Exploring the Wafer-Scale GPU

Daoxuan Xu
William & Mary
Williamsburg, VA, USA
dxu05@wm.edu

Le Xu
UT Austin
Austin, TX, USA
le.xu@bytedance.com

Jie Ren
William & Mary
Williamsburg, VA, USA
jren03@wm.edu

Yifan Sun
William & Mary
Williamsburg, VA, USA
ysun25@wm.edu

Abstract

Wafer-scale GPU technology has recently been proposed to improve the scalability of multi-GPU systems. By expanding a multi-chiplet module GPU design to the scale of the whole wafer, GPU chiplets can communicate much faster than using off-chip networks. Yet, while the technology is mature in the Electrical Engineering domain, architecture- and system-level research is limited, impeding the deployment of wafer-scale GPUs. In this paper, we perform a series of simulator-based design explorations using three experiments that cover hardware organization, system-level scheduling, and software adaptation. In the hardware organization experiment, we explore how the GPU chiplet size impacts the performance while keeping the wafer size unchanged. The system-level scheduling experiment explores how the performance changes if we change the number of chiplets used, revealing the scalability limitation of current GPU kernels running on wafer-scale GPUs. In the software adaptation experiment, we explore the possibilities of forming pipelines on a wafer to improve performance. Overall, we find that wafer-scale GPUs demonstrate unique properties, and the performance improvement of wafer-scale GPUs cannot rely on solutions established for traditional multi-GPU systems and MCM-GPU devices.

CCS Concepts

• **Computer systems organization → Parallel architectures**; **Multicore architectures**; **Single instruction, multiple data**; **Interconnection architectures**.

Keywords

Wafer-Scale GPU, GPU, Network-on-Chip

ACM Reference Format:
Daoxuan Xu, Le Xu, Jie Ren, and Yifan Sun. 2025. Exploring the Wafer-Scale GPU. In *Proceedings of The 17th Workshop on General Propose Processing Using GPU (GPGPU'25).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The ever-increasing complexity of modern workloads demands higher and higher GPU computing capability [12] [9] [5] [2]. Due to the physical limitations, the die size and total number of transistors in a single GPU cannot increase indefinitely. The multi-chip-module (MCM) allows GPUs to equip more computing resources beyond the size limitation of a single die, providing higher performance [1, 7] [4]. While each chip is not getting more powerful, chips within one package are connected with an in-package network, which is faster than off-chip interconnects (e.g., PCIe). In theory, we get better scalability if we can connect more computing resources with faster networks. The limit is to utilize the whole wafer, forming wafer-scale GPUs.

Recent research and implementation demonstrated two different wafer-scale chip designing paradigms. The small tile wafer-scale GPU is designed to satisfy the commercialization and the requirement of yield, which has been implemented in Cerebras products [6]. Engineers designed a thousand small and simple computational structures and connected them with a huge mesh network. Conversely, the larger tile with fully functional processors simulated in the university's lab shows their powers [8], researchers at UIUC interposed conservative GPUs in a wafer and connected them together.

Researchers in UIUC describe a prototyped wafer-scale GPUs with advanced technologies in the paper [8]: 1) Silicon Interconnect Fabric (Si-IF) on a passive interconnect silicon substrate, allowing the links to provide high density, high bandwidth, low latency, and low energy communication. 2) Copper pillar-based I/O pins, exposed on the surface of the silicon substrate, serving as the physical chip-network interface. 3) Serially connected GPUs, allowing multiple tiles to share a power management circuit [13]. 4) Package-free circuits, allowing GPM, HBM, and memory management circuits to be directly placed on the silicon substrate. The prototyped wafer-scale GPU, alongside commercialized wafer-scale processors [11] demonstrates that the networking, packaging, and cooling solutions are no longer barriers to implementing wafer-scale GPUs.

While the technology barriers are resolved, architectural research of wafer-scale GPU is still in its infancy. Existing research dealt with wafer-scale GPUs as a special case of multi-chip-module GPUs [8], several graphic processing models (GPM) are placed on

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GPGPU'25, March 1, 2025, Las Vegas, NV

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

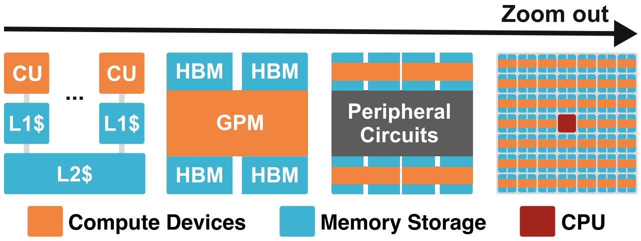

Figure 1: The structure of a wafer-scale GPU. From left to right: the GPM architecture, tile architecture, multiple tiles share a peripheral circuit, and the overview of the wafer-scale GPU.

a wafer-scale mesh network. neglecting the uniqueness of wafer-scale GPUs. For example, the wafer-scale GPUs are likely too large to be utilized by a single kernel. An under-explored problem is the proper zoning and pipelining in wafer-scale GPUs so that the GPU can run multiple kernels simultaneously. Addressing challenges like zoning and pipelining is required before users can harness the power of wafer-scale GPUs.

Designing solutions for wafer-scale GPUs requires systematic research. Rather than providing an ultimate design, this paper inspires future research directions at the architecture level related to wafer-scale GPUs. This paper aims to perform preliminary experiments that demonstrate the unique challenges of wafer-scale GPU system design. Specifically, we consider three research directions that may need to be addressed, including 1) hardware organization, 2) system-level scheduling, and 3) software-level adaptation.

We consider wafer-scale GPUs as an array of identical GPU tiles (consisting of GPMs and HBMs) arranged in a grid-like pattern (see Figure 1). In this paper, we explore unique problems related to wafer-scale GPUs from three aspects, including 1) hardware organization, 2) system-level scheduling, and 3) software-level adaptation.

1) Hardware organization. Hardware organization design needs to decide the tile sizes. Considering the yield, smaller tiles are cost-efficient. However, small tiles increase the overhead of peripheral circuits (e.g., DeCap, power management circuits) and communication latency. Therefore, we perform a study that explores how the tile size impacts the overall performance. Our results inform design decisions that balance performance and manufacture cost.

2) System-level scheduling. MCM-GPUs are typically marketed as single GPU units. However, as system scales grow, modern workloads become less likely to fully utilize an entire wafer. It's often unnecessary to use all tiles for a single task. We must determine the optimal tile count for each workload.

3) Software-level adaptation. New hardware needs software support. A single kernel often fails to fully utilize an entire wafer. Current GPU programming frameworks, which execute kernels sequentially, do not provide the necessary APIs for users to define spatial partitions and layouts. We believe that the choice of partitioning scheme is critical for optimizing system performance.

The contribution of this paper includes:

- Set of exploratory experiments on wafer-scale GPUs, highlighting how wafer-scale GPUs differ from MCM-GPUs.
- Discussion of potential future research directions on wafer-scale GPU.

Table 1: Size of different Components

Benchmark	CTA count
Finite Impulse Response (FIR)	65536
Fast Fourier Transform (FFT)	262144
Matrix transpose (MT)	65536
Image to Column (I2C)	516128
PageRank (PR)	262144
ReLU	65536
Sparse matrix-vector multiplication (SPMV)	65536
Simple convolution (SC)	65665
Stencil 2D (S2D)	65408

2 Methodology

We design three experiments that explore wafer-scale GPU design decisions at hardware, system, and software levels.

Simulator. We developed Akkalat, a modified version of MGPUSim [10] tailored for wafer-scale GPU evaluation. Major additions to MGPUSim in Akkalat include a mesh network and configurations for wafer-scale GPUs. We change tile configurations, including CU counts, L2 Cache size, memory size, the number of tiles, and the bandwidth in the mesh network to model different wafer-scale GPUs.

Workloads. We evaluate wafer-scale GPU with a wide range of benchmarks supported by MGPUSim. The benchmark input sizes shown in Table 1 are large enough to utilize all the compute units within all tiles and fixed in all experiments (strong scaling) in this paper.

Configuration. We configure the wafer-scale GPU as mesh network-connected tiles, with each tile equipping a certain number of CUs and a certain amount of memory. We designate the center tile as the CPU, which is responsible for dispatching kernels and providing page-fault handling services.

All instructions are pre-filled into the instruction cache in each tile, kernels and memory are evenly distributed to the selected tiles. For memory, the pages are also evenly distributed to all the tiles (aligned to page boundaries). For kernels, the thread blocks (a thread block is a group of threads typically between 64 and 512) are also evenly distributed to all the tiles.

2.1 Performance vs. Tiles Size

In the first experiment, we explore the relationship between tile size and performance, considering wafers with diameters of $215mm$ and area of $46,255mm^2$. Keeping the wafer-size unchanged, we alter the die size of each tile and the total number of tiles. A tile comprises two parts: Graphics Processing Module (GPM) and High Bandwidth Memory (HBM). To save space for placing more tiles, multiple tiles shares peripheral circuits (i.e., voltage regulator module (VRM), decoupling capacitors (DeCap), and V^{int} regulator [8])). The number of tiles connected to a single VRM depends on the voltage supply from the peripheral circuits. We assume the voltage for a tile corresponds to its Compute Unit (CU, a CU is a GPU core) count, and the number of tiles connected with a periphery circuit is scaled up or down based on the CU count of a tile. We estimate the die size of each component of a tile and the memory controllers

Table 2: Size of different Components

Component	Size (mm^2)	Component	Size (mm^2)
CU + L1$	3.12	HBM (8GB)	92.00
L2$/MB	4.34	VRM & DeCap	1, 380
HBM Control	14.90	V^{int} Regulator	200

Table 3: Tile configuration and number

CU/GPM	L2/GPM	HBM	bandwidth	GPM/Wafer
256	16MB	64GB, 8 ctrler	6TB/s	14
128	8MB	32GB, 4 ctrler	3TB/s	27
64	4MB	16GB, 4 ctrler	1.5TB/s	48
32	2MB	8GB, 2 ctrler	768GB/s	80
16	1MB	4GB, 1 ctrler	384GB/s	121
8	512KB	2GB, 1 ctrler	192GB/s	158
4	256KB	1GB, 1 ctrler	96GB/s	182

from the die shot of an AMD MI100 GPU [7], and estimate the size of the peripheral power supply circuits according to previous on-die implementations of wafer-scale GPUs [8]. The die size of each component is detailed in Table 2.

We generate 7 different configurations (see Table 3), ranging from 4 CUs per tile to 256 CUs per tile. We also scale the L2 cache size, HBM memory size, and network bandwidth in proportion to the computing power of each tile. We arrange the tiles on a wafer to be as close to a rectangle as possible.

This experiment is unique to wafer-scale GPUs as the hop count is not a major concern in MCM GPUs. When each tile has 64 CUs, the maximum count can be 13 (7×7 tile), creating a major performance and scalability issue.

2.2 Running GPU Kernels with Part of the Wafer

In the first experiment, each kernel is executed by all the tiles on the wafer. However, given that the GPU programs were not written for wafer-scale GPUs, scalability issues may arise. While utilizing more tiles provides additional computational resources, it also may increase communication overhead due to long-distance, cross-wafer communication. Reducing the number of tiles allocated for the kernel may not harm performance but can potentially spare resources for another workload, improving overall utilization.

Aligning with prior work[8], we use 64-CU tiles in the following experiments. We assume the whole wafer-scale GPU contains a 7×7 mesh network. When scheduling tasks to part of the wafer, we organize tiles to form a square, which can reduce the maximum communication distance. We evaluate 1×1, 2×2, 3×3, 4×4, 5×5, 6×6, and 7×7 configurations. We always use the tiles from the top-left corner, so when the size is greater or equal to 4×4, and one tile is the CPU.

This experiment is unique to wafer-scale GPUs because scalability is a more serious problem than resource-limited single GPUs. Moreover, the size of the tile groups assigned to execute a kernel can make a performance difference as the size impacts communication distances.

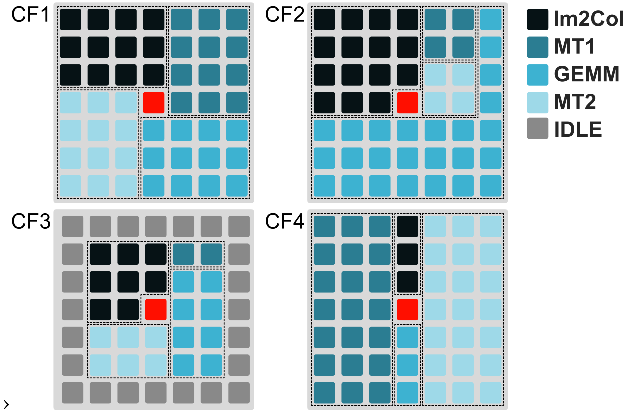

Figure 2: Various pipelining configurations for wafer-scale GPUs. "CF" stands for "configuration".

2.3 Pipelining Kernels on a Wafer-Scale GPU

The second experiment suggests utilizing too many tiles may not yield the best performance (see Figure 4). Following this insight, we partition the wafer-scale GPU into several sub-GPUs and pipeline tasks on each sub-GPU. In this experiment, we select 2D convolution, a key part of convolutional neuron network workloads. The 2D convolution implementation contains four kernels, including image-to-column conversion (Im2Col), matrix transpose(MT1), general matrix multiplication (GEMM), and another matrix transpose (MT2).

In this experiment, we still use the 7×7 wafer-scale GPU configuration. Since there are four kernels in 2D convolution, the wafer-scale GPU is partitioned into four sub-GPUs. We try four different partition configurations (see Figure 2). We define the four configurations as follows: CF1) evenly split the wafer, CF2) use the best performing sub-GPU size for IM2Col and MT according to experiment 2 and use the rest for GEMM, CF3) minimize communication distance with a cost of not utilizing some tiles, and CF4) minimize the matrix transform execution time (we intentionally add this design to demonstrate how low the performance can be in a bad pipeline design). We also divide the input data into four sub-batches to form a pipeline.

This experiment is unique to wafer-scale GPUs because scalability is a more serious problem than resource-limited single GPUs. Moreover, the size of the tile groups assigned to execute a kernel can make a performance difference as the size impacts communication distances.

While pipelining is not unique to wafer-scale GPU systems, assigning the tiles to the stages is not present in MCM-GPUs or regular multi-GPU systems. The assigning should balance the computation requirement, intra-kernel communication distance, and inter-kernel communication distance to achieve the best performance and utilization.

3 Results

Next, we show the results of the proposed experiments.

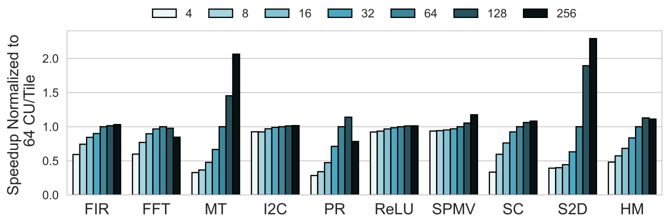

Figure 3: Simulation results for different tile size configurations (normalized by 64CU/tile configuration).

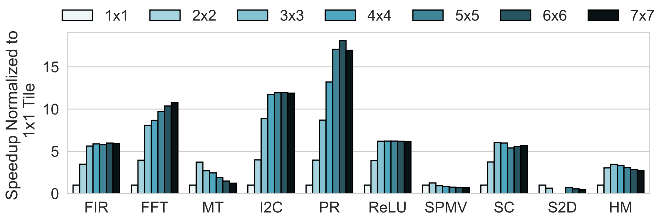

Figure 4: Normalized speedup when using different numbers of tiles on each kernel (normalized to only using a single tile). The results suggest that different benchmarks have different saturation points.

Performance vs. tile size. In the first experiment, we examine how the tile size impacts performance. Note that since we keep the total wafer size the same, wafers with smaller tiles have lower overall computing capability due to the overhead of peripheral circuits.

Our results (see Figure 3) suggest that the performance generally improves as the tile size increases. We use harmonic means [3] to compare the overall performance improvement between different configurations. The improved performance is because of the higher computing power, higher GPU-GPU bandwidth and fewer hops required to route data movements. However, we also observe that some benchmarks (e.g., I2C, ReLU, and SPMV) are not sensitive to tile sizes. This is because they do not require much inter-GPU communication and are constrained by local memory bandwidth.

Additionally, we find that, in many benchmarks (e.g., FIR, FFT, I2C, PR, ReLU, and SC), the performance stops increases when the tile size is greater than 64. In FFT and PR, the 256-CU-per-tile wafer is even slower than the 128-CU-per-tile wafer. This is because the overall available resources are similar in 64-, 128-, and 256-CU-per-tile wafers. Only the MT and S2D benchmarks always benefit from larger tiles because these two benchmarks involve massive data movement, and fewer hops result in smaller queueing latency at switches. Considering larger tiles are much more expensive to manufacture, we consider 64-CU-per-tile wafer strikes a good balance between manufacturing cost and performance, and hence, we use 64-CU-per-tile in experiments 2 and 3. Yet, future wafer-scale GPU design should consider the other factors (e.g., yields at particular process size) that may affect tile size selection.

A more detailed performance metrics analysis reveals that if our configurations are used, the inter-GPU communication bandwidth is not a major performance bottleneck. Instead, the HBM bandwidth and the number of hops (switch latency and switch queueing latency) are the critical limiting factors for the overall wafer performance. In MCM-GPU systems, we improve performance mainly by reducing inter-GPU traffic; but in wafer-scale GPUs, reducing hops becomes a major considering factor.

Performance vs. tile counts. Our next question is whether current workloads have kernels that utilize the whole wafers. What if we use only a portion of the wafer to run kernels?

As we analyze performance across tile sizes ranging from 1×1 to 7×7, nearly all benchmarks show improvement before reaching 3×3 (see Figure 4). This improvement is due to increased compute resources and memory bandwidth associated with larger tile counts. For instance, since 3×3 tiles provide nine times the resources of 1×1 tiles, some benchmarks (e.g., FFT, I2C, PR) achieve nearly a ninefold performance boost, while others (e.g., FIR, ReLU, SC) exhibit a speedup of 5-9 times.

However, most kernels' performance improvement plateaus when using tile sizes larger than 3×3, revealing the limited scalability of these workloads. The increase in communication costs offsets the gains in computation capacity.

A few outlier benchmarks (e.g., MT, SPMV, and S2D) only scale their performance up to 2×2 tiles or fail to scale entirely (S2D). It is no coincidence that these benchmarks align with those that consistently benefit from larger tiles (see Figure 3). Their substantial inter-GPU communication demands prevent them from leveraging additional computing resources effectively.

This experiment reveals a unique challenge for wafer-scale GPUs: strategically using fewer resources can enhance performance. Moreover, there isn't a one-size-fits-all solution. The optimal approach depends on the specific benchmark, algorithm, kernel implementation, and input size, which indicates the necessity of system-level scheduling to fully utilize the power of wafer-scale GPUs.

Performance vs. pipeline organization. As demonstrated above, kernels cannot scale to the whole wafer. Given a workload with multiple kernels, a natural solution is to implement a software-level pipeline that allows the kernels to execute concurrently on different regions of the wafer. However, exhaustively searching all possible zoning and pipelining possibilities is a finite but NP-hard problem. We cannot try all combinations of tiles assignments. Therefore, future designs need to find proper heuristics for proper zoning and pipelining. In the third experiment, we consider four different configurations and compare their performance using the whole wafer as a single GPU (unified GPU). Using the experiment, we demonstrate how different zoning and pipelining schemes impact performance.

For most pipeline configurations (CF1, CF2, CF3), we observe that performance can significantly improve over the unified GPU configuration. For example, CF1 improves performance by 1.24×. While our manual performance improvement attempts (CF2, CF3) fail to improve performance over the naive design (CF1), a bad design (CF4) can significantly slow down the performance by 0.59×. This experiment reveals that a proper pipeline organization can significantly impact the performance of a wafer-scale GPU, which is a problem not seen in traditional multi-GPU or MCM-GPU systems.

4 Discussion

Wafer-scale GPU is a promising method for meeting the ever-growing demand for computing power. Massive computing tasks,

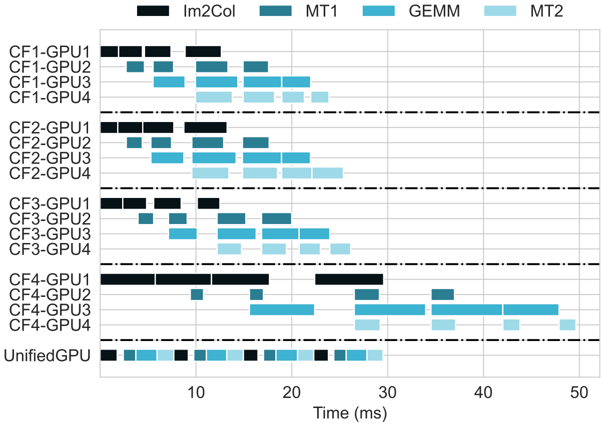

Figure 5: Simulation results for 2D convolution with different pipeline configuration.

such as Large Language Model training and large-scale scientific computing applications, urgently demand unprecedented parallel computing powers. Although traditional multi-GPU systems provide massive farms of GPU chips for enough computational power across racks of servers, the low bandwidth and high communication latency still significantly hinder the efficient leverage of the computational power. Thanks to the emerging package-level network technologies, it is now possible to connect multiple GPU chips on an interposer, which scales up to whole wafers, addressing the drawbacks of traditional multi-GPU systems.

Compared with MCM GPUs or other solutions, Wafer-scale GPU has advantages. In traditional multi-GPU systems, the communication between GPUs needs to pass the off-chip networks (e.g., PCIe, NVLink), and the inter-GPU communication becomes very expensive. The package-level network technology converts the majority of inter-GPU communication to intra-GPU communication. Wafer-scale GPU also brings large-size on-chip memory. Compared with traditional GPUs, wafer-scale GPUs have a hundred times larger on-chip memory size compared with MCM-GPUs. This gives wafer-scale GPU enough space to place data each time and reduces communication with external memory storage. All of them save communication costs and speed up computation.

The wafer-scale GPU possesses distinctive features, demanding reconsideration of the GPU architecture, system, and software design. All the challenges are rooted in the nature of the large scale of wafer-scale GPUs. Below, we list unaddressed wafer-scale GPU research challenges at each level.

Architecture-level challenges.

- As seen in Section 2.1, what should be an optimal tile size that balances computing capability and manufacturing cost?
- What degree of memory consistency and cache coherency is required and practical? How can we efficiently implement the desired level of consistency and coherency?
- How page fault and page migration can be handled in such a massive device to reduce communication and latency?
- Should we have the same tiles or slightly different tiles across the wafer? E.g., should central tiles have more memory and less computing power?
- If we can integrate other devices (e.g., CPUs, FPGAs, ASICs) on the wafer, what and where should we integrate?
- How can we tailor the package-level network to support the massive, long-distance communication requirement? Is optical communication a good solution? If so, how should electric and optical communication collaborate?

System-level challenges.

- As demonstrated in Section 2.2, how can we estimate the performance of the application if a certain number of tiles are assigned? Given a computing job, how can we allocate the optimal number of tiles (and the right shape)?
- Where should we place the compute thread and memory pages to minimize data movement? How to maintain balanced computational load across CUs while adhering to data locality constraints?
- If we assume the wafer-scale GPU is too large to be used by one user, how do we schedule users' tasks on the tiles to maximize throughput while balancing fairness and service-level objectives (SLOs)?
- If we allow multiple users to share the wafer-scale GPU, how can we ensure their jobs are properly isolated and there are no security vulnerabilities?

Software-level challenges.

- What are the representative applications for wafer-scale GPUs? A new benchmark suite may be required.
- Automatic code partitioning, scheduling, and memory placement optimizations become significantly more complex at the wafer scale. Compilers must be aware of tile locations, interconnect constraints, and potential faults.
- How to design a user-friendly programming model? Programming GPU has already been a challenge for programmers because programmers must consider how computing tasks map to hardware resources. Wafer-scale GPUs create many more possibilities and may require more effort for programmers in planning. Therefore, a tailored, user-friendly programming model (e.g., pipeline-based model) for wafer-scale GPUs.

5 Conclusion

Wafer-scale GPU technology is a promising solution to the multi-GPU performance scalability issue, as it replaces the slow off-chip interconnect with the high-speed on-wafer network. However, to harness the power of wafer-scale GPUs, more architecture-level research is needed.

This paper highlighted potential research directions related to wafer-scale GPU system design, including 1) hardware organization, 2) system-level schedule, and 3) software-level adaptation. By conducting simulation-based architecture design space exploration using three experiments, we concluded that wafer-scale GPU design has unique challenges that cannot be addressed with solutions tailored for traditional multi-GPU systems or MCM-GPU devices.

Acknowledgments

We would like to thank our anonymous reviewers for their valuable feedback. This work was partially funded by AMD and NSF grant OAC-2402947 and CCF-2402804.

References

[1] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans. 2017. MCM-GPU: Multi-chip-module GPUs for continued performance scalability. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 320–332.

[2] Jeffrey Dean. 2016. Large-scale deep learning for building intelligent computer systems. (2016).

[3] Lieven Eeckhout. 2024. RIP Geomean Speedup Use Equal-Work (Or Equal-Time) Harmonic Mean Speedup Instead. *IEEE Computer Architecture Letters* (2024).

[4] Yuan Feng and Hyeran Jeon. 2023. Understanding Scalability of Multi-GPU Systems. In *Proceedings of the 15th Workshop on General Purpose Processing Using GPU*. 36–37.

[5] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[6] Mandy La and Andrew Chien. 2020. Cerebras Systems: Journey to the Wafer-Scale Engine. *University of Chicago, Tech. Rep* (2020).

[7] Gabriel H. Loh and Raja Swaminathan. 2023. The Next Era for Chiplet Innovation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6. https://doi.org/10.23919/DATE56975.2023.10137172

[8] Saptadeep Pal, Daniel Petrisko, Matthew Tomei, Puneet Gupta, Subramanian S Iyer, and Rakesh Kumar. 2019. Architecting waferscale processors-a gpu case study. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 250–263.

[9] Guglielmo Rossi, Filippo Catani, Lorenzo Leoni, Samuele Segoni, and Veronica Tofani. 2013. HIRESSS: a physically based slope stability simulator for HPC applications. *Natural Hazards and Earth System Sciences* 13, 1 (2013), 151–166.

[10] Yifan Sun, Trinayan Baruah, Saiful A Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, et al. 2019. MGPUSim: enabling multi-GPU performance modeling and optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*. 197–209.

[11] James Wang. March 12, 2024. Cerebras CS-3: the world's fastest and most scalable AI accelerator. https://www.cerebras.net/blog/cerebras-cs3 Accessed on Month Day, Year.

[12] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* (2024), 100211.

[13] Qixiang Zhang, Liangzhen Lai, Mark Gottscho, and Puneet Gupta. 2016. Multi-story power distribution networks for GPUs. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 451–456.