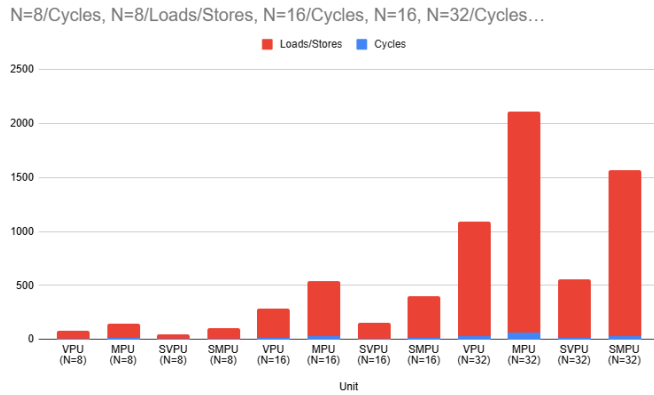
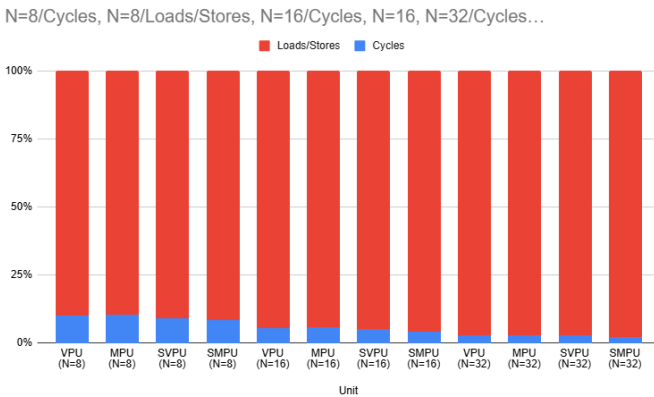


Fig. 15: Power efficiency for different units

Fig. 13: Performance graph comparing absolute cycles across sparse and non-sparse units for $N = 8, 16, 32$ Fig. 14: Performance graph comparing relative cycles across sparse and non-sparse units for $N = 8, 16, 32$

cycle counts for low N) increase their importance, due to the quadratic growth of memory accesses but linear growth of cycles. Even though sparse models alleviate this to an extent, their impact is only up to a factor and as N increases, the quadratic term still dominates.

Originally, I had planned to run a GPT-2 transformer model on one of my units. However, since the transmission protocol had about a 10 second overhead (with FPGA computation

taking solely 0.00037s), and we operate on 8×8 blocks, since $N = 762$ for the GPT-2 transformer, this leads to each layer taking around 9200 blocks. This would mean that during, e.g., the decode stage, a single would take roughly 25 hours to generate, which is not realistic or useful to perform benchmarks on.

I also explored smaller LLM models, for example, TinyStories with $N = 64$, which would require around 10 minutes per word which would still be unreasonable, hence the lack of any real benchmarks (as opposed to the plethora of microbenchmarks).

3) *Power efficiency (cycles/W)*: Since cycles are directly proportional to the runtime (as the clock frequency is fixed), it makes sense to compute power efficiency in cycles/W. Taking $N=8$, figure 15 shows the power efficiency of each of the units. Note that any performance metric need not be shown for hybrid units, since the mode switching implies they are equivalent to either a VPU or an MPU at any given time.

As is visible from the graph, indeed the sparse units are both more performant and power efficient across the board, with the SVPU unit achieving $\sim 30\%$ better power efficiency and SMPU achieving $\sim 30\%$ power efficiency too.

VI. DISCUSSION AND FUTURE STEPS

While this project was successful in implementing a full-stack, fully-functional VPU unit with PC-FPGA implementation, due to the tight time constraints, the rest of the units (while still implemented in Verilog and the C++ sim) did not have the transmission capabilities on the FPGA. While this did not affect the static analysis of performance, it severely undermined the power analysis, since the lack of transmission capabilities led to the synthesiser over-optimising and thus providing values which were erroneous.

Future steps may include:

- 1) Implementing correct data transmission for the rest of the units and using ‘real-world’ (as opposed to simulation) performance metrics and validation.
- 2) Switching from UART protocol to a faster one (e.g. ethernet) to avoid the 10 second overhead.
- 3) Adding support to the transmission protocol for pipelining (i.e. pre-loading data into a buffer and have the PC

communicate data while computation goes on, with the data being pipelined as shown in the above diagrams - this was not possible for the current transmission protocol, so each block operated independently)

- 4) Investigate effects of using pruners tuned for column pruning or different ways of sparse-packing matrices

VII. REFERENCES

REFERENCES

- [1] C.-T. Chen, H. Mun, J. Meng, M. S. Abdelfattah, and J. sun Seo, "Hybrid systolic array accelerator with optimized dataflow for edge large language model inference," 2025. [Online]. Available: <https://arxiv.org/abs/2507.09010>
- [2] A. A. S. F. D.-H. P. A. R. H. Y. Y. C. R. D. T. M. Xin He, Subhankar Pal, "Sparse-tpu: Adapting systolic arrays for sparse matrices," 2020. [Online]. Available: https://tnm.engin.umich.edu/wp-content/uploads/sites/353/2020/08/2020.6.sparse-tpu_ics2020.pdf<https://arxiv.org/abs/2507.09010>
- [3] A. Fradkin, "Demand for llms: Descriptive evidence on substitution, market expansion, and multihoming," 2025. [Online]. Available: <https://arxiv.org/abs/2504.15440>
- [4] G. Bai, Y. Li, C. Ling, K. Kim, and L. Zhao, "Sparsellm: Towards global pruning for pre-trained language models," 2024. [Online]. Available: <https://arxiv.org/abs/2402.17946>
- [5] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," 2023. [Online]. Available: <https://arxiv.org/abs/2301.00774>
- [6] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 821–834. [Online]. Available: <https://doi.org/10.1145/3297858.3304028>
- [7] H. Zhang and J. Huang, "Challenging gpu dominance: When cpus outperform for on-device llm inference," 2025. [Online]. Available: <https://arxiv.org/abs/2505.06461>
- [8] C. Zhang, X. Zhu, L. Chen, T. Yang, E. Pan, G. Yu, Y. Zhao, X. Wu, B. Li, W. Mao, and G. Han, "Enhancing llm inference performance on arm cpus through software and hardware co-optimization strategies," *Integrated Circuits and Systems*, vol. 2, no. 2, pp. 49–57, 2025.
- [9] H. Shen, H. Chang, B. Dong, Y. Luo, and H. Meng, "Efficient llm inference on cpus," 2023. [Online]. Available: <https://arxiv.org/abs/2311.00502>
- [10] S. Cao, L. Ma, and T. Cao, "Advances to low-bit quantization enable llms on edge devices," 2025. [Online]. Available: <https://arxiv.org/abs/2311.00502>
- [11] K. Freund, "How to run large ai models on an edge device," 2023. [Online]. Available: <https://cambrian-ai.com/how-to-run-large-ai-models-on-an-edge-device/>
- [12] ggml org, "llama.cpp," 2025. [Online]. Available: <https://github.com/ggml-org/llama.cpp>
- [13] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Ubaweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," 2019. [Online]. Available: <https://arxiv.org/abs/1906.08172>
- [14] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc 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, 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. Snellman, 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," 2017. [Online]. Available: <https://arxiv.org/abs/1704.04760>
- [15] L. Chen, D. Feng, E. Feng, Y. Wang, R. Zhao, Y. Xia, P. Xu, and H. Chen, "Characterizing mobile soc for accelerating heterogeneous llm inference," 2025. [Online]. Available: <https://arxiv.org/abs/2501.14794>
- [16] M. Seo, X. T. Nguyen, S. J. Hwang, Y. Kwon, G. Kim, C. Park, I. Kim, J. Park, J. Kim, W. Shin, J. Won, H. Choi, K. Kim, D. Kwon, C. Jeong, S. Lee, Y. Choi, W. Byun, S. Baek, H.-J. Lee, and J. Kim, "lanus: Integrated accelerator based on npu-pim unified memory system," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Volume 3, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 545–560. [Online]. Available: <https://doi.org/10.1145/3620666.3651324>
- [17] A. Liu, J. Liu, Z. Pan, Y. He, G. Haffari, and B. Zhuang, "Minicache: Kv cache compression in depth dimension for large language models," 2024. [Online]. Available: <https://arxiv.org/abs/2405.14366>
- [18] J. Thomson, A. Shah, and L. Tewari, "Introducing new kv cache reuse optimizations in nvidia tensorrt-llm," 2025. [Online]. Available: <https://developer.nvidia.com/blog/introducing-new-kv-cache-reuse-optimizations-in-nvidia-tensorrt-llm/>
- [19] "Ieee standard for systemverilog—unified hardware design, specification, and verification language," *IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017)*, pp. 1–1354, 2024.

VIII. APPENDIX

A. *Division of Labour*

All work has been done by me (David Raibaut).