Leveraging Hot Data in a Multi-Tenant Accelerator for Effective Shared Memory Management

Chunmyung Park, Jicheon Kim, Eunjae Hyun, Xuan Truong Nguyen[*], Hyuk-Jae Lee
Department of Electrical and Computer Engineering, Seoul National University
{lukpcm, jckim, silverhj, truongnx, hyuk_jae_lee}@capp.snu.ac.kr

***Abstract*—Multi-tenant neural networks (MTNN) have been emerging in various domains. To effectively handle multi-tenant workloads, modern hardware systems typically incorporate multiple compute cores with shared memory systems. While prior works have intensively studied compute- and bandwidth-aware allocation, on-chip memory allocation for MTNN accelerators has not been well studied. This work identifies two key challenges of on-chip memory allocation in MTNN accelerators: *on-chip memory shortages*, which force data eviction to off-chip memory, and *on-chip memory underutilization*, where memory remains idle due to coarse-grained allocation. Both issues lead to increased external memory accesses (EMAs), significantly degrading system performance. To address these challenges, we propose HotPot, a novel multi-tenant accelerator with a *runtime temperature-aware memory allocator*. HotPot prioritizes *hot data* for global on-chip memory allocation, reducing unnecessary EMAs and optimizing memory utilization. Specifically, HotPot introduces a *temperature score* that quantifies reuse potential and guides runtime memory allocation decisions. Experimental results demonstrate that HotPot improves system throughput (STP) by up to 1.88× and average normalized turnaround time (ANTT) by 1.52× compared to baseline methods.**

I. Introduction

Deep neural networks (DNNs) have shown unparalleled accuracy and have been widely adopted in various tasks, including image recognition [1]–[4] and object detection [5]. The popularity of DNNs motivates the use of multi-tenant neural networks (MTNNs) in practical applications. For example, modern smartphones can simultaneously run different applications such as voice recognition, image processing, and personalized recommendations. Similarly, self-driving vehicles handle multiple tasks such as road/lane detection, object detection, and path planning. As MTNNs emerge as a new workload, they demand a new and effective computing system.

The accelerator design for MTNN has been intensively studied in previous works [6]–[12]. A generic MTNN accelerator typically consists of a global controller, multiple neural processing units (NPUs), a global scratchpad memory (GSM), and off-chip memory, as shown in Fig. 1. A global controller (e.g., ARM or RISC-V) allocates the hardware resource for multiple networks.

One common resource management strategy in multi-tenant accelerators is based on dynamic compute-aware allocation [6], [7]. Planaria [6] dynamically splits the accelerator into smaller units to enhance resource utilization, while Layer-Puzzle [7] uses latency prediction and dynamic scheduling to optimize resource allocation and improve throughput.

[*] Corresponding author.

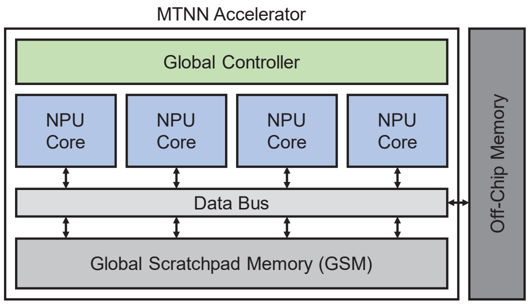


Fig. 1: General multi-core NPU architecture for multi-tenant workloads.

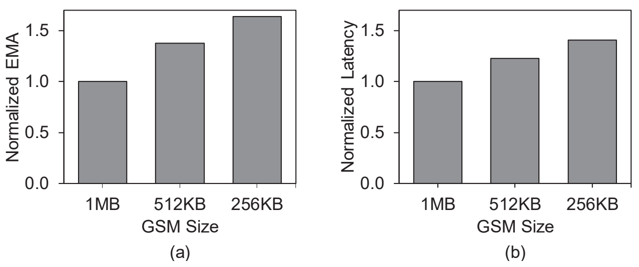


Fig. 2: Performance of an MTNN accelerator using varying GSM sizes. (a) Normalized external memory access. (b) Normalized average latency of networks.

Another typical strategy is based on off-chip memory bandwidth allocation [8], [9]. MAGMA [8] developed a genetic algorithm that maps and schedules tasks to avoid memory contention. MoCA [9] introduces a throttling mechanism to effectively manage the limited off-chip memory bandwidth during runtime, modulating external memory access (EMA) rates based on latency targets and user-defined priorities.

Unlike the previous works, this study focuses on *dynamic on-chip memory allocation* in MTNN accelerators. We identify two common issues while running MTNNs with limited on-chip memory, *on-chip memory shortage and underutilization* - that generally require evicting data to off-chip memory, thereby significantly degrading performance. For example, as shown in Fig. 2, an MTNN accelerator with 256KB of GSM increases EMA and latency by 1.64×, 1.41×, respectively, compared to one with 1MB GSM (see Section II-B for details). These observations pose a strong demand to design an MTNN accelerator with an effective GSM allocation.

To address these shortcomings, this work proposes HotPot, a novel MTNN accelerator with a runtime temperature-aware memory allocator. The key contributions of this work are as follows:

1) **Methodology.** HotPot introduces a *temperature score* to identify and prioritize *hot data* in GSM and employs fine-grained allocation to reduce EMAs and optimize memory utilization.
2) **Architecture and Memory Allocation.** HotPot utilizes a register in the NPU controller to measure the *temperature score* during runtime and modifies the DMA unit with additional registers to enable fine-grained allocation. The global controller executes a temperature-aware and fine-grained allocation algorithm to optimize GSM usage and minimize EMAs during runtime.
3) **Evaluation.** HotPot is implemented with Verilog HDL and evaluated through cycle-accurate RTL simulation. Experimental results demonstrate that HotPot achieves up to 1.88× improvement in system throughput (STP) and 1.52× improvement in average normalized turnaround time (ANTT) compared to baseline methods.

II. Background and Motivation

A. Multi-Tenant Neural Network Accelerators

MTNN workloads include multiple DNNs consisting of core operations such as convolution, batch normalization, pooling, and fully connected layers. Earlier works [10]–[12] execute MTNNs in a time-multiplexing manner utilizing a single NPU core. In contrast, recent MTNN accelerators [7]–[9] mainly use multiple NPU cores, as illustrated in Fig. 1.

Memory contention, caused by multiple cores competing for shared memory, has become as one of the main performance bottlenecks in recent MTNN accelerators [8], [9]. To address this issue, MAGMA [8] distributes memory-intensive tasks to prevent scheduling them together, thus mitigating memory contention. Similarly, MoCA [9] dynamically regulates memory requests by considering latency targets and user-defined task priorities. However, these works commonly rely on large on-chip memory (e.g., 2MB shared L2 memory [9]) which requires significant hardware area and may not be suitable for many practical cases such as edge computing [13], [14]. More importantly, relying on large GSM can mask issues like GSM shortages and underutilization, which increases EMAs, and significantly degrade performance, as will be explained in the following section.

B. On-chip memory management with MTNNs and Challenges

Utilizing on-chip memory to avoid EMAs is often considered during the compile time on conventional accelerators [15], [16]. With a given on-chip memory constraint and a single workload, these approaches employ optimized memory allocation schemes to minimize EMA. However, unlike conventional accelerators that run a single workload at a time, on-chip memory allocation in MTNN accelerators presents three new challenges: heterogeneity of GSM usage, GSM shortage, and GSM underutilization.

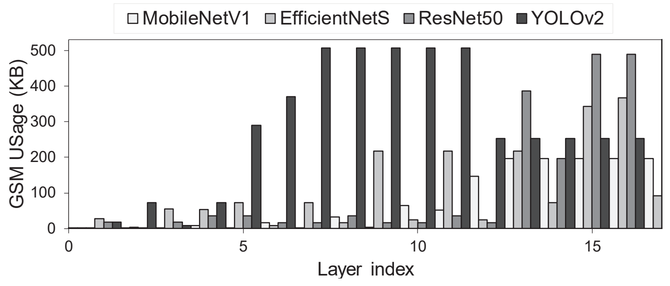


Fig. 3: Layer-wise GSM usage across different neural networks.

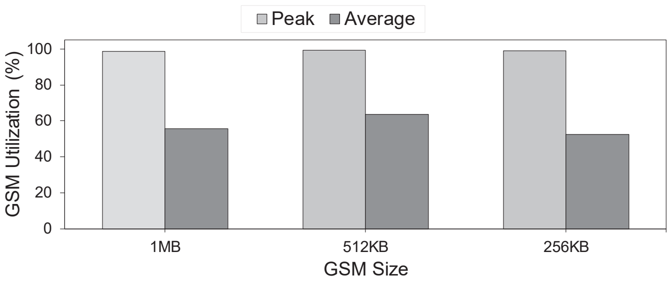


Fig. 4: Peak and average GSM utilization for different GSM sizes while executing multi-tenant workloads.

Challenge 1. Heterogeneity of GSM Usage. In an MTNN accelerator, multiple layers of different models are executed concurrently, each with varying memory requirements, as shown in Fig. 3. This diversity in memory usage becomes challenging when a new layer begins execution, as its arrival time is typically unpredictable. These heterogeneous and dynamic memory requirements demand a low-latency and cost-effective GSM allocator that can dynamically allocate memory space during runtime.

Challenge 2. GSM Shortage. GSM shortage may frequently appear in MTNNs, especially with limited on-chip memory, when the total required GSM usage of concurrent layers exceeds the GSM capacity. An GSM shortage may lead to two straightforward approaches: (1) pause a layer's execution until other layers free its on-chip memory space or (2) access off-chip memory instead of on-chip memory. In the first scheme, performance will be significantly reduced as an NPU core remains idle while waiting for the memory resource. With the second one, the computing core does not need to stay idle, but a layer's execution time will be increased due to additional EMAs.

To demonstrate GSM shortage, we ran four well-known neural networks, MobileNetV1 [1], EfficientNetS [2], ResNet50 [4], and YOLOv2 [5] on an MTNN accelerator with GSM sizes of 1MB, 512KB, and 256KB. When a GSM shortage occurred during runtime, the system retrieved data from off-chip memory (the aforementioned scheme 2). Fig.2(a) and 2(b) report normalized latency and amount of EMA occurred during runtime. It is observed that the accelerator with 512KB and 256KB GSM increased EMA by 1.38x and 1.64x, respectively, and increased the total latency by 1.23× and 1.41×, respectively, compared to the setup with 1MB GSM.

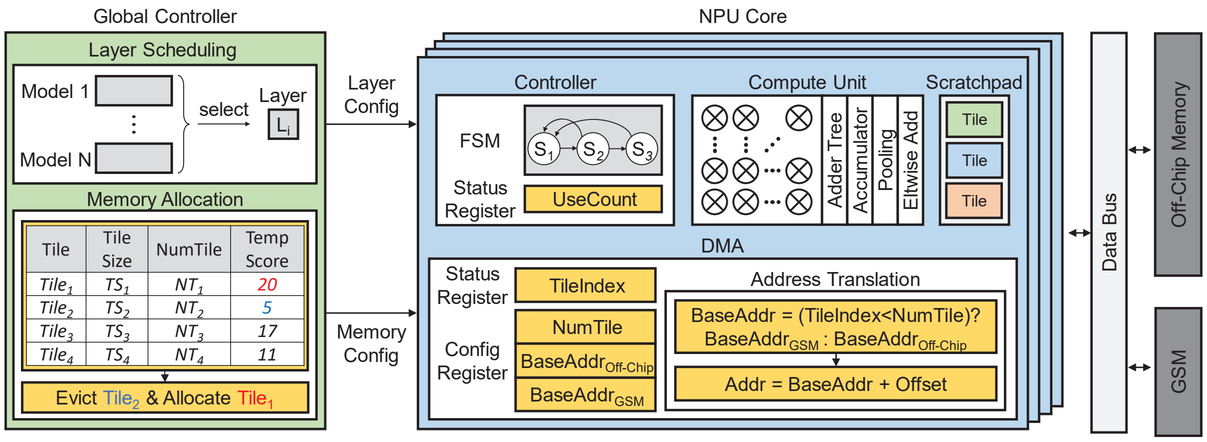


Fig. 5: Overall Architecture of HotPot.

Challenge 3. GSM Underutilization. Another challenge for on-chip memory allocation in MTNNs is underutilization. Fig. 4 shows the peak and average GSM utilization under various configurations. When using 1MB of GSM, the peak utilization reaches 98.7%, but the average utilization is only 55.8%, indicating that GSM remains underutilized for most of the runtime, leading to significant inefficiency. This problem persists even with smaller GSM sizes (e.g., 512KB, 256KB), where average utilization remains low at 63.7% and 52.6%, respectively. This inefficiency not only wastes on-chip memory resources, but also increases reliance on off-chip memory, leading to additional EMAs and latency.

III. HotPot

A. Methodology

This section presents two observations to mitigate on-chip memory shortage and underutilization. First, we propose temperature-based allocation to reduce EMA caused by on-chip memory shortage. Second, we propose fine-grained allocation to mitigate GSM underutilization.

Observation 1. Prioritizing Hot Data Reduces EMA. When a new layer begins execution while other layers are still running on the accelerator, on-chip memory shortages can trigger multiple EMAs. Conventional accelerators often prefetch frequently accessed data to maximize data reuse during computation. However, when this data is evicted to off-chip memory due to GSM shortages, the number of EMAs increases significantly. This increase can be quantified as shown in (1). $ReuseFactor$ denotes the number of access per data, and $MemSize$ represents the amount of data evicted to off-chip memory. This highlights the importance of prioritizing highly reused data (i.e., hot data) in GSM while evicting less reused data (i.e., cold data) to off-chip memory. By adopting this approach, EMA can be significantly reduced, thereby improving overall performance.

$$EMA = \sum ReuseFactor \cdot MemSize \tag{1}$$

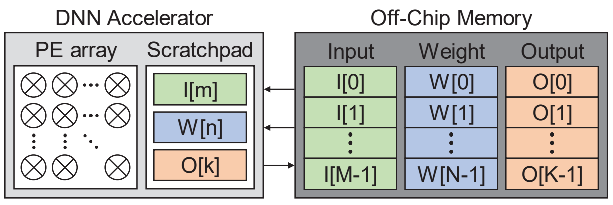
Observation 2. Fine-grained Allocation Mitigates GSM Underutilization. GSM utilization can be constrained by the granularity of memory allocation. For example, if an accelerator requires 60KB of memory to allocate a tensor but only 50KB is available, the entire tensor may be evicted to off-chip memory, leaving the 50KB unused. Fine-grained allocation resolves this issue by enabling partial data blocks to be stored in GSM while allocating the remaining blocks to off-chip memory, thereby maximizing utilization. However, conventional accelerators often lack this capability, resulting in significant underutilization. Supporting fine-grained memory allocation is therefore essential to enhance GSM utilization and ensure its efficient use.

B. HotPot Architecture

This section presents HotPot, a novel MTNN accelerator with an effective on-chip memory allocator. As shown in Fig. 5, It consists of a global controller and multiple NPU cores, each with its own controller, compute unit, and DMA unit. On each NPU core, HotPot utilizes a dedicated register in the NPU controller for tracking data temperature during runtime, which is used to compute a temperature score. Additionally, the DMA unit is enhanced with new registers to support fine-grained allocation, enabling more efficient use of GSM. Lastly, HotPot incorporates a global controller that orchestrates temperature-based and fine-grained memory allocation across all cores.

NPU Controller with a Temperature Score. EMA caused by on-chip memory shortage can degrade the system performance of an MTNN accelerator. To minimize EMA, it is crucial to keep highly reused data in GSM, as shown in (1). However, the number of reuse of data dynamically changes during runtime. As the NPU accesses data for computation, the remaining number of reuses gradually decreases. To quantify this dynamic change, a temperature score is defined as the difference between the total expected accesses per data (i.e., $ReuseFactor$) and the number of accesses that have already occured (i.e., $UseCount$), as shown in (2). The $ReuseFactor$ is determined at compile time based on the predefined dataflow, while $UseCount$ is tracked at runtime by finite state machine (FSM) inside the NPU controller and stored in a status register.

$$TempScore = ReuseFactor - UseCount \tag{2}$$



(a)

```
// UseCount monitored by loop variable m
for m = [0, M):
  Load I[m] // ReuseFactor = 1
  for n = [0, N):
    Load W[n] // ReuseFactor = M
    O[k] = I[m] * W[n]
    Store O[k]
⇒ TempScore of weight tiles = M - m
```

(b)

Fig. 6: (a) Overview of data loading from off-chip memory in a DNN accelerator. (b) Temperature score calculation.

Fig. 6 illustrates an example of temperature score calculation. The accelerator employs tiling to execute each layer, requiring input and weight tiles to be loaded multiple times from off-chip memory depending on the tiling factor and loop order. Input tiles are loaded only once ($ReuseFactor$ = 1), while weight tiles are loaded M times due to the outer loop ($ReuseFactor$ = M). As the computation progresses, the outer loop variable m increases, and the accelerator monitors m to calculate the $UseCount$ of weight tiles. The temperature score $(M - m)$ represents the remaining reuse potential of weight tiles. HotPot leverages this score for GSM allocation, prioritizing data with higher scores to reduce EMA.

DMA with Fine-Grained Address Registers. As mentioned earlier, on-chip memory underutilization primarily stems from the large granularity of memory allocation. To address this issue, we implement fine-grained allocation, which enables data to be allocated at tile-wise granularity. This allows some tiles to remain in GSM while the rest are placed in off-chip memory. To support this functionality, we modified the DMA unit of the NPU core, as shown in Fig. 5. The DMA unit has two base address registers: one for GSM and the other for off-chip memory. For each memory request, the DMA unit determines from where to load the data using a special register ($NumTile$). If the current tile index is smaller than the number of tiles allocated to GSM ($TileIndex < NumTile$), the request is sent to GSM. Otherwise, it is directed to off-chip memory. The DMA unit calculates the data address by adding an offset to the base address of GSM or off-chip memory, depending on the tile location.

Global Controller with Memory Allocation. Similar to previous works [7], [9], the global controller manages the layer allocation for each NPU core. It selects a layer from multiple models based on its scheduling policy and allocates it to an NPU core. However, unlike previous works [7], [9], HotPot's global controller also manages the memory allocation. It reads the register value ($UseCount$) from each core to calculate the temperature score of each layer, checks for GSM space overflow, and then makes a decision on memory allocation. Once the allocation is complete, the global controller sends a memory configuration to the NPU core, setting the configuration registers for the DMA unit.

C. Memory Allocation Algorithm

Algorithm 1 Memory Allocation Algorithm

```
1:  % Check overflow when allocating new tile $Tile_i$
2:  $overflow \leftarrow Size(Tile_i) \times Num(Tile_i) - memAvail$
3:  if $overflow > 0$ then
4:      % Find coldest tile among existing tiles $\rightarrow Tile_j$
5:      for tile $t$ in existing tiles do
6:          $TempScore(t) \leftarrow ReuseFactor(t) - UseCount(t)$
7:          if $TempScore(t) < temp_{min}$ then
8:              $(Tile_j, temp_{min}) \leftarrow (t, Temp(t))$
9:          end if
10: end for
11:     % Evict $Tile_j$ if colder than $Tile_i$
12:     if $Temp(Tile_i) > Temp(Tile_j)$ then
13:         $Num(Tile_j) \leftarrow \max\{Num(Tile_j) - \frac{overflow}{Size(Tile_j)}, 0\}$
14:     end if
15: end if
16: % Allocate memory for $Tile_i$
17: $Num(Tile_i) \leftarrow \min\{Num(Tile_i), \frac{memAvail}{Size(Tile_i)}\}$
18: % Set base address for GSM and off-chip mem
19: $SetBaseAddr_{GSM}(Tile_i)$
20: $SetBaseAddr_{off-chip}(Tile_i)$
```

Algorithm 1 illustrates the memory allocation process run by the global controller, leveraging temperature-based and fine-grained allocation. The global controller calculates the required memory by checking the tile size and the number of tiles for the new layer. If the requirements exceed available memory, it searches for the coldest tile in GSM by computing the temperature score. If the new tile ($Tile_i$) has a higher score than the coldest tile ($Tile_j$), $Tile_j$ is evicted to off-chip memory. The remaining number of $Tile_j$ is calculated as line 13. After eviction, memory for $Tile_i$ is allocated accordingly. Lastly, the base address registers are updated to point to the start of the allocated space.

IV. Evaluation

A. Methodology

1) Implementation and Settings: We developed a system-on-chip (SoC) platform with multicore NPUs to run multi-tenant workloads. The overall hardware architecture is shown in Fig. 5. We used an open-source RISC-V core [17] as the global controller and open-source IPs for on-chip interconnects [18]. The NPU cores were implemented based on a conventional accelerator [16], modified to support temperature-based and fine-grained allocation. The configuration of the hardware components is shown in Table I. We evaluated with four NPU cores, each with 32×32 MACs with INT8 precision. We used three different GSM sizes (1MB, 512KB, 256KB) to assess the impact of different GSM constraints. DRAM bandwidth is set to 32GB/s and the system frequency is set to 1GHz. All hardware components were implemented with Verilog HDL and evaluated through cycle-accurate RTL simulation.

The scheduling and memory allocation process was implemented in C, compiled with GCC, and stored as a hex file in instruction memory (i.e., RAM). During runtime, the RISC-V core loads instructions from this memory.

TABLE I: SoC Configuration

Parameter	Value
# of NPU cores	4
# of MACs per core	32×32
NPU precision	INT8
GSM size	1MB / 512KB / 256KB
DRAM bandwidth	32GB/s
Frequency	1GHz

2) Workloads: We used six popular CNN models for evaluation including MobileNetV1 [1], EfficientNetS [2], YOLOv2-Tiny [5], VGG-16 [3], ResNet50 [4], and YOLOv2 [5]. To evaluate multi-tenant workloads, we created three workload sets, as shown in Table II, similar to prior works [6], [9]. Workload set A includes lightweight models with small on-chip memory requirements, set B consists of heavy models that requires larger on-chip memory, and set C combines all six models, a mixture of light and heavy models. For image classification, the image size was set to 224×224, and for object detection, the input size was set to 416×416.

TABLE II: CNN Workloads

Workload	Model size	Domain	Model
Workload Set A	Light	Image Classification	MobileNetV1
			EfficientNetS
		Object Detection	YOLOv2-Tiny
Workload Set B	Heavy	Image Classification	VGG-16
			ResNet50
		Object Detection	YOLOv2
Workload Set C	Mixed	All	All

3) Metrics: We evaluated the performance of the multi-tenant accelerator using two metrics; System Throughput (STP) and Average Normalized Turnaround Time (ANTT). STP indicates the overall throughput of multi-tenant execution, calculated as the sum of normalized progress for each task. Normalized progress (NP) is defined as the ratio of solo-execution latency to multi-execution latency, as shown in (3). ANTT indicates the slowdown caused by resource contention and is calculated as the average of normalized turnaround time of each task, when normalized turnaround time (NTT) defined as the ratio of multi-execution latency to solo-execution latency, as shown in (4). While STP is higher-the-better metric, ANTT is lower-the-better metric.

$$STP = \sum_{i=1}^{N} NP_i = \sum_{i=1}^{N} \frac{solo\_latency_i}{multi\_latency_i} \tag{3}$$

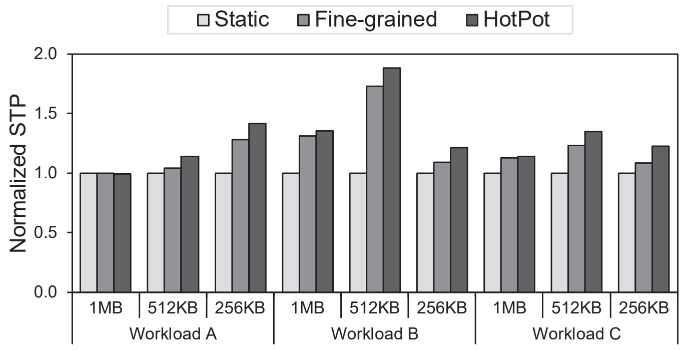


Fig. 7: STP comparison with baseline methods under various GSM size.

$$ANTT = \frac{1}{N}\sum_{i=1}^{N} NTT_i = \frac{1}{N}\sum_{i=1}^{N} \frac{multi\_latency_i}{solo\_latency_i} \tag{4}$$

For further analysis, we evaluated the GSM utilization and EMA to see the impact of fine-grained allocation and temperature-based allocation. GSM utilization was measured by collecting allocation status data from global controller, while EMA was measured using the traffic monitoring logic implemented in our system. Finally, we analyzed the latency breakdown of the global controller and NPU to assess the overhead associated with memory allocation.

4) Comparison: We compared HotPot with two baseline methods: *static memory allocation* and *fine-grained allocation*. *Static memory allocation* assigns a fixed amount of on-chip memory to each layer prior to execution, based on pre-compiled memory requirements. When the required memory exceeds the available on-chip memory, execution is paused until sufficient memory is released by other layers. On the other hand, *fine-grained allocation* allocates the data in tile-wise granularity, but does not consider the temperature score during the allocation.

B. Experimental Results

1) System Throughput (STP): We evaluated the STP performance of HotPot under various scenarios, running 30 randomly selected neural networks from the workload sets in TableII. We used three different GSM sizes (1MB, 512KB, 256KB) to assess the impact of different GSM constraints. As shown in Fig. 7, HotPot consistently outperformed baseline methods. With 512KB GSM, HotPot achieved a 42.5% geometric mean improvement, with a maximum improvement of 88.0% over the static allocation method. Compared to the fine-grained allocation method, HotPot showed a 9.55% geometric mean improvement, with a maximum of 9.89%. HotPot maintained strong performance across varying GSM sizes, except when using an oversized GSM for lightweight models (e.g. 1MB GSM for Workload set A).

2) Average Normalized Turnaround Time (ANTT): We evaluated the ANTT performance for each neural network model in Workload sets A and B with 512KB GSM. As shown in Fig. 8, HotPot significantly lowered ANTT by 19.1% for

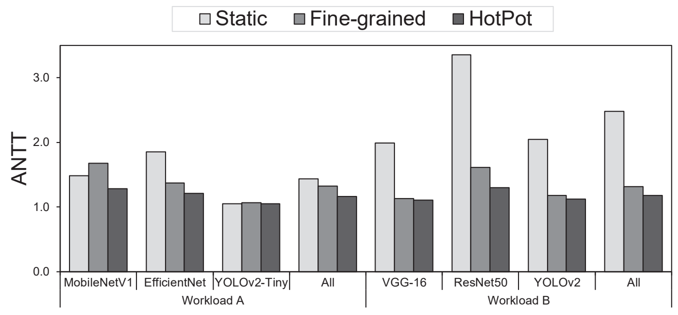


Fig. 8: ANTT comparison with a 512KB GSM.

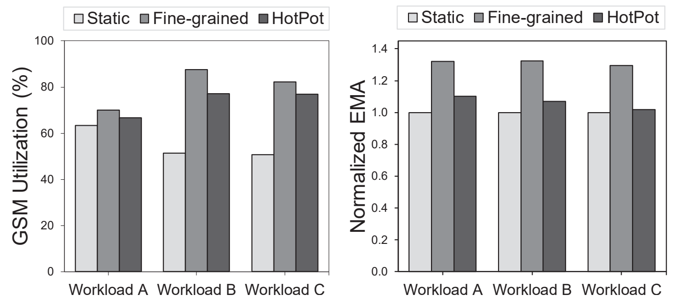


Fig. 9: GSM utilization and EMA comparison with a 512KB GSM.

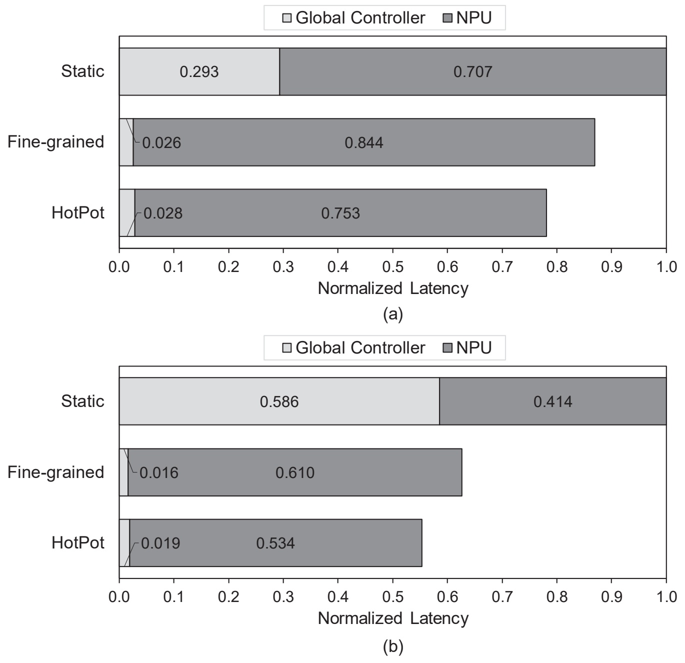


Fig. 10: Latency Breakdown of Workload A. (a) with 512KB GSM. (b) with 256KB GSM.

Workload set A and by 52.4% for Workload set B compared to the static method. Compared to the fine-grained allocation, HotPot reduced ANTT by 12.2% in Workload set A and by 10.3% for Workload set B. While baseline methods performed well with lightweight models (Workload set A), they struggled with heavy models (Workload set B) which require large on-chip memory. In contrast, HotPot achieved low ANTT across both workload sets, demonstrating its ability to handle on-chip memory limitations effectively.

3) GSM Utilization and External Memory Access (EMA): Fig. 9 shows GSM utilization and EMA while running multi-tenant workloads with a 512KB GSM. The static method achieved an average GSM utilization of 55.1%, whereas the fine-grained allocation reached 79.9%, demonstrating superior GSM utilization. However, since the fine-grained method focused solely on maximizing GSM utilization without considering data prioritization, it led to a 31.4% increase in EMA compared to the static method. In contrast, HotPot's temperature-based allocation prioritized hot data in GSM, resulting in a 19.0% reduction in EMA compared to fine-grained method.

4) Memory Allocation Overhead: Fig. 10 shows the latency breakdowns for Workload set A with different GSM sizes, normalized against the latency of static allocation method. In both the 512KB and 256KB GSM setups, the static method exhibited significant overhead due to frequent execution pauses, accounting for 29.3% and 58.6% of the total latency, respectively. In contrast, the fine-grained method, which triggers EMA rather than pausing, significantly reduces latency overhead by 11.2× compared to the static method. HotPot, although showing 1.09× and 1.16× higher latency overhead than the fine-grained method due to its memory allocation computations, ultimately minimized total latency by effectively reducing EMA. This demonstrates that despite the overhead from its temperature-based memory management, HotPot consistently outperforms other methods in terms of total system efficiency, especially under constrained memory conditions.

V. Conclusion

In this work, we introduced HotPot, a new accelerator designed for multi-tenant neural networks that address challenges in on-chip memory allocation. By using temperature-based memory allocation, HotPot prioritizes hot data to stay in on-chip memory, reducing external memory access. Our experiments showed that HotPot achieves significant improvements in both system throughput and turnaround time, outperforming baseline methods. Even in scenarios with limited on-chip memory, HotPot demonstrated its effectiveness, reducing overhead and improving overall performance. These results highlight HotPot as a practical and efficient solution for improving memory utilization and performance in multi-tenant systems.

Acknowledgment

This work was supported in part by the Technology Innovation Program (or Industrial Strategic Technology Development Program – No. 20014490, Development of Technology for Commercializing Lv.4 Self-driving Computing Platform Based on Centralized Architecture) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) and in part by the IITP grant funded by MSIT (No.RS 2023-00228255, PIM-NPU Based Processing System Software Developments for Hyperscale Artificial Neural Network Processing). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

References

[1] Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[2] Mingxing Tan. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017. ISSN: 1063-6919.

[6] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 681–697. IEEE, 2020.

[7] Chengsi Gao, Ying Wang, Cheng Liu, Mengdi Wang, Weiwei Chen, Yinhe Han, and Lei Zhang. Layer-puzzle: Allocating and scheduling multi-task on multi-core npus by using layer heterogeneity. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[8] Sheng-Chun Kao and Tushar Krishna. Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 814–830. IEEE, 2022.

[9] Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 828–841. IEEE, 2023.

[10] Yujeong Choi and Minsoo Rhu. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 220–233. IEEE, 2020.

[11] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. A multi-neural network acceleration architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 940–953. IEEE, 2020.

[12] Young H Oh, Seonghak Kim, Yunho Jin, Sam Son, Jonghyun Bae, Jongsung Lee, Yeonhong Park, Dong Uk Kim, Tae Jun Ham, and Jae W Lee. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 584–597. IEEE, 2021.

[13] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35:22941–22954, 2022.

[14] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus. *IEEE Transactions on Computers*, 70(8):1253–1268, 2021.

[15] Jicheon Kim, Chunmyung Park, Eunjae Hyun, Xuan Truong Nguyen, and Hyuk-Jae Lee. A highly-scalable deep-learning accelerator with a cost-effective chip-to-chip adapter and a c2c-communication-aware scheduler. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2024.

[16] Duy Thanh Nguyen, Hyeonseung Je, Tuan Nghia Nguyen, Soojung Ryu, Kyujoong Lee, and Hyuk-Jae Lee. Shortcutfusion: From tensorflow to fpga-based accelerator with a reuse-aware memory allocation for shortcut data. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(6):2477–2489, 2022.

[17] YosysHQ. picorv32. https://github.com/YosysHQ/picorv32.

[18] Andreas Kurth, Wolfgang Rönninger, Thomas Benz, Matheus Cavalcante, Fabian Schuiki, Florian Zaruba, and Luca Benini. An open-source platform for high-performance non-coherent on-chip communication. *IEEE Transactions on Computers*, 71(8):1794–1809, 2021.