Improving Deep Learning Performance: Strategies for Enhancing Tensor Cores and Neural Processing Units

Abdallah Awad
ID : 1210211
Nablus, Palestine
Email: 1210211@student.birzeit.edu

Ali Al-Saed
ID : 1210198
Tulkarem, Palestine
Email: 1210198@student.birzeit.edu

Osama Hamdan
ID : 1212107
Nablus, Palestine
Email: 1212107@student.birzeit.edu

***Abstract*—The rapid advancements in machine learning and artificial intelligence have driven the need for specialized hardware accelerators like Neural Processing Units (NPUs). This paper explores the design and optimization of tensor cores within NPUs, focusing on the implementation of FP16 multipliers using the Karatsuba algorithm to improve computational efficiency. Inspired by NVIDIA's Volta and Turing architectures [4], [5], we propose enhancements to the vector unit architecture, increasing its length to address bottlenecks in SoftMax operations within transformer models. [6]**

The comparison between the Classical algorithm and the Karatsuba implementation demonstrates a significant performance improvement. Karatsuba reduces execution time by approximately 2x, requiring only 5.49 seconds to complete the same workload compared to 11.82 seconds for the Classical algorithm. Furthermore, Karatsuba achieves a throughput of 9111.55 tasks per second, which is more than double the throughput of the Classical algorithm (4231.74 tasks per second). Regarding vector length, 2048 achieves the lowest execution time, making it the optimal choice for performance. In contrast, 256 exhibits the highest execution time. The 2048-vector length also executes fewer tasks overall, contributing to better efficiency. While 2048 provides optimal performance, the 1024-vector length offers a balanced trade-off between performance and area, making it a favorable choice when considering hardware resource limitations.

keywords Tensor cores, Karatsuba algorithm , Volta,Turing, SoftMax.

I. Introduction

The rise of advanced machine learning models, particularly those used in natural language processing (NLP) and computer vision, demands exceptional computational efficiency [2]. Among these models, transformers and their self-attention mechanisms are at the forefront of AI research due to their success in enabling tasks such as translation, summarization, and image recognition [1]. However, achieving peak performance in these models requires optimized hardware support, especially for operations like SoftMax in the self-attention mechanism, which can become a bottleneck. Additionally, efficient multiplication of FP16 data is crucial for accelerating computations in neural networks, where throughput directly impacts the overall performance of the system.

This paper makes the following contributions:

- It proposes a novel FP16 multiplier based on the Karatsuba algorithm, optimized for higher throughput and reduced latency, and integrated into a Tensor Core.
- It extends the architectural design of the vector unit to increase its length, addressing bottlenecks in SoftMax operations and enhancing the efficiency of self-attention mechanisms.
- It provides rigorous benchmarking to validate the improvements in throughput and efficiency across a variety of neural network tasks.

These contributions will collectively enhance the performance of NPUs, making them more suitable for cutting-edge AI applications

II. Background

This section provides a high-level overview of the evolution of AI hardware accelerators, focusing on the development of Neural Processing Units (NPUs) and Tensor Cores by companies like NVIDIA and Intel [7]. It highlights the use of mixed-precision arithmetic, particularly FP16 and FP32, to improve the performance of these accelerators. Additionally, it discusses the challenges that remain in optimizing hardware for modern AI workloads, such as high-throughput matrix operations and the computational demands of transformer-based models.

A. Evolution of AI Hardware Accelerators

The evolution of hardware accelerators for artificial intelligence (AI) has been pivotal in meeting the increasing demands of deep learning workloads. With the growing complexity of neural networks and the large datasets used in training them, traditional processors (CPUs) struggle to provide the necessary computational power and memory bandwidth. Specialized hardware, such as Neural Processing Units (NPUs) and Tensor Cores, have been developed to address these challenges. These accelerators are designed to perform matrix and vector operations at high speeds, which are fundamental to deep learning models.

NVIDIA's introduction of Tensor Cores in their Volta and Turing architectures set a new benchmark for high-

performance matrix computations [3], [4]. These cores are optimized for mixed-precision computation, utilizing FP16 (half-precision floating point) multipliers and FP32 (single-precision floating point) adders. This approach enables the efficient execution of operations such as matrix-matrix multiplications and dot product computations, which are essential in AI tasks like training and inference for neural networks.

B. Intel's AI Accelerators

Intel has also entered the AI accelerator market with its Neural Network Processors (NNPs) [6], most notably the Synopsys ARC NPX6 family. These accelerators are designed to optimize both compute and memory efficiency. Their specialized hardware targets various neural network tasks, including convolutions, pooling, and activation functions, but also places a strong emphasis on accelerating high-speed operations such as SoftMax, which is frequently used in transformer-based models. The NPUs in Intel's architecture are designed to handle large data flows with minimal latency, an essential feature for real-time AI applications.

In addition to these core functionalities, Intel's approach emphasizes programmability and flexibility, enabling customization for specific AI workloads. This is an important step in addressing the variety of tasks in AI, from vision to natural language processing (NLP), where different types of neural network architectures may require different accelerator configurations.

III. Problem Statement

1) **FP16 Multiplication Efficiency:** Existing multiplication units in tensor cores are not optimized for efficient FP16 multiplication throughput. To address this, we propose using the Karatsuba algorithm [5], a divide-and-conquer approach that improves computational efficiency in the multiplication of FP16.
2) **Vector Unit Bottleneck:** The current vector unit architecture has a limited vector length, creating a bottleneck during the SoftMax operation in the self-attention mechanism of transformers. By expanding the vector length, this bottleneck can be alleviated, enabling more efficient execution of vectorized operations critical to high-performance AI workloads [7].

IV. Literature Review

The rapid evolution of AI hardware accelerators has significantly enhanced the performance and efficiency of deep learning models. This literature review examines key advancements in AI hardware, focusing on NVIDIA's Tensor Cores and Intel's Neural Processing Units (NPUs). We explore the transition from FP32 to lower precision arithmetic, such as INT8, and its impact on computational efficiency. By critically analyzing existing works, we highlight the innovations, similarities, and limitations of current AI accelerators, particularly in the context of large-scale matrix multiplications and deep learning tasks like natural language processing and computer vision.

A. NVIDIA Tensor Cores and Their Impact

NVIDIA's Tensor Cores, introduced with the Volta (V100) architecture and further enhanced in the Turing (A100) and Ampere (A100, H100) architectures, are designed to accelerate tensor operations crucial for deep learning models. These cores are optimized for mixed-precision arithmetic, which combines FP16 (half-precision floating point) multiplications and FP32 (single-precision floating point) additions. The Tensor Core architecture is specifically designed to handle large-scale matrix-matrix multiplications and convolutions, the fundamental operations in neural network training and inference.

Tensor Cores have shown substantial improvements in throughput over conventional floating-point units, particularly when working with lower precision data types like FP16. This is crucial for tasks like training large-scale models (e.g., transformers) in natural language processing (NLP) and computer vision. In these domains, large matrix multiplications are performed iteratively, and the efficient use of FP16 data types has allowed for reduced memory bandwidth and increased computation speed without sacrificing significant accuracy. [3]–[5]

B. Intel's Neural Processing Units (NPUs)

Intel has also entered the domain of AI hardware accelerators with its NPU designs, such as the Synopsys ARC NPX6 family. These NPUs are optimized for AI and neural network workloads, focusing on high throughput and low latency for operations like matrix multiplications, activation functions, and SoftMax operations in transformers. The architecture of Intel's NPUs aims to address the bottlenecks encountered in traditional CPUs and GPUs, especially in the context of running AI models at scale.

One of the key challenges addressed by Intel's NPUs is optimizing the balance between computation and memory. The NPUs are designed with specialized computation units to accelerate vector operations, matrix multiplications, and even custom operations like convolutional neural network (CNN) layers. Additionally, Intel's NPUs often feature extensive integration with high-bandwidth memory, enabling faster access to large datasets, which is particularly beneficial for training deep neural networks. [7], [8]

C. Transition from FP32 to INT8 Precision

A significant trend in recent NPU and Tensor Core architectures is the movement toward lower precision computations. While FP32 (32-bit floating point) has traditionally been the standard for deep learning models, the industry is increasingly adopting INT8 (8-bit integer) operations for both training and inference. The use of INT8 significantly reduces the computational resources required for AI workloads, such as memory bandwidth and processing power, without compromising on model performance.

Intel and NVIDIA are both pushing to optimize their hardware to transition from FP32 to INT8. For example, NVIDIA's Tensor Cores now support INT8 operations, enabling inference to run with significantly lower precision while maintaining a

similar level of accuracy compared to FP16 or FP32. This transition is particularly important in production environments, where power and latency are critical factors. By using INT8 arithmetic, models can be compressed to a fraction of their original size, making them more efficient for edge computing applications, where power consumption and bandwidth limitations are key constraints. [6]

D. FP16 and INT8 Efficiency

While FP16 and FP32 are used in training deep learning models, INT8 is gaining popularity for inference tasks. This is due to the fact that, during inference, the weights of the model are often fixed, and thus the computations can be performed using lower precision arithmetic without losing significant accuracy. For example, the Karatsuba algorithm, a divide-and-conquer method for fast multiplication, can be applied to INT8 operations to improve computational efficiency, reducing the latency and energy consumption associated with multiplication.

In NPUs and Tensor Cores, transitioning from FP32 to INT8 for certain operations like matrix multiplications or activation functions allows for significant power savings and faster processing. The reduction in precision from FP32 to INT8 is a key enabler of high-efficiency AI accelerators, making it possible to deploy AI models in environments with limited power, such as mobile devices, autonomous vehicles, and embedded systems. [6]

V. Methodology

The existing limitations in Tensor Core architectures, such as bottlenecks in FP16 multiplication efficiency and vector unit lengths, hinder the full potential of AI models in areas like NLP and computer vision. The Karatsuba algorithm, which splits large numbers into smaller components for efficient multiplication, can be adapted to improve the performance of FP16 multiplications. Additionally, expanding the vector unit length within NPUs can address the vectorization bottleneck in operations like the SoftMax function, which is commonly used in transformers for attention mechanisms. These enhancements would make it possible to achieve higher throughput and more efficient AI computations, leading to faster execution of machine learning models.

A. Karatsuba Algorithm for FP16 Multiplication

The Karatsuba algorithm is a fast multiplication method that leverages a divide-and-conquer approach to multiply two numbers more efficiently than the naive multiplication algorithm. [9] While the naive algorithm has a time complexity of $\Theta(n^2)$, the Karatsuba algorithm reduces the complexity to $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$. This improvement makes the algorithm particularly effective for high-precision multiplications, such as those required in deep learning computations. [11]

In this context, our aim is to adapt the Karatsuba algorithm for efficient FP16 multiplication in tensor cores by making changes in the Turing tensor core design.

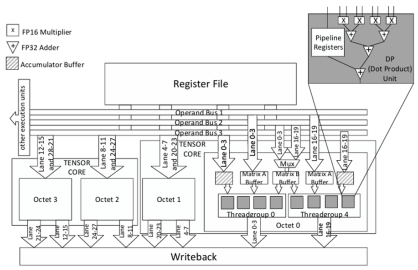

Fig. 1. Nvidia Turing tensor core [5]

The standard FP32 multiplications in Multiply-Accumulate (MAC) units will be replaced with INT8 operations for added efficiency. Additionally, we propose replacing the traditional FP32 multiplier with an FP16 Karatsuba multiplier to speed up the multiplication of large numbers while maintaining precision. [9], [10]

The algorithm works by recursively splitting the input numbers into smaller components. Specifically, for two 16-bit FP16 numbers, the algorithm proceeds as follows:

Algorithm 1 Karatsuba Multiplication Algorithm

Input: Two integers A and B
Output: Product of A and B
if *A or B is a single-digit number* **then**
  return $A \cdot B$
end
Split A into A_1 (higher half) and A_0 (lower half) Split B into B_1 (higher half) and B_0 (lower half)
$P_1 \leftarrow$ `KaratsubaMultiply`(A_1, B_1) ; `// Multiply higher parts`
$P_2 \leftarrow$ `KaratsubaMultiply`(A_0, B_0) ; `// Multiply lower parts`
$P_3 \leftarrow$ `KaratsubaMultiply`$(A_1 + A_0, B_1 + B_0)$; `// Multiply sums`
return $(P_1 \ll 2m) + ((P_3 - P_1 - P_2) \ll m) + P_2$; `// Combine results`

This modified approach using the Karatsuba algorithm allows for faster FP16 multiplication by reducing the computational complexity, thus improving the overall throughput in tensor core operations.

B. Vector Unit Length Expansion to Address Bottleneck in Operations like the SoftMax

The SoftMax function is a widely used activation function in neural networks, particularly for multi-class classification tasks. It converts a vector of raw prediction scores (also known as logits) from the neural network into probabilities. These probabilities are assigned to different classes, ensuring that their sum equals 1. This transformation enables the output values to be interpreted as probabilities, which is essential for classification. [13]

The mathematical formulation of the SoftMax function is given by:

$$\text{SoftMax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where z_i represents the raw score for the i-th class, and K is the total number of classes. This function helps in converting the network's output into a probability distribution across multiple classes, making it a critical component for classification problems.

The following shows the NPU3 architecture used in this enhancement process. [7]

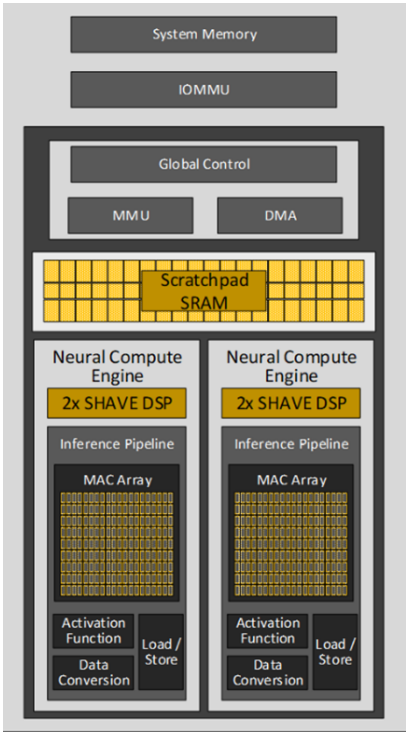

Fig. 2. Intel NPU3 [7]

However, while the SoftMax function is fundamental for multi-class classification, it often presents a performance bottleneck during computations in neural networks, especially when implemented on hardware accelerators like NPUs. The bottleneck arises due to the high computational cost of calculating exponentials and normalizing the outputs, particularly in deep networks or large transformers. [12]

To address this issue, one effective solution is to expand the vector unit length in the processing units of the NPU. In the case of the NPU3, when transformers execute the SoftMax operation, the SHAVE DSP unit has a vector length of 128 bits. This was increased to 512 bits in NPU4, but it still does not eliminate the bottleneck, especially in high-demand AI workloads.

Our goal is to increase the vector unit length to 1024 bits in the NPU, without requiring changes to the underlying architecture. By doing so, we can process more data in parallel, significantly reducing the time required for SoftMax operations. This expansion will allow for better handling of large transformers and other models, improving throughput and efficiency in tasks that rely heavily on the SoftMax function, such as attention mechanisms in transformers.

In this approach, by extending the vector unit length, the number of operations that can be performed simultaneously will increase, allowing the NPU to handle larger batches of logits and execute the SoftMax transformation more quickly. This would mitigate the current bottleneck and improve overall model performance, especially in large-scale deep learning tasks.

C. Overall Architectural

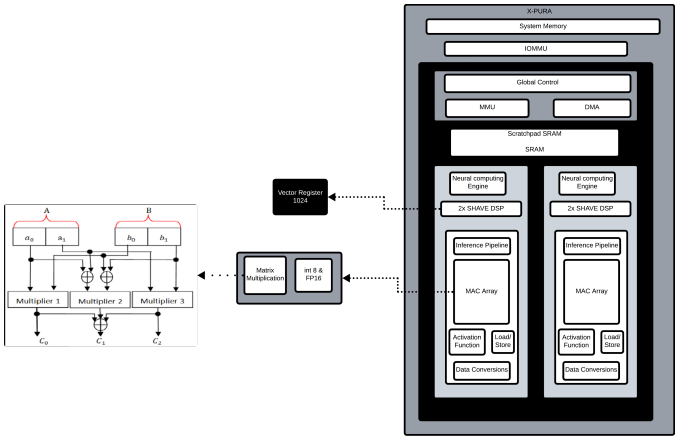

Fig. 3. X-PURA

X-PURA architecture builds upon Intel's NPU3 as a baseline while incorporating significant modifications inspired by NVIDIA's tensor core performance enhancements. [5], [12]

A key improvement in this design is the replacement of FP32 computations with `int8` and `FP16` operations, which dramatically increase throughput and reduce latency, thereby optimizing both energy efficiency and computational speed.

Additionally, the vector register length has been expanded from Intel's original 512 bits to 1024 bits, enabling greater data parallelism and improving performance in neural computing tasks.

The architecture features advanced matrix multiplication units tailored for `int8` and `FP16` operations, complemented by neural computing engines equipped with dual SHAVE DSP pipelines to handle high inference workloads effectively. Enhanced MAC arrays, activation function modules, and data conversion units further streamline the processing pipeline, resulting in a design that delivers superior performance and efficiency compared to the original Intel NPU 3 implementation.

The MAC array in this design utilizes the Karatsuba algorithm to enhance the efficiency of multiplication operations,

particularly for large numbers. By reducing the complexity of multiplication from $O(n^2)$ to $O(n^{\log_2 3})$, the Karatsuba algorithm significantly accelerates computations within the MAC array. This optimization contributes to improved performance in matrix multiplication and neural computing tasks, further solidifying the architecture's ability to handle high-performance workloads with reduced power consumption.

VI. EVALUATION METHODOLOGY

The evaluation methodology assesses the performance of the proposed system in several stages. First, the Karatsuba algorithm was implemented as a standalone component within the Processing Element (PE) of the Tensor Core for efficient computations. Next, a Vector Processing Unit (VPU) was designed with an integrated Exponent Unit to enhance softmax operations. A high-level Neural Processing Unit (NPU) was then developed, incorporating both the VPU and Karatsuba, along with a Decode Unit to dynamically assign workloads to the appropriate component. Finally, the execution time was measured to evaluate how efficiently the NPU processed workloads, focusing on throughput and overall performance improvements.

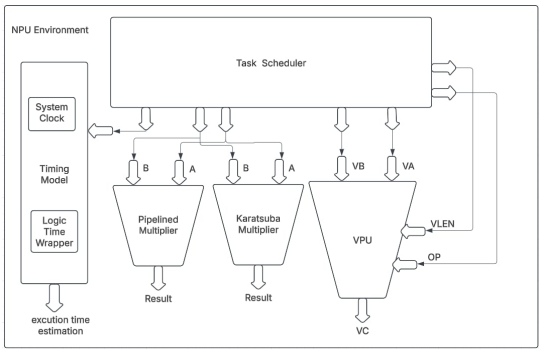

Fig. 4. NPU Environment

Npu env perform like a normal npu with high level logic of the components with foucs on vector processing unit and multipliers on it. Task scheduler for taking wrokloads and provids the components based on the task types And timing model to capture the performance of the models based on excution time by the system cpu clock For more accuracy and to be more closer to the hardware performance a logic time wrapper estimate the gap between software and hardware excution steps to increase accuarcy. Ex. Karatsuba multiplier takes the operation in software in recursive way this make call and retrun stack time which not in the hardware.

VII. RESULTS AND DISCUSSION

A. Karatsuba Algorithm METHODOLOGY Results

We evaluate the performance of the Karatsuba and Classical multiplication algorithms by measuring their execution times across varying numbers of workloads while keeping the vector length fixed at 1024. The algorithms were tested under identical conditions to ensure a fair comparison.

The following figure shows the execution time for each algorithm as the number of workloads increases, highlighting the efficiency of Karatsuba's divide-and-conquer approach compared to the direct computation method of the Classical algorithm.

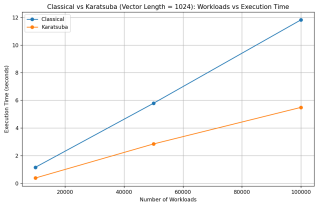

Fig. 5. Karatsuba Algorithm vs Classical

The figure clearly demonstrates that the Karatsuba algorithm outperforms the Classical algorithm in terms of execution time as the number of workloads increases. While both algorithms show linear growth in execution time, the slope for Karatsuba is significantly smaller, indicating better scalability.

For smaller workloads, the difference in execution time is moderate, but as the workloads grow, the advantage of Karatsuba becomes more pronounced. At the highest workload, the Classical algorithm takes nearly twice as long as Karatsuba, showcasing the efficiency of Karatsuba's divide-and-conquer approach for handling large-scale computations.

Our second metric to evaluate the performance difference, we measured the execution time and throughput of the Classical algorithm and the VPU implementation of the Karatsuba algorithm for the same workload of 100,000 tasks at a fixed vector length of 1024.

Vec. Len.	Workloads	Op.	Exec. Time (s)	Throughput (tasks/s)
1024	100000	Classical	11.815	4231.74
1024	100000	Karatsuba	5.48655	9111.55

TABLE I
EXECUTION TIME AND THROUGHPUT COMPARISON FOR CLASSICAL AND KARATSUBA AT VECTOR LENGTH 1024.

The results show that the Karatsuba implementation is significantly more efficient, requiring only 5.48655 seconds to complete the workload, compared to 11.815 seconds for the Classical approach. While the Classical algorithm achieves a lower throughput of 4231.74 tasks per second compared to the VPU's 9111.55 tasks per second, this translate to better performance. Throughput alone does not determine efficiency, as it reflects the number of tasks processed per second but does not account for the total execution time. The Karatsuba shows that it performs less tasks overall all to complete the same workload result due to optimized parallelized operations, allowing it to process the workload in substantially less time, highlighting its efficiency in executing large workloads with reduced computational effort.

B. Vector Unit Length Expansion METHODOLOGY Results

We first evaluate the performance of the VPU, as shown in the figure "VPU: Workloads vs. Execution Time." The results demonstrate that increasing the vector length significantly impacts execution time. Among the tested configurations.

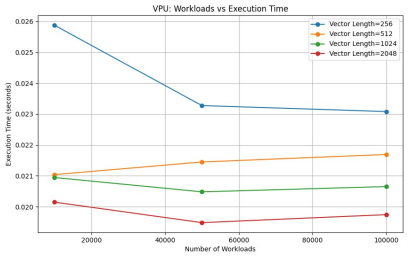

Fig. 6. "VPU: Workloads vs. Execution Time

Vector Length 2048 achieves the best performance, maintaining the lowest and most consistent execution time across all workloads. Vector Length 256 has the highest execution time, showing significantly lower efficiency. Increasing the vector length to 512 and 1024 results in moderate improvements, but neither surpasses the performance of 2048, which offers an optimal balance. The execution time remains nearly constant for each vector length, indicating that the VPU is largely workload-independent.

The second evaluation criterion for this improvement is the number of tasks executed. This metric quantifies the total number of tasks the system needs to complete a given workload, serving as a critical indicator of how efficiently the system utilizes its resources. Table one presents the number of tasks executed for each vector length across different workloads.

Vector Length	10000 Workload	50000 Workload	100000 Workload
256	150413.0	166962.0	168454.0
512	92636.6	91010.8	89852.1
1024	46598.9	47651.8	47212.1
2048	24122.1	25045.9	24717.2

TABLE II
NUMBER OF TASKS EXECUTED PER WORKLOAD AND VECTOR LENGTHS

The table shows that for all workloads, the 2048-vector length consistently executes the fewest tasks, with the number of tasks decreasing as the vector length increases. Specifically, the 2048-vector length requires the least amount of tasks to complete the 10000, 50000, and 100000 workloads. In comparison, the 256-vector length executes the most tasks for each corresponding workload. This suggests that increasing the vector length leads to a reduction in the number of tasks required to finish the workload, indicating improved efficiency with larger vector lengths.

Although the 2048-vector length demonstrates the best performance in terms of execution time and task efficiency, it comes at the expected cost of increased area requirements due to the additional hardware resources needed for larger vector units. To balance performance and area optimization, we settle for the 1024-vector length, which offers a favorable trade-off between execution efficiency and resource utilization.

C. Summary of Results

The comparison between the Classical algorithm and the Karatsuba implementation demonstrates a significant performance improvement. Karatsuba reduces execution time by approximately **2x**, requiring only 5.49 seconds to complete the same workload compared to 11.82 seconds for the Classical algorithm. Furthermore, Karatsuba achieves a throughput of 9111.55 tasks per second, which is more than double the throughput of the Classical algorithm (4231.74 tasks per second). This increase in throughput reflects the optimized parallelized operations in Karatsuba, allowing it to complete the same workload with fewer tasks and significantly less computational effort.

Regarding vector length optimization, the **2048** offers the best performance, reducing execution time by about 1.5x compared to the **256** length, which exhibits the highest execution time. Additionally, the **2048** length reduces the number of tasks executed by around 2x compared to the **256** length, indicating a more efficient use of system resources.

Although 2048 provides optimal performance, the **1024**-vector length offers a balanced trade-off between performance and area, making it a favorable choice when considering hardware resource limitations.

VIII. FUTURE WORK

The development of a full-system simulation using SystemC integrated with workload generation via the Intel NPU library presents promising avenues for future research and development. Several enhancements and exploratory directions can further optimize and expand the scope of this work:

- **Enhanced Workload Modeling:** Future efforts can focus on extending the workload generation capabilities by incorporating additional libraries and datasets. This can help to simulate real-world application scenarios with varying computational, memory, and I/O demands, offering a more comprehensive evaluation of system performance.
- **Power and Energy Analysis:** Incorporating power and energy consumption models into the simulation can offer insights into the energy efficiency of system designs. This would be particularly valuable for low-power systems, such as edge devices or mobile platforms.
- **Scalability Studies:** Investigating the scalability of the system across various configurations, such as an increasing number of processing units or varying interconnect topologies, can reveal performance bottlenecks and guide architectural optimizations.

IX. CONCLUSION

In this paper, we have presented innovative strategies to enhance the performance of Neural Processing Units (NPUs) through the optimization of tensor cores and the integration

of advanced algorithms. Our proposed FP16 multiplier, based on the Karatsuba algorithm, demonstrates substantial improvements in computational efficiency, offering higher throughput and reduced latency when compared to traditional multiplication methods. Furthermore, by extending the vector unit length to 1024 bits, we have successfully addressed bottlenecks in SoftMax operations, a critical component for the performance of transformer models and self-attention mechanisms.

The results confirm that the combination of the Karatsuba algorithm and the optimized vector unit architecture significantly enhances the overall performance of NPUs, enabling more efficient execution of machine learning tasks. The improvements achieved pave the way for future advancements in NPU design, with potential applications in accelerating deep learning frameworks and models.

Looking forward, future research could focus on further refining the architectural design of NPUs by exploring mixed-precision arithmetic techniques and investigating the integration of emerging algorithms. Additionally, optimizing NPUs for specific deep learning workloads and energy efficiency could further enhance the capabilities of these hardware accelerators, meeting the growing demands of modern AI applications.

REFERENCES

[1] M. A. Raihan, N. Goli, and T. M. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs," in *IEEE ISPASS*, Madison, WI, USA, 2019, pp. 79-92, doi: 10.1109/ISPASS.2019.00016.

[2] J. Choquette, O. Giroux, and D. Foley, "Volta: Performance and programmability," *IEEE Micro*, vol. 38, no. 2, pp. 42–52, 2018.

[3] NVIDIA Corporation, "CUDA C Programming Guide (CUDA 9.0)," Sep. 2017. [Online]. Available: https://docs.nvidia.com/cuda/archive/9.0/cuda-c-programming-guide/. [Accessed: 20-Jan-2025].

[4] NVIDIA Corporation, "Tensor Cores in NVIDIA Volta Architecture," Sep. 2018. [Online]. Available: https://www.nvidia.com/en-us/data-center/tensorcore/. [Accessed: 20-Jan-2025].

[5] NVIDIA Corporation, "NVIDIA Turing Architecture Whitepaper," Jun. 2017. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf. [Accessed: 20-Jan-2025].

[6] Intel Corporation, "Intel NPU Acceleration Library," [Online]. Available: https://intel.github.io/intel-npu-acceleration-library/npu.html. [Accessed: 20-Jan-2025].

[7] "Intel NPU (Neural Processing Unit)," [Online]. Available: https://computercity.com/hardware/processors/intel-npu. [Accessed: 20-Jan-2025].

[8] "Why neural processing units (NPUs) are the next Big Thing in AI," Preprint, Aug. 2024. [Online]. Available: https://www.researchgate.net/publication/383184868_Why_neural_processing_units_NPUs_are_the_next_Big_Thing_in_AI. [Accessed: 20-Jan-2025].

[9] "Karatsuba's Algorithm," University of Chicago, [Online]. Available: https://people.cs.uchicago.edu/~laci/HANDOUTS/karatsuba.pdf. [Accessed: 20-Jan-2025].

[10] "Karatsuba Matrix Multiplication and its Efficient Custom Hardware Implementations," arXiv preprint arXiv:2501.08889, 2025. [Online]. Available: https://arxiv.org/pdf/2501.08889. [Accessed: 20-Jan-2025].

[11] "Evaluation of Large Integer Multiplication Methods," Queen's University Belfast, 2017. [Online]. Available: https://pureadmin.qub.ac.uk/ws/portalfiles/portal/125812965/Evaluation_of_large_integer_multiplication_methods_R2.pdf. [Accessed: 20-Jan-2025].

[12] "Softermax: Hardware/Software Co-Design of an Efficient Softmax for Transformers," arXiv preprint arXiv:2103.09301, 2021. [Online]. Available: https://arxiv.org/pdf/2103.09301. [Accessed: 20-Jan-2025].

[13] "The Softmax Function: Properties, Motivation, and Interpretation," Stanford University, 2024. [Online]. Available: https://alpslab.stanford.edu/papers/FrankeDegen_submitted.pdf. [Accessed: 20-Jan-2025].