Flexible Photonic Memory Pooling Architecture For Efficient Compute Resource Allocation

Zhenguo Wu[1] and Keren Bergman[1]

[1] *Department of Electrical Engineering, Columbia University, 500 W 120th St., New York, New York, USA. 10027*

zw2542@columbia.edu

Abstract: We present a photonic memory pooling architecture and a co-designed optimization methodology for flexible allocation of compute, memory, and network resources. We demonstrate up to 5.4× to 7.5× improvements in DL training and inference time.

1. Introduction

The rapid advancement of artificial intelligence applications has created diverse computational requirements for computing systems. Specifically, the training and inference of large-scale deep learning (DL) models have diverse demands on compute power, memory capacity, and memory bandwidth. Current compute racks mainly consist of computing units (CUs) that are connected to a fixed amount of high-bandwidth memory (HBM), limiting their ability to adapt to changing demands. To enable more efficient use of compute and memory resources, memory pooling has been proposed, where CUs are connected to a shared pool of memory units (MU) for dynamic memory allocation. Current memory pooling approaches generally use local HBMs as high-bandwidth suppliers, while slower remote MUs (e.g., DDR, GDDR) connected via a network fabric are used as capacity expanders. One reason for having the memory hierarchy is the higher latency introduced by the networking layer when accessing remote memory pools. Another reason for this setup is that HBM's massive pin count is limited by the physical length of electrical traces, requiring it to be placed near the compute die, which has limited periphery length.

In this work, we present a Silicon Photonic Accelerated Memory-Pooling architecture (SiPAM). SiPAM expands the current memory pooling design space by leveraging embedded silicon photonic (SiP) I/Os to create a unified high-bandwidth domain for both memory and network data movement. We propose a co-designed optimization methodology to dynamically allocate compute power, memory capacity, and memory bandwidth for different workloads. We validate the SiPAM architecture and optimization methodology using an analytical DL evaluation framework. Evaluation results demonstrate 5.4× to 7.5× improvements in training and inference completion time compared to Nvidia H100 based compute system connected over NVLink fabric on DL workloads.

2. System Architecture

The SiPAM architecture adopts SiPAC's physical topology design [1] by using Optical Circuit Switches (OCS) and Dense Wavelength Division Multiplexed (DWDM) links to replace electrical Top-of-Rack (ToR) switches and links (Fig.1a), achieving a multi-dimensional all-to-all topology. Additionally, it adopts the intra-rack resource disaggregation model [2] by replacing some of the compute trays in a rack with memory trays containing remote MUs (Fig.1a,b). The packet-switch-less design eliminates intermediate electrical-to-optical conversions and data buffering, reducing in-network delays for low-latency remote memory access.

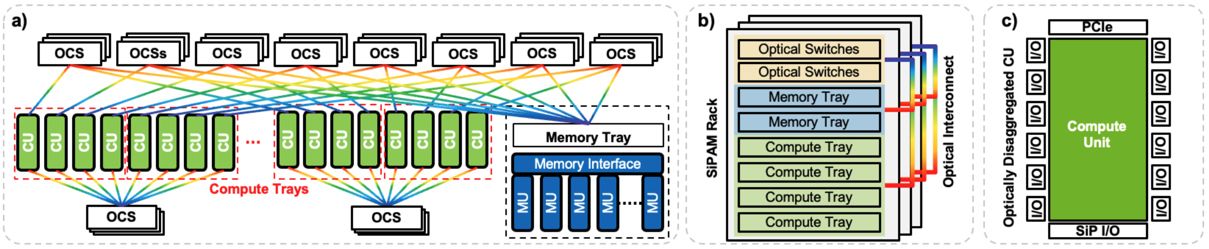

Fig. 1: Schematic of SiPAM architecture adopting (a) the SiPAC [1] physical topology and (b) the intra-rack resource disaggregation model with (c) embedded SiP I/Os around each compute die, replacing local HBMs.

The compute architecture is designed with the compute die's shoreline width as a critical resource. Recent works on silicon photonic chip I/O have demonstrated embedded SiP chip delivering >2 TB/s of bandwidth within dimensions of 8.10$mm \times$ 8.62mm [3]. This achieves HBM-comparable bandwidth without distance limitations, making it feasible to connect larger pools of remote HBMs to increase both capacity and bandwidth. Therefore, SiPAM directly embeds high bandwidth SiP I/Os [3] along the compute die's periphery, fully replacing HBMs and high-speed inter-CU networking interfaces (Fig. 1c) to create a high-bandwidth communication domain for both memory and network data movement. Each SiP I/O can be flexibly allocated for memory access or network communication through an optically switched fabric, with a one-time reconfiguration before each workload starts, guided by our co-designed optimization methodology. To interface with the multiple SiP I/Os per CU, we extend the SiPAC design by adding rails of OCSes, ensuring full memory bandwidth to the memory pool when needed. For the memory pool, while we do not specify a particular memory interface in this work, CXL [4] is an example memory semantic interconnect showing promise for future adaptation to optical communication. We acknowledge that time-of-flight delays and interface protocol overhead will introduce additional latency compared to directly attaching HBMs around the compute die. However, a recent study on CXL interfaces has shown that this increased memory latency can be entirely mitigated by increasing bandwidth when the memory system is fully loaded [4].

3. Optimization Methodology

We propose an optimization methodology to dynamically allocate compute power, memory bandwidth, and memory capacity for a given workload. First, given the peak compute floating-point operation per second (FLOPs/s), F_{peak}, of the CU (Fig.2a) and the arithmetic intensity, I_w, of the workload (Fig.2b), the minimum required memory bandwidth can be calculated as $B_{req} = F_{peak}/I_w$ which corresponds to the ridge point on the roofline plot (Fig.2c). We assume that this ridge point stays invariant with increasing number of CUs, meaning both compute power and memory bandwidth increase proportionally as additional CUs are introduced. Each MU in the memory pool provides a fixed bandwidth of B_m, allowing us to determine the total number of required MUs: $N_m = \lceil B_{req}/B_m \rceil$ (Fig.2d). From this, we can determine the number of SiP I/Os allocated for the memory pool: $N_{IO,m} = \lceil (N_m B_m)/B_{IO} \rceil$, where B_{IO} is the bandwidth per SiP I/O (Fig.2e). Any remaining I/Os will be used for connecting to other CUs in the SiPAC topology. Next, we estimate the number of CUs needed to fit the workload as: $N_c = C_w/(N_m C_m)$ where C_w and C_m represent the size of the workload and the capacity of each MU (Fig.2f). We then determine the parallelization strategy based on N_c. Depending on the chosen strategy, the memory capacity and bandwidth requirements may change. In such cases, we revisit the previous steps to adjust the configuration accordingly. This iterative process continues until a stable configuration is achieved or hitting a predefined iteration threshold, which we consider the optimal solution. Note that N_c represents the minimum number of CUs required to host the workload. When scaling up N_c, both compute and memory are increased proportionally, maintaining the optimal compute-to-memory ratio.

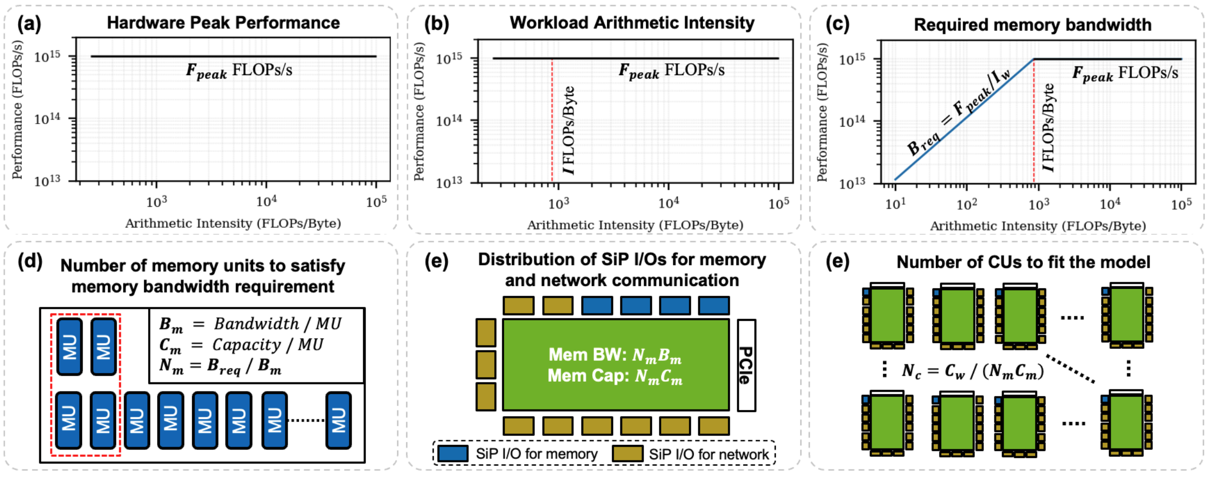

Fig. 2: Step-by-step illustration of the optimization methodology to determine the optimal resource allocation.

4. System-Scale Evaluation

We use an analytical DL model, Calculon [5], to evaluate the performance of our proposed SiPAM architecture on Transformer-based workloads of varying parameter count. The arithmetic intensity of each workload is estimated as the ratio of the total number of FLOPs to the total memory bytes accessed. And the total workload size is estimated by summing the memory requirements for weights, activations, weight gradients, activation gradients, and optimizer states. For baseline comparison, we model the Nvidia H100 compute architecture, where each CU has 1000 TFLOPs (float16) compute power and a fixed 80 GB of HBM (5×16 GB) with a combined memory bandwidth of 3000 GB/s (5×600 GB/s). CUs in the baseline are connected within NVLink clusters of 8 at 900 GB/s and InfiniBand networks between clusters at 100 GB/s. For SiPAM, we assume that CUs have the same compute power as the baseline and that each CU has a periphery length of 96 mm, allowing it to accommodate up to 12 SiP I/O modules of 8 mm width [3]. For remote memory access, each MU provides 16 GB of capacity at 600 GB/s bandwidth. In addition to HBM access latency, we incorporate an additional latency of 60 ns based on estimates from [4] to account for intra-rack time-of-flight and other protocol overheads. We normalize the architectures to the same per-CU bandwidth that is equivalent to the combined memory and network bandwidth of the baseline.

From Fig.3a) and b), we observe that training workloads exhibit higher arithmetic intensity than inference, indicating that they perform more FLOPs per byte of data moved from memory due to their greater computational demands. Across different workloads, SiPAM's compute intensity closely tracks each workload's arithmetic intensity for both training and inference, and the slight differences are due to rounding operations during optimization. In contrast, the baseline compute intensity remains constant across different workloads since its compute-to-memory bandwidth ratio is fixed. Fig.3c) and d) show the iteration time for training and inference normalized to SiPAM's iteration time for each workload. SiPAM consistently outperforms the baseline architecture by optimally allocating the needed resources based on each workload. This results in up to 5.4× to 7.5× reduction in iteration time for training and inference, respectively. We observe that our proposed methodology is particularly effective for memory-bound workloads, as it can provide additional memory bandwidth by connecting more MUs from the memory pool. For workloads where the arithmetic intensity matches the compute intensity of the baseline, we observe minimal gains from SiPAM. For compute-bound workloads, the performance improvement comes from the optimization process allocating more CUs to the task. Overall, inference tasks benefit more from the dynamic allocation of additional memory bandwidth, given their memory-bound nature.

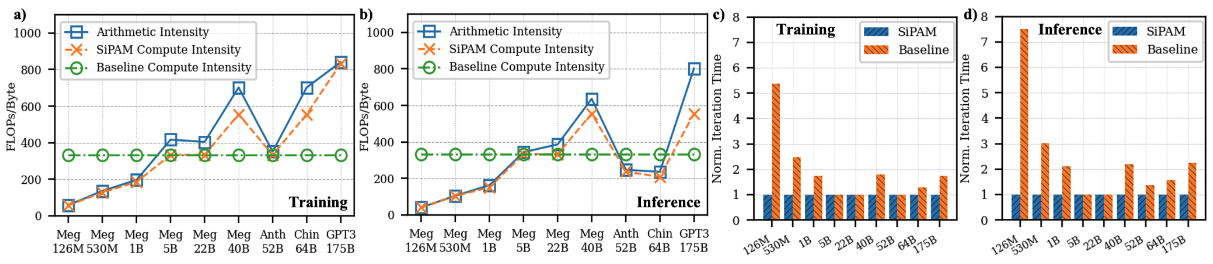

Fig. 3: Arithmetic and compute intensity for training (a) and inference (b). Normalized iteration completion time for training (c) and inference (d). The workloads shown are Transformer-based, with their parameter count indicated next to each workload abbreviation.

5. Conclusion

In this work, we propose a silicon photonic based memory pooling architecture and an optimization methodology to optimally allocate compute power, memory capacity and bandwidth. We report system-level evaluation results that show up to 5.4× to 7.5× workload completion time improvement over Nvidia H100 based compute system.

Acknowledgments. This work is supported in part by the ARPA-E ENLITENED Program, the National Security Agency (NSA) Laboratory for Physical Sciences (LPS) Research Initiative, and the Center for Ubiquitous Connectivity (CUbiC), and is sponsored by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) under the JUMP 2.0 program.

References

1. Z. Wu *et al.*, "Flexible silicon photonic architecture for accelerating distributed deep learning," *Journal of Optical Communications and Networking*, vol. 16, no. 2, pp. A157–A168, 2024.
2. G. Michelogiannakis *et al.*, "Efficient intra-rack resource disaggregation for hpc using co-packaged dwdm photonics," in *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2023, pp. 158–172.
3. Y. Wang *et al.*, "Silicon photonics chip i/o for ultra high-bandwidth and energy-efficient die-to-die connectivity," in *2024 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2024, pp. 1–8.
4. A. Cho *et al.*, "A case for cxl-centric server processors," *arXiv preprint arXiv:2305.05033*, 2023.
5. M. Isaev *et al.*, "Calculon: a methodology and tool for high-level co-design of systems and large language models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–14.