2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS) | 979-8-3503-2649-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICECS58634.2023.10382797

Optimizing Data Availability and Utilization in Deep Learning Accelerator SoCs

Cagla Irmak RUMELILI KOKSAL[1,2] Nihat Mert CICEK[1,2] Ayse YILMAZER METIN[2] Berna ORS[2]
[1]*Aselsan Inc., Ankara, Turkiye*
irumelili,nmcicek@aselsan.com.tr
[2]*Istanbul Technical University, Istanbul, Türkiye*
yilmazerayse,orssi@itu.edu.tr

***Abstract*—Deep learning accelerators are pivotal in accelerating computation-intensive tasks in modern AI applications. Optimizing the utilization of system resources, including shared cache, on-chip SRAM, and data movement mechanisms, is crucial for achieving superior performance and energy efficiency. In this study, we propose an efficient system architecture specifically tailored for deep learning workloads. Our architecture enables to reconfigure the last level cache as a scratchpad with prefetch capability, which eliminates cache misses and thereby offers resource efficiency, improved performance, and energy efficiency. By implementing a strategy to overlap accelerator execution with data movement, we achieved remarkable results, including a 14x speedup and %5 reduction in energy consumption for the ResNet50 benchmark when compared to the base system configuration. These findings demonstrate the substantial benefits of incorporating prefetch support and scratchpad reconfiguration in the last level cache, leading to enhanced performance and energy efficiency in real-world deep learning accelerator applications.**

***Index Terms*—deep learning accelerators, cache hierarchy, cache configuration, data availability, accelerator utilization, latency, energy efficiency, cache design, prefetching mechanism.**

I. Introduction

Deep learning accelerators are specialized hardware designed to accelerate the computationally intensive tasks of deep learning algorithms. They outperform CPUs and GPUs in terms of performance per watt for specific deep learning workloads [6], [3]. With optimized hardware components and algorithms, deep learning accelerators deliver faster and more efficient processing, making them ideal for AI applications requiring real-time decision-making and large-scale neural networks.

Deep learning accelerators can be categorized into standalone accelerators [3], which possess their own dedicated memory hierarchy and computation units, and accelerators that share resources within a system on chip (SoC) environment [2]. In the latter case, where resources are shared, it is essential to consider the entire system to achieve high energy efficiency and throughput. Resource sharing introduces challenges, particularly in the areas of memory hierarchy and data movement. Efficient management of the memory hierarchy is crucial to minimize latency and optimize data access. Additionally, effective data movement mechanisms are vital to facilitate seamless communication between the accelerator and other components. These considerations are of utmost importance in shared-resource accelerators, as they directly impact system performance and efficiency.

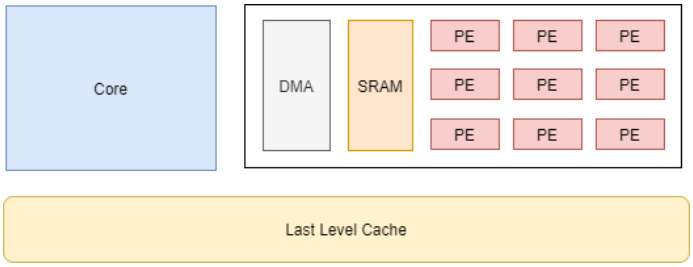

Fig. 1. Base architecture for a deep learning accelerator with SRAM, DMA, and execution units.

In this paper, we propose a system solution for a deep learning workload. In this system, we add prefetch capability to the last level cache (LLC). In addition to that, dedicated portion of the LLC can be reconfigured as scratchpad. This way, we aim to reduce cache misses and improve overall performance and energy efficiency of the system. Furthermore, we propose a strategy to overlap accelerator execution with data movement. Our contributions can be summarized as follows:

- Reconfiguration of the system's LLC as scratchpad with prefetch capability provides us resource efficiency along with improvement in performance and energy efficiency.
- We propose a strategy to overlap accelerator execution with data movement, which maximizes utilization in the accelerator.
- We obtain 14x speedup and %5 reduction in energy consumption for ResNet50 benchmark compared the base system described in Section II.

The remaining sections of the paper discuss the overview of the base system, proposed architecture, experimental results, related work and conclusion, respectively.

II. Overview

A typical deep learning accelerator system is depicted in Figure 1. The base architecture consists of a shared LLC that is accessible by both the core and the accelerator. Additionally, the accelerator incorporates its own on-chip SRAM, which, although smaller, offers faster access times compared to the

LLC. Not only accessing the cache require more power, but it is also slower compared to accessing data from the SRAM. Therefore, there is a strong emphasis on reusing data stored in the SRAM to minimize the number of requests to lower-level memory. This approach aims to optimize data movement and reduce both power consumption and latency in the system.

To facilitate efficient data movement between the cache and the accelerator's SRAM, a Direct Memory Access (DMA) mechanism is employed. This DMA mechanism enables direct transfers of data between the cache and SRAM without involving the CPU, minimizing latency for the accelerator.

One of the key challenges in this system architecture is the minimization of cache misses. Cache misses happen when the requested data is not found in the cache and needs to be fetched from a lower-level memory, such as the main memory or external DRAM.

In the context of a shared set-associative cache used by both cores and accelerators, cache misses can occur due to several reasons. One common cause is the conflict between the memory access patterns of the cores and the accelerator. If the cores and accelerator frequently access data that maps to the same cache set, it can lead to increased cache conflicts and subsequent cache misses. Another reason for cache misses is the limited cache capacity. When the cache becomes saturated and cannot accommodate all the required data, cache evictions occur, resulting in cache misses for subsequent memory accesses. Additionally, cache misses can also be caused by data dependencies and irregular memory access patterns inherent in certain applications or algorithms.

Cache misses can significantly impact system performance as they introduce additional latency due to the time required to fetch the data from the slower memory hierarchy levels. These cache misses result in stalls or delays in the execution of the accelerator, reducing the overall efficiency and throughput of the system. Minimizing cache misses is therefore crucial for improving system performance and ensuring efficient utilization of the available resources.

III. Proposed Architecture

To minimize cache misses, we suggest implementing a software prefetch mechanism [12] alongside a scratchpad memory. We offer to use the configurable amount of the LLC as scratchpad. The Prefetch unit retrieves data from the main memory and writes it into the scratchpad region, effectively eliminating cache misses. Prefetch mechanism adopts a strategy of bringing in larger chunks of data from the main memory compared to what the DMA transfers, optimizing data availability for the accelerator. Software determines which memory segments to prefetch into the scratchpad to enhance performance. While the execution for the current chunk continues, we simultaneously prefetch the next chunk from the main memory, ensuring a continuous flow of data without any interruptions. Furthermore, the eviction of the data that will be used in the near future is eliminated.

We are examining an accelerator comprising 32x32 processing element (PE) units. For each row of the PE array, there is a dedicated SRAM in a multi-bank fashion. DMA is used to bring filters and inputs from the lower memory, and it writes into the SRAM in small chunks. Chunk size is programmed by software according to the layer size and available space in SRAM. In order to maximize utilization in the accelerator, we adopt decoupled access execute strategy. For this, when DMA finishes its operation for the selected chunk size, accelerator starts immediately. During execution, DMA continues to bring next chunk and fills the SRAM in a circular manner.

32 channels of a filter is placed into a PE row while each row has a different filter. We adopt a weight-stationary dataflow where each input is broadcasted into each row. As we obtain partial results, we store them in a accumulator SRAM. When the whole convolution is calculated, it is written back to memory.

Half of the LLC is configured as scratchpad. Our prefetch unit is programmed by software to bring larger chunks than DMA so that we can parallelize accelerator execution, DMA, and prefetch operations. This approach provides us great improvements in execution time since we avoid cache misses thanks to prefetch unit and scratchpad configuration. In addition to that, energy consumption is minimized by optimizing data movement between different memory levels.

IV. Evaluation

A. Experimental Setup

In our evaluation, we employ convolutional layers based on the ResNet50 [7] benchmark. We perform output calculations on smaller matrices denoted as $OXOXPErow$. To ensure efficiency, we set an upper limit for O to avoid exceeding the capacity of the output buffer. The number of PE rows dictates the number of outputs that can be computed simultaneously. We further divide both the kernel and input into smaller units, referred to as input partials (Ip) and weight partials (Wp), respectively. The structure of the weight partials depends on the kernel size (K) and channel size (Cin), with the number of kernels ($Cout$) restricted to match the number of PE rows (1). The size of the input partials is determined by the values of O, the stride, padding and Cin of the input [5]. We define the ratios of the total input size to Ip and weight size to Wp as Ic and Wc, respectively. The number of Ip's that fits into SRAM are defined as Ipc.

The execution process initiates when at least one input partial and one weight partial are loaded into the accelerator's SRAM. The execution cycles are calculated as described in (2). Subsequently, the transfer of weight and input chunks occurs concurrently with the execution cycles. However, the accelerator cannot execute under the following conditions:

- When either the input partial or weight partial is not available in the SRAM.
- When weight or input data is being broadcast to PEs.
- When output data is prepared and being written to the LLC

These waiting periods are collectively referred to as wait cycles. The performance metric is determined by the sum of execution cycles and wait cycles.

To quantify dynamic energy consumption resulting from data movement, we calculate the sizes of data read from and written to the accelerator's SRAM and LLC, and then divide them by the bus data width, which is 128 bits per accelerator cycle. The total weight read from SRAM, input read from SRAM, input/weight read from LLC, and output written to LLC are quantified using equations (3), (4), (5), and (7), respectively.

$$Wp = PErow * K * K * Cin \tag{1}$$

$$ExeCycle = OxOxCout/PEcol * K * K * Wc * Ic \tag{2}$$

$$SramWR = Cout * Cin * K * K * Ic \tag{3}$$

$$SramIR = Ip * Ic * K * K * Wc \tag{4}$$

$$LLCRead = (Wp * Wc + Ip * iloop) \tag{5}$$

$$iloop = \begin{cases} Ic & \text{if Ipc} > \text{Ip*Ic + Wp} \\ Ic + (Wc - 1) * (Ic - Ipc) & \text{otherwise} \end{cases} \tag{6}$$

$$LLCWrite = O * O * Cout * Ic \tag{7}$$

For energy consumption analysis, we consider that DRAM accesses consume 1.8 nJ, while accessing the LLC requires 150 pJ and accessing the accelerator SRAM consumes 60 pJ [9].

For evaluation, we compare the following schemes:

- **Case 1**: We assume an ideal scenario where both the SRAM and cache have infinite capacity. To ensure data availability, we access the DRAM to bring the required data into the system cache while the DMA concurrently fetches data from the cache. Given the infinite SRAM capacity, data availability for the accelerator is guaranteed, eliminating any potential bottlenecks.
- **Case 2**: We consider a scenario where the SRAM has a finite capacity, while the cache remains ideal. The proposed DMA engine works in a circular manner to provide data to SRAM and manage data availability for the accelerator.
- **Case 3**: We examine the scenario, described in Section II, where both the SRAM and cache have finite capacities. When accessing the system cache, cache misses occur due to the limited cache size. In our setup, we consider the cache block as 64KB and cache misses take 350 cycles on average [4].
- **Case 4**: Our proposed solution, as explained in Section III, involves the integration of a prefetch feature into the LLC and the flexible use of cache as loosely integrated memory to eliminate cache misses experienced in case 3. The size of the scratchpad is chosen as 1.5 megabytes. The initial input is transferred from memory to the scratchpad. The calculated outputs, which serve as the input for the next layer, overwrite the input values since they are no longer required for further calculations. The remaining space in the scratchpad is allocated for storing weight chunks. When a weight chunk is no longer in use another one is fetched from SRAM.

Note that case 1 and case 2 are idealized scenarios to observe the theoretical bandwidth, while case 3 and case 4 represent a more practical and realistic setup.

B. Results and Discussion

Figure 2 shows the performance and energy consumption in terms of accelerator cycles and uj, respectively. Our proposed solution (case 4), using prefetch mechanism in a shared system cache for deep learning workloads, gives much faster execution and lower energy consumption. Specifically, our proposed architecture is 14X times faster than the realistic scenario described in case 3 and 5% times more energy efficient. Besides, overall performance is too close to the almost ideal scenario described in case 2.

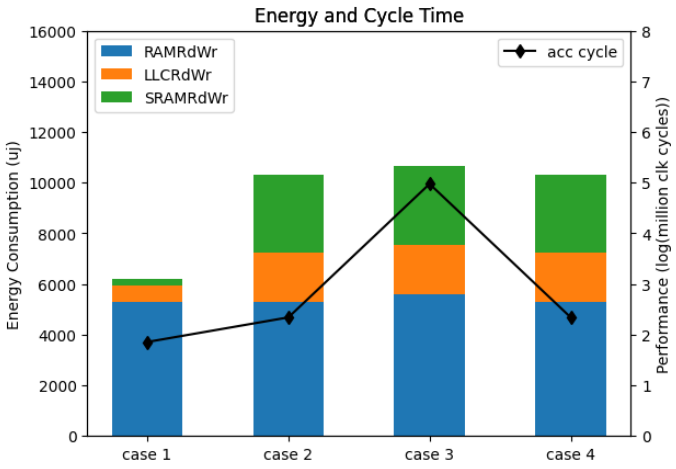

Fig. 2. Performance and Energy consumption analysis for four scenarios described above.

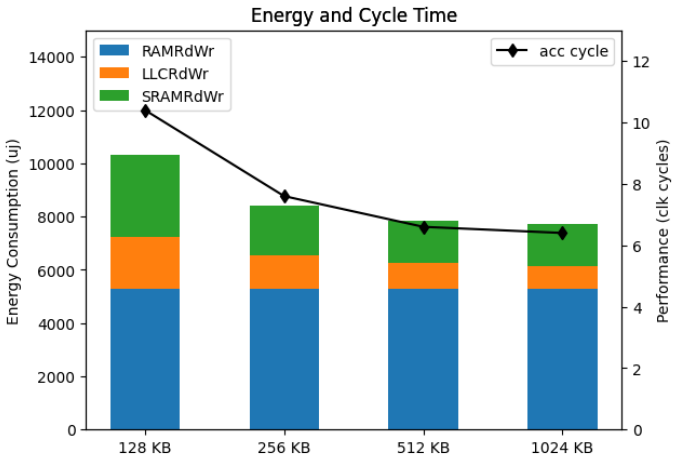

Fig. 3. Design space exploration for accelerator SRAM size.

C. Design Space Exploration

The size of on-chip memory, specifically the accelerator SRAM, is a critical factor affecting the overall performance of deep learning accelerators. Achieving high performance

requires a larger SRAM size to enable faster access for the accelerator. However, this comes at the cost of increased chip area utilization. Balancing the desire for a larger SRAM for improved performance with the need to minimize area overhead poses a design challenge. To address this, we conducted a comprehensive analysis, varying the sizes of the accelerator SRAM for case 4, and examined their impact on performance. The results, as depicted in Figure 3, indicate that there is a clear threshold in terms of performance improvement. Specifically, we observe that increasing the SRAM size up to 512 KB led to significant performance enhancements. However, beyond this threshold, there is a limited enhancement in performance since the majority of the input and weight values are already accommodated within the 512 KB SRAM.

V. Related Work

Memory latency is a critical factor that significantly impacts computational throughput. To mitigate this issue, several proposals have been introduced, focusing on enhancing execution and data access parallelism. One such proposal is hardware (HW) prefetching [2] [11], which demonstrates improved LLC miss rates. In our design, we propose utilizing a scratchpad memory along with software (SW) prefetching to ensure efficient data access. Unlike HW prefetching, which relies on predictions, SW prefetching offers a more organized approach suitable for static applications such as neural network inference.

For addressing dynamic memory requirements such as branches and dependencies, various implementations of hardware-assisted scheduling have been put forth [8] [10] [1]. However, in the case of static applications, such solutions are unnecessary.

Another enhancement involves organizing the accelerator SRAM into multibank partitions, allowing access to these submemories without encountering bank conflicts [13] [14]. We modeled our accelerator SRAM in a similar manner.

VI. Conclusion

In conclusion, our study highlights the crucial role of effective utilization of the LLC in deep learning accelerator systems, particularly evident with the inclusion of a prefetch unit and reconfiguration as a scratchpad. By leveraging the capabilities of the LLC, we observed significant improvements in both performance and energy efficiency. The prefetching mechanism, coupled with optimized data movement strategies, resulted in cache miss elimination and improved overall system performance. This enhanced performance translated into higher throughput and faster execution of deep learning workloads. Furthermore, the efficient use of the LLC contributed to reduced energy consumption by minimizing the need for frequent accesses to the main memory and energy-intensive operations. Our findings underscore the critical importance of maximizing the potential of the LLC in deep learning accelerator systems, emphasizing its role in achieving superior performance and energy efficiency.

References

[1] Saehyun Ahn, Jung-Woo Chang, and Suk-Ju Kang. An efficient accelerator design methodology for deformable convolutional networks. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3075–3079, 2020.

[2] Tao Chen and G. Edward Suh. Efficient data supply for hardware accelerators with prefetching and access/execute decoupling. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.

[3] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[4] Yun Chen, Ali Hajiabadi, Lingfeng Pei, and Trevor E. Carlson. New cross-core cache-agnostic and prefetcher-based side-channels and covert-channels, 2023.

[5] PyTorch Contributors. Pytorch documentation conv2d. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html, 2023. Accessed: 2023.

[6] NVIDIA Corporation. Nvdla: A free and open-source deep learning accelerator. *GitHub repository*, 2018.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Chen-Han Ho, Sung Jin Kim, and Karthikeyan Sankaralingam. Efficient execution of memory access phases using dataflow specialization. *SIGARCH Comput. Archit. News*, 43(3S):118–130, jun 2015.

[9] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014.

[10] Lana Josipović, Andrea Guerrieri, and Paolo Ienne. From c/c++ code to high-performance dataflow circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(7):2142–2155, 2022.

[11] Pavlos Malakonakis, Andreas Brokalakis, Nikolaos Alachiotis, Evripides Sotiriades, and Apostolos Dollas. Exploring modern fpga platforms for faster phylogeny reconstruction with raxml. In *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 97–104, 2020.

[12] David A. Patterson and John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. 2nd edition, December 11 2020.

[13] Dhananjaya Wijerathne, Zhaoying Li, Manupa Karunarathne, Anuj Pathania, and Tulika Mitra. Cascade: High throughput data streaming via decoupled access-execute cgra. *ACM Trans. Embed. Comput. Syst.*, 18(5s), oct 2019.

[14] Chen Yin, Naifeng Jing, Jianfei Jiang, Qin Wang, and Zhigang Mao. A reschedulable dataflow-simd execution for increased utilization in cgra cross-domain acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(3):874–886, 2023.