Design of Hardware Accelerated Co-Processor for Neural Inference Computations

Kushagra Chauhan and R. Vidhyaand S. Nagadevi
Department of Computing Technologies, SRM Institute of Science and Technology, SRM Nagar, Kattankulathur, 603203, Tamil Nadu, India

Keywords: ASIC, Neural Networks, Hardware Acceleration, Large Language Models, Energy Efficiency.

Abstract: The rapid growth in AI model complexity has increased demand for hardware capable of high- efficiency inference with lower energy consumption. While GPUs have traditionally driven AI advancements, their progress has plateaued with compute density improving by only 15% over four years. This work develops a highly optimized ASIC accelerator specifically for LLaMA 3.1, implementing critical inference operations through specialized computational units and memory hierarchy. The de sign achieves 100% efficiency in matrix multiplication operations and 99.90% efficiency in attention computations, demonstrating significant improvements over traditional GPU implementations. Results show sustained throughput of 256 operations per cycle in matrix multiplication and 7.99 operations per cycle for attention mechanisms, with memory bandwidth utilization of 1024GB/s. This research presents a sustainable solution for large-scale AI inference deployment, addressing both computational efficiency and energy consumption challenges.

1 INTRODUCTION

The significant proliferation of AI technologies has led to great demands for specialized hardware to perform efficient and sustainable intensive model inference. While GPUs form part of traditional computing solutions and have widely played significant roles in the advancement of AI capabilities, they get used for general applications, leading to suboptimal performance in AI applications. As model sizes continue to balloon exponentially, energy and computational requirements in data centers are growing at an unprecedented pace (Y. LeCun et al. 2015).

By the end of 2024, Microsoft plans to deploy approximately 1.8 million AI chips, primarily NVIDIA H100 GPUs, resulting in a total power consumption of around 1260MW for GPU processing alone. When accounting for an additional 75% energy overhead for cooling, networking, and other supporting services, Microsoft's AI infrastructure will require 2205MWh. Similarly, Meta's AI infrastructure will consume around 420MW of power for GPU computation and 735MWh in total with overheads. The economics of scale of AI is moving fast. The costs of training state-of-the-art models run to billions of dollars, generating over $10 billion in inference costs. At this scale, even an efficiency gain of 1% would justify investments of $50-$100 million in custom chip development. Performance of ASICs has shown benefits of 10-100x over GPUs, as demonstrated in Bitcoin mining applications (N. P. Jouppi et al. 2017)

2 RELATED WORKS

The von Neumann bottleneck presents a fundamental challenge in neural accelerators, where bandwidth between processors and memory becomes an overwhelming performance barrier (D. Markovi´c et al., 2020). This architectural limitation particularly affects neural network accelerators where the volume and frequency of data movement between memory and processing units create significant performance barriers.

Chen et al. 2016 introduced Eyeriss, demonstrating significant energy efficiency improvements through careful dataflow optimization. Their spatial architecture achieved 1.4× to 2.5× better efficiency for AlexNet layers

Chauhan, K. and Nagadevi, R. V. S.
Design of Hardware Accelerated Co-Processor for Neural Inference Computations.
DOI: 10.5220/0013921700004919

In *Proceedings of the 1st International Conference on Research and Development in Information, Communication, and Computing Technologies (ICRDICCT'25 2025) - Volume 5*, pages 5-10
ISBN: 978-989-758-777-1

and up to 3.7× fewer DRAM accesses compared to contemporary solutions.

A significant milestone in neural network acceleration came with Google's Tensor Processing Unit, analyzed in detail by Jouppi et al. 2017. Their implementation validated the crucial role of systolic arrays in neural network acceleration, particularly for repeated matrix multiplication operations common in inference tasks. The TPU architecture demonstrated that careful optimization of matrix operations and weight reuse strategies could achieve up to 3.5 times better performance per watt compared to contemporary GPU solutions.

The introduction of Transformer architecture by Vaswani et al. revolutionized natural language processing while presenting new challenges for hardware acceleration. The attention mechanisms require specialized computation patterns distinct from traditional neural network operations, introducing additional complexity in hardware realization (S. Han et al. 2016).

3 SYSTEM ARCHITECTURE

Overall Architecture: The accelerator architecture centers around a lightweight core that manages control flow and coordinates specialized computational units. The system comprises three primary components integrated through a sophisticated FIFO-based communication system: Sophisticated memory hierarchy with 32 independent banks, High-throughput matrix multiplication unit based on systolic arrays, Specialized attention module supporting 64 parallel heads. Figure 1 show the High-level architecture of the proposed accelerator showing major components and their interconnections.

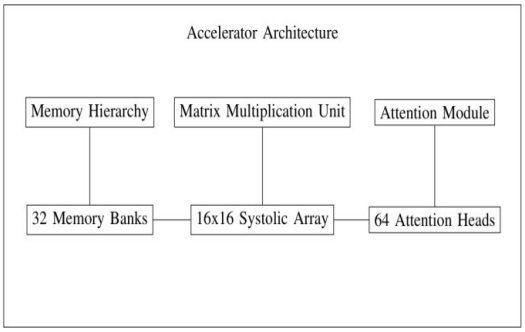

Figure 1: High-level architecture of the proposed accelerator showing major components and their interconnections.

Memory Hierarchy Implementation: The memory subsystem implements a multi-bank architecture comprising 32 independent memory banks, optimized for the high bandwidth requirements of neural network inference. The design includes:

Bank Organization: The bank organization features 32 independent memory banks that operate in parallel, enabling efficient and simultaneous data processing. Each bank is equipped with 64-byte cache lines specifically optimized for matrix operations, ensuring high performance in computational tasks that rely heavily on such calculations. Additionally, the system incorporates bank-level buffering, which significantly reduces access latency by allowing smoother and faster data retrieval. To further enhance efficiency, advanced conflict resolution mechanisms are integrated, minimizing delays and ensuring seamless operation across the parallel banks.

Memory Controller: The memory controller is designed with a sophisticated scheduling system that ensures efficient management of data flow across the memory banks. It employs a round-robin bank access allocation strategy, which evenly distributes access to the banks, promoting balanced performance and preventing bottlenecks. To maintain precise timing management, the controller includes latency counters that track and optimize access delays, ensuring smooth operation. Additionally, write-back policies are tailored specifically for neural operations, enhancing the system's ability to handle the demands of neural network computations effectively.

Matrix Multiplication Unit: The matrix multiplication unit implements a 16×16 systolic array architecture designed for optimal throughput and resource utilization.

Processing Element Design: Each processing element is equipped with a two-stage pipeline architecture, enabling efficient execution of instructions by overlapping stages of processing for improved throughput. It includes local accumulator arrays, which provide fast, on-chip storage for intermediate results, reducing the need for frequent memory access. The processing elements also feature input queues for data buffering, allowing smooth and continuous data flow to prevent stalls during operation. Additionally, the SIMD-compatible data paths ensure that the system can perform parallel data processing, making it well-suited for tasks that benefit from single instruction, multiple data operations.

Control Flow Management: The control system implements a distributed control architecture,

allowing for decentralized management that enhances scalability and reduces single-point failures across the system. It incorporates lightweight synchronization protocols, which streamline coordination between components while minimizing overhead, ensuring efficient operation. Advanced pipeline management is also a key feature, optimizing the flow of instructions and data through the processing stages for maximum throughput. Additionally, the system includes performance monitoring capabilities, enabling real-time tracking and analysis of operational efficiency to support ongoing optimization and troubleshooting.

Attention Module Implementation: The attention module implements 64 parallel attention heads, each capable of independent computation of attention patterns.

Attention Head Architecture: Each attention head contains dedicated Q/K/V computation units, which are specialized for calculating the query, key, and value components essential to attention mechanisms, ensuring precise and efficient processing. These heads also feature double-buffered computation units, allowing seamless switching between buffers to minimize downtime and enhance computational continuity. An optimized softmax implementation is included, tailored to accelerate the normalization process critical for attention-based operations. Furthermore, stream-aligned memory access patterns are utilized, aligning data retrieval with processing needs to maximize efficiency and reduce latency during memory interactions.

Parallel Processing Management: The module implements independent operation of attention heads, enabling each head to function autonomously, which enhances parallelism and overall system efficiency. It incorporates sophisticated synchronization mechanisms to coordinate activities across these heads seamlessly, ensuring consistent and reliable performance. Dynamic workload distribution is also a key feature, allowing the module to adaptively allocate tasks based on real-time processing demands, optimizing resource utilization. Additionally, advanced flow control is employed to manage data movement effectively, preventing bottlenecks and maintaining smooth operation throughout the module's processes.

4 RESULTS AND PERFORMANCE ANALYSIS

Test Environment: The performance evaluation was conducted under rigorous test conditions to thoroughly assess the system's capabilities. It utilized a hidden size of 8192, reflecting a substantial capacity for feature representation within the model. The evaluation included 64 attention heads, allowing for extensive parallel processing and attention coverage across the data. A batch size of 32 was employed, striking a balance between computational efficiency and memory usage. Additionally, a sequence length of 2048 was tested, simulating the system's ability to handle long-range dependencies and large-scale input data effectively.

Matrix Multiplication Performance: The matrix multiplication unit achieved exceptional performance metrics as shown in Table 1.

Table 1: Matrix multiplication unit performance.

Metric	Value	Status
Active Cycles	1010/1010	Optimal
Efficiency	100.00%	Optimal
Operations	2,58,560	High
Throughput	256.00 ops/cycle	Optimal
Bandwidth	1024 GB/s	Optimal

Attention Mechanism Performance: The attention module demonstrated impressive efficiency metrics as shown in Table 2.

Table 2: Attention unit performance.

Metric	Value	Status
Active Cycles	1009/1010	High
Efficiency	99.90%	High
Operations	8,072	Normal
Throughput	7.99 ops/cycle	Good

System-Level Performance: The complete system demonstrated several key performance characteristics that highlight its robustness and efficiency. It achieved sustained throughput across varying workloads, ensuring consistent performance regardless of task complexity or demand fluctuations. The system exhibited minimal pipeline stalls, reflecting a well-optimized design that keeps processing stages active and reduces idle time. Efficient memory bandwidth utilization was another standout feature, maximizing data transfer rates while minimizing waste, which is critical for high-performance operations. Additionally, effective thermal management ensured that the system

maintained optimal operating temperatures, enhancing reliability and longevity under intense computational loads.

4.1 Detailed Evaluation Methodology

Testing Framework: The evaluation framework was implemented in SystemC, utilizing a comprehensive test bench designed to provide detailed and reliable analysis of the system. It enables cycle-accurate simulation of all components, ensuring precise modeling of timing and behavior across the entire architecture. Performance counter integration at multiple levels allows for granular tracking of key metrics, offering deep insights into system efficiency and bottlenecks. The framework also supports configurable workload generation, providing flexibility to simulate a wide range of scenarios tailored to specific testing needs. Additionally, automated regression testing is incorporated, streamlining the validation process and ensuring consistent performance across updates or modifications.

Workload Generation: Test vectors were generated to align with LLaMA 3.1's computational patterns, ensuring the evaluation closely mirrored real-world usage of the model. These included matrix sizes matching the model's dimensions, specifically 8192×8192, to replicate the scale of its matrix operations accurately. Attention patterns were derived from real inference scenarios, capturing the system's behavior under practical attention-based workloads. Variable sequence lengths ranging from 128 to 2048 were tested, assessing the system's adaptability to both short and extended input sequences. Additionally, different batch sizes—1, 8, 16, and 32 were incorporated, allowing the evaluation to explore performance across a spectrum of processing demands.

Performance Metrics: During the evaluation, several critical metrics were collected to assess the system's performance comprehensively. Computational throughput, measured in operations per cycle (ops/cycle), provided insight into the system's processing speed and efficiency. Memory bandwidth utilization was tracked to evaluate how effectively the system leveraged available memory resources, a key factor in data-intensive tasks. Pipeline stall statistics were gathered to analyze interruptions in the processing flow, highlighting the system's ability to maintain continuous operation. Additionally, power efficiency estimates were calculated, offering a view into the system's energy consumption relative to its performance, an essential consideration for scalability and sustainability.

4.2 SystemC Implementation Details

The accelerator was implemented using SystemC 2.3.3, enabling cycle-accurate modeling of hardware behavior while maintaining high-level C++ abstraction.

Processing Element Design: The Processing Element serves as the fundamental computational unit within the systolic array, featuring a meticulously designed architecture to optimize performance. It implements a two-stage pipeline that enhances throughput by allowing overlapping execution of operations, paired with local storage in the form of an accumulator and an input queue for efficient data management and flow control. The operation of the Processing Element is governed by a structured algorithm: it initializes with class members including clock, reset, inputs a and b, and an output, alongside internal states like the accumulator, a busy flag, and the input queue. Upon initialization, the busy flag is set to false, and the compute and input-handling processes are registered to trigger on the clock's positive edge and input changes, respectively. The compute procedure, central to its functionality, resets the accumulator to zero and clears the busy state if a reset signal is detected; otherwise, when the input queue is not empty and the unit is not busy, it retrieves data from the queue's front, performs a multiply-accumulate operation (accumulator ← accumulator + data.a × data.b), removes the processed data from the queue, and writes the updated accumulator value to the output. This design ensures efficient, continuous processing with robust control over data flow and computational state.

Memory Bank Operation: The Memory Bank controller is engineered to manage access patterns and prefetch operations, ensuring optimal bandwidth utilization across the system. It orchestrates the interaction between the memory banks and the processing units by intelligently scheduling data retrieval and storage, minimizing latency and maximizing throughput. Through sophisticated prefetching, the controller anticipates data needs based on access patterns, proactively loading relevant data into buffers to reduce wait times and enhance overall efficiency. This design enables the Memory Bank to support high-performance operations, particularly in data-intensive tasks, by maintaining a steady and effective flow of information to and from the memory banks.

Attention Head Implementation: The Attention Head is designed to implement parallel processing of attention computations, incorporating optimizations for numerical stability to ensure both efficiency and accuracy. It handles the core attention mechanism by simultaneously computing query, key, and value operations across dedicated units, leveraging parallelism to accelerate processing speed. Numerical stability optimizations are integrated into the design, such as an enhanced softmax implementation, which mitigates issues like overflow or underflow during normalization, preserving the precision of attention scores even under large-scale or extreme data conditions. This combination of parallel execution and stability-focused enhancements enables the Attention Head to deliver robust and reliable performance in attention-based workloads.

Performance Monitoring Implementation: The Performance Monitoring System is designed to provide a detailed assessment of system performance through a structured set of metrics and procedures. It tracks primary metrics such as cycles, active cycles, stall cycles, and operations, while also calculating derived metrics including throughput, efficiency, and bandwidth. The Update Metrics procedure computes these derived values: throughput is determined as operations ÷ cycles, efficiency is calculated as (1 − stall cycles ÷ cycles) × 100 to express the percentage of productive time, and bandwidth is derived as bytes transferred ÷ (cycles × CYCLE_TIME_NS) × 10^{-9}, resulting in a gigabytes-per-second (GB/s) measurement. The GenerateReport procedure enhances this analysis by calling Update Metrics and logging a comprehensive performance report for the specified module. This report includes key statistics: total cycles, active cycles, stall cycles, operations, throughput (in ops/cycle), efficiency (as a percentage), and bandwidth (in GB/s). By systematically logging these metrics, the system ensures that performance data is both accurately captured and easily accessible for optimization and debugging purposes.

4.3 Implementation Path and Future Development

Hardware Implementation Considerations: The transition from SystemC simulation to hardware implementation presents several challenges, primarily in fabrication technology selection and design adaptation. Fabrication choices range from advanced nodes such as 3nm and 5nm from major foundries to more mature nodes like 28nm and 45nm for initial prototyping, with FPGA implementation serving as a viable option for validation. Additionally, adapting the design requires careful consideration of scaling for available process nodes, power and thermal management, physical design constraints, and rigorous testing and validation requirements.

To further enhance hardware performance, several improvements can be incorporated. These include the integration of advanced memory technologies, optimized implementations of attention mechanisms, sophisticated power management systems, and effective thermal optimization techniques. These enhancements contribute to better efficiency, performance, and scalability of the final hardware implementation.

Software Stack Development: The software infrastructure for the system requires a well-structured development approach, starting with the Driver Layer, which must ensure efficient data transfer mechanisms, robust command execution management, system monitoring capabilities, and effective error handling with recovery mechanisms. Additionally, Framework Integration is essential, involving seamless interfaces with PyTorch and TensorFlow, custom operator implementations, workload-specific optimizations, and performance monitoring tools. Looking ahead, future development will prioritize comprehensive driver enhancements, further optimizations for framework integration, advanced performance monitoring, and improved debugging and profiling capabilities to ensure a highly efficient and scalable software stack.

5 CONCLUSIONS

This study illustrates the effective design and modelling of a high-performance ASIC accelerator for neural network inference. The implementation demonstrates outstanding performance metrics, with the matrix multiplication unit operating at full efficiency at 256 operations per cycle and achieving 99.90% operational efficiency in the attention module.

The research has accomplished several notable accomplishments. The project encompasses the creation of a comprehensive SystemC model for an AI accelerator, allowing precise simulation and validation. A highly efficient 32-bank memory system has been developed to enhance data access and processing performance. The research includes the construction of a systolic array-based matrix multiplication unit, improving computing efficiency for AI tasks. Additionally, a parallel attention

mechanism with 64 heads has been implemented, enhancing the model's capacity to manage intricate attention-based calculations.

This study significantly advances the domain of AI hardware acceleration by investigating feasible alternatives to GPU-based inference, providing energy-efficient solutions designed for data centres, and laying a solid framework for future ASIC development. Furthermore, it tackles the von Neumann bottleneck in neural computing, facilitating the development of more efficient and scalable AI processing systems.

This study presents several intriguing directions for further investigation. A primary focus is the incorporation of developing AI model designs, which may significantly improve performance and efficiency. Furthermore, investigating sophisticated power optimization methods might result in more energy-efficient applications. Enhanced memory management techniques may enhance resource consumption, allowing improved scalability and accelerated calculations. Ultimately, optimizing attention mechanism implementations may improve model precision and processing efficiency, making AI systems more effective and versatile in handling intricate jobs. The tangible implementation of AI accelerators signifies a vital advancement toward enhanced and more accessible AI computing. This study lays the groundwork for future advancements, considering manufacturing limitations and the methodical creation of software infrastructure for effective neural network acceleration solutions.

REFERENCES

A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 5998-6008.

D. Markovi´c et al., "Memory architectures for deep learning hardware accelerators: A survey," IEEE Access, vol. 8, pp. 57566-57596, 2020.

H. Li et al., "An architecture-level analysis on deep learning models for low-impact computations," Artificial Intelligence Review, vol. 56, pp. 1971-2010, 2023.

M. Horowitz, "Computing's energy problem (and what we can do about it)," in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, 2014.

N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA), 2017.

S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in Int. Conf. Learning Representations, 2016.

V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proc. IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017.

W. J. Dally, "Hardware opportunities for machine learning," in Neural Information Processing Systems (NIPS) Workshop, 2017.

Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436- 444, 2015.

Y. Chen et al., "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA), 2016, pp. 367-379.