ECE 552: Project Status 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 will develop 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 energy efficiency impact.**

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 exposited 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$1B 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?
- How much performance and energy gains does the novel packed sparse matrix HSAs provide in practice?

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

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

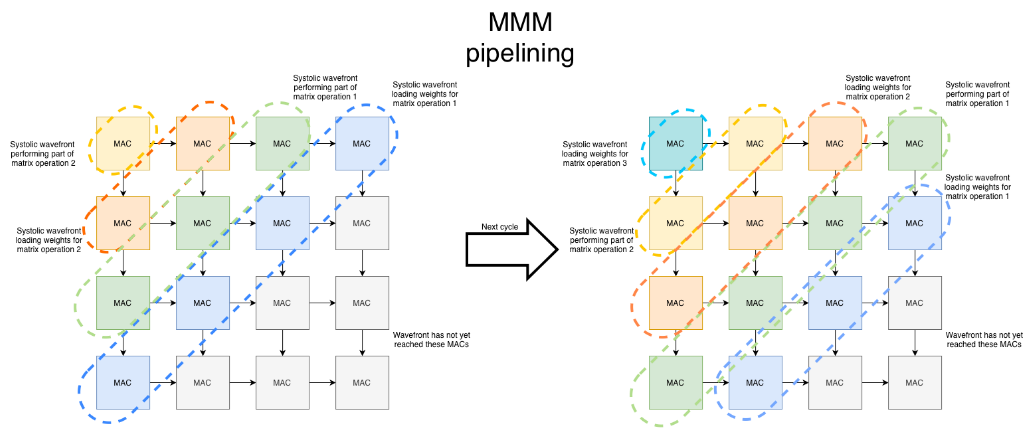

MMM pipelining
Systolic wavefront performing part of matrix operation 1
Systolic wavefront loading weights for matrix operation 1
Systolic wavefront performing part of matrix operation 2
Systolic wavefront loading weights for matrix operation 2
MAC
Wavefront has not yet reached these MACs
Next cycle
Systolic wavefront loading weights for matrix operation 2
Systolic wavefront performing part of matrix operation 1
Systolic wavefront loading weights for matrix operation 3
Systolic wavefront loading weights for matrix operation 1
Systolic wavefront performing part of matrix operation 2
Wavefront has not yet reached these MACs

Fig. 1. The pipelining model I came up with to mitigate the under-utilisation. Shown is the MMM mode, which is the most complex, the MVM is identical except wavefronts are columns as opposed to diagonals. 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 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.
- 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 activatino 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 - Current Status

A. Methodology

The hardware units will be implemented in both 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 estimate I will be able to fit a 32 by 32 SA unit, along with the accompanying driving logic, however this is subject to change. Assuming this, the weight and activation SRAMs will be 64kB and 4kB each in total.

Table I shows the status of the units being implemented. An 'FV' indicates units that have been implemented and functionally validated (to a large extent, but not exhaustively - this will come at a later milestone), a 'NS' indicates a unit that has not yet been started, and a 'S-P%' indicates a unit that has been started and is P% percent of the way there.

TABLE I
PROJECT STATUS

Unit	C++?	Verilog?
MAC-WS	FV	FV
MAC-OS	FV	NS
VPU-OS	FV	NS
MPU-OS	FV	NS
VPU-WS	FV	FV
MPU-WS	FV	NS
HSA	FV	NS
SVPU	S-50%	NS
SHSA	NS	NS
NPU[1]	NS	NS

Figure 2 shows an 2x2 MAC circuit of the HVPU implementation in verilog as generated by Vivado, and figure 3 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 to see all my work so far ($\sim$ 1000 lines of C++, and $\sim$ 200 lines of verilog) at this link: https://github.com/DR-Reg/ECE-552-project

B. Functional Validation

(a) *Unit testing*: Test benches in SystemVerilog will be provided for each of the aforementioned units. Currently the single VPU-WS unit (called VpuHsa.v) has an accompanying test bench (VpuHsa_tb.v), with the rest of units pending.

(b) *Software implementation*: The C++ implementation itself has been made cycle accurate and has been functionally validated and traced by hand for all units that have

[1]Due to the large ambition and scope of this project, implementing the NPU unit has been moved to a secondary objective, as this is not critical in showing the performance difference between different VPU/MPU/HSA units, but rather a 'bells-and-whistles' unit required to run an actual LLM model such as RetNet on the units. They can instead be benchmarked using matrices directly extracted from the weight/activation values of such models.

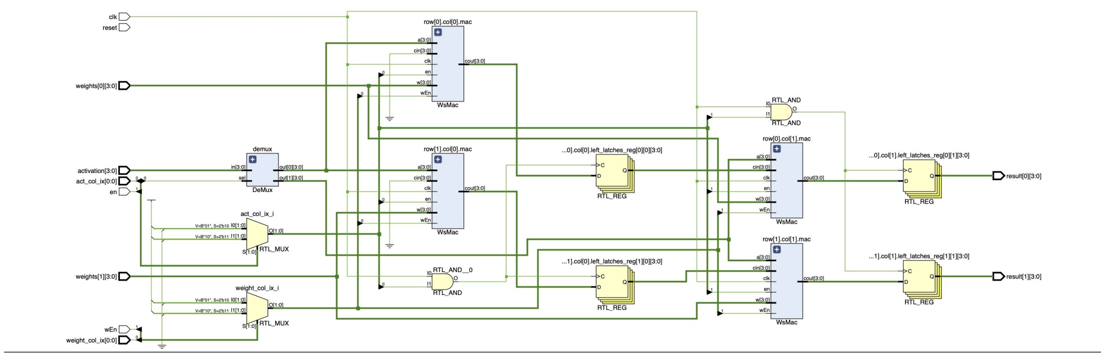
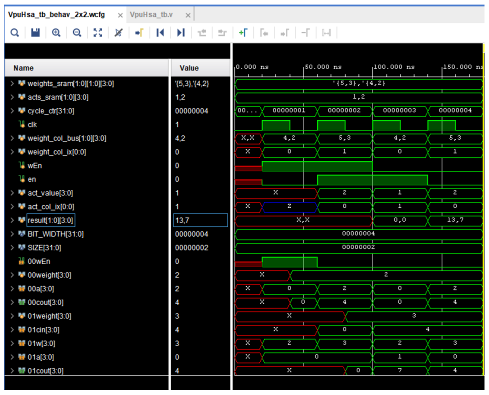


Fig. 2. The 2x2 VPU-WS unit running on 4-bit quantisation



Name	Value
weights_sram[1:0][1:0][3:0]	'{5,3},'{4,2}
acts_sram[1:0][3:0]	1,2
cycle_ctr[31:0]	00000004
clk	1
weight_col_bus[1:0][3:0]	4,2
weight_col_ix[0:0]	0
wEn	0
en	0
act_value[3:0]	1
act_col_ix[0:0]	1
result[1:0][3:0]	13,7
BIT_WIDTH[31:0]	00000004
SIZE[31:0]	00000002
00wEn	0
00weight[3:0]	2
00a[3:0]	2
00cout[3:0]	4
01weight[3:0]	3
01cin[3:0]	4
01w[3:0]	3
01a[3:0]	0
01cout[3:0]	4

Fig. 3. 2x2 VPU-WS trace showing functional validity

been implemented so far (∼90% of units). Most of the important files for this can be found in cpp_impl/include

C. Physical Characteristics

Vivado software provides tools for finding power: both compiler estimates and XADC System Monitor readings will be provided for power.

Vivado also provides tools for assessing the maximal frequency, namely the timing report after synthesis, which includes the 'Worst Negative Slack' (WNS) metric. In the case of a negative WNS, an analysis of the critical paths and iterative redesign/testing will be performed, until all designs meet the timing requirements. Ideally, all structures should operate at the same frequency, since supporting NPU package (either hardware implemented or software emulated) will be identical as control variables. However, in the case that they are not clocked equally, start-to-end latency and total energy expenditure will be used as the performance metrics.

Vivado synthesis also provides data for logic cell and BRAM utilisation which will be used as an estimate for chip area.

D. Performance Impact

Performance impact will be compared across multiple metrics:

1) Accuracy (for sparse-packing, % diff vs non-packed)
2) Raw performance (TOPS)
3) Energy efficiency (pJ/Token)
4) Power efficiency (TOPS/W)
5) Area (mm2) - directly from Vivado

Raw performance will be approximated by estimating the operations per clock cycle (or simply end-to-end latency in seconds in the case where different frequencies have to be used), and energy efficiency will be calculated from the power efficiency (which can be estimated using Xilinx's XPower tool). Accuracy can be found by computing the percent-difference per-element between packed and non-packed results.

VI. Revised Milestones

A slight modification is done to the milestones, since the NPU supporting package has become a secondary priority:

1) **Milestone 1: C++ Software Implementation (90% done):** I expect to finish this by the end of this week
2) **Milestone 2: Unit Implementation in Verilog (15% done):** I expect to finish this by the middle of next week
3) **Milestone 3: Functional Validation and Performance Testing (40% done):** I have been validating my units as I made them, but formal validation will be performed once all units are finished and performance metrics extracted.
4) **Milestone 4: Final Evaluation and Reporting (Not Started):** Analyze performance and write report, once milestone 3 is finished.
5) **Bells and Whistles: NPU Package (Not Started)** If possible, get these units to work so I can showcase my hardware accelerator running a real workload start to finish.

Although it may seem I am slightly behind with respect to my original planning, I am in fact ahead, since my decision to make the C++ software implementation cycle accurate means that the software is almost a 1:1 representation of the hardware, so making the Verilog modules takes significantly less time (1-2h per unit). Making the C++ implementation has allowed me to refocus the project to a novel idea for architectural support of packed matrices and is more realisable in the given time frame.

VII. Division of Labour

All work has been and will continue to be done by me (David Raibaut).

VIII. 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.pdfhttps://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. Uboweja, 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. 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," 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, "Ianus: 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.