Beyond Linear Scaling for LLM Training on Wafer-Scale GPUs

Qijun Zhang[1], Jingchen Zhu[2†], Chen Zhang[3], Zixiao Chen[3], Yiqi Chen[4], Mengming Li[1],
Guangyu Sun[4], Cheng Zhang[2], Zhe Zhou[2*], Zhiyao Xie[1*]
Hong Kong University of Science and Technology[1], Huawei Technologies Co. Ltd.[2],
Shanghai Jiao Tong University[3], Peking University[4]
{qzhangcs, mengming.li}@connect.ust.hk, {zhujingchen2, eric.zhangcheng, zhouzhe22}@huawei.com,
{chenzhang.sjtu, chen_zx}@sjtu.edu.cn, {yiqi.chen, gsun}@pku.edu.cn, eezhiyao@ust.hk

***Abstract*—Wafer-scale GPUs integrate many dies on a single substrate to break the reticle limit for LLM training. Current systems adopt distributed execution that achieves at best linear scaling, but linear scaling is suboptimal: ideal single-GPU performance scaling, e.g., attainable FLOP per second relative to the hardware resources, is beyond linear, as proportional resource scaling creates memory-system redundancy. The root cause preventing this is *L2 isolation*, where distributed execution confines each die to its local L2.**

We propose HALO, a hardware-software co-design framework that breaks L2 isolation through architecturally managed SM-direct remote loads, enabling practical cross-die L2 sharing at wafer scale. *Remote-access scheduling* reduces communication overhead by confining traffic within topology-aligned regions and partitioning wafer into L2-sharing domains. *Speculative pipelining* hides the elevated remote-access latency by deferring shared-memory allocation until data arrives, allowing deep software pipelining beyond physical shared-memory capacity. Evaluated on a cycle-accurate simulator with diverse LLM workloads, HALO achieves 1.24× average speedup over state-of-the-art distributed execution, reaching 98% of the theoretical beyond-linear bound, at <0.02% die-area overhead.

***Index Terms*—GPU, Wafer-scale Chip, LLM Acceleration.**

I. INTRODUCTION

The rapid growth of large language models (LLMs) demands ever-growing computation, memory, and bandwidth from training systems [46], [4], [24], [3]. Yet scaling a monolithic GPU is increasingly constrained by the *reticle limit*, the field-size limitation of lithography steppers that caps a single die at 858 mm^2 [29]. State-of-the-art GPU dies such as NVIDIA H100 [31] and B200 [32] already approach this limit, leaving little room for single-die expansion. Recent progress in chiplet design [39], die-to-die interconnects [45], and advanced packaging [25] pushes chip scale-up beyond this barrier. As the ultimate extension, wafer-scale GPUs [34], [22] integrate many reticle-limited dies on a single substrate, offering higher compute density, greater inter-die bandwidth, and lower latency than conventional multi-GPU systems, making them a compelling platform for LLM training [48], [47], [54], [50], [43].

The most natural approach to a wafer-scale GPU is transparent **single-GPU execution** [34]: exposing the entire wafer as one large GPU so that performance scales without software awareness of the multi-die structure. However, as the die count grows to tens or more, global all-to-all accesses incur prohibitive inter-die communication overhead, reducing compute utilization to below 30%, as confirmed in Section IV-B1. Therefore, current systems retreat to **distributed execution** [48], [47], [54], [50], [43], treating the wafer as a tightly integrated multi-GPU system. Through communication-centric optimizations such as collective operators and compute-communication fusion, distributed execution achieves near-linear scaling: When resources, including total die area and HBM bandwidth, increase to $N\times$, performance also improves by $N\times$. In the multi-GPU literature, linear scaling is widely regarded as optimal.

However, we argue that linear scaling is actually a suboptimal target: it is optimal when additional dies are treated as several independent GPU instances, but not when scaling up one ideal single GPU. An ideal single GPU should exhibit beyond-linear scaling, i.e., performance can improve by more than $N\times$ when resources scale up by $N\times$, as shown in

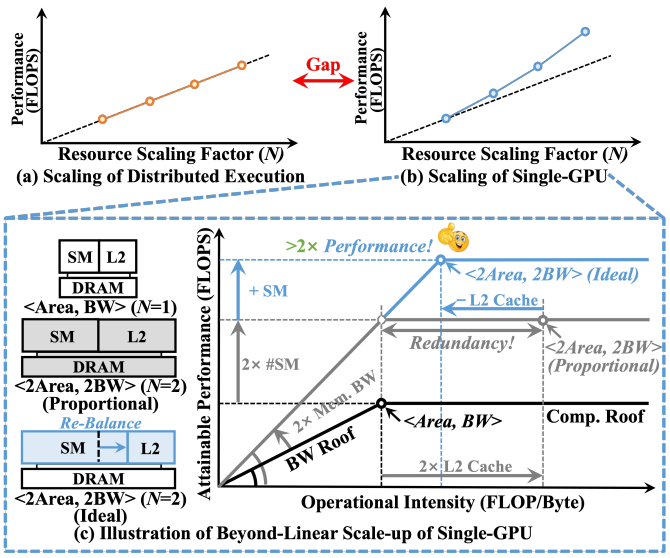

Fig. 1. Gap between linear scaling and ideal beyond-linear scaling. (a) Distributed execution scales linearly. (b) An ideal single GPU scales beyond linear. (c) Roofline explanation of (b) for N=2: proportional resource scaling creates memory-system redundancy, while an ideal single GPU reallocates area of redundant L2 to SMs, achieving $> 2\times$ performance.

† Project leader.
* Corresponding authors.

Figure 1(b). Roofline model in Figure 1(c) explains why the ideal single-GPU scale-up in Figure 1(b) can be beyond linear, using an $N{=}2$ example. When die area and DRAM bandwidth are doubled, proportionally doubling SMs and L2 can have a $2\times$ performance. However, the doubled L2 also raises operational intensity past the balance point, pushing the design into compute-bound regime and exposing *memory-system redundancy*. An ideal single GPU can therefore reduce redundant L2 capacity and reallocate freed area to additional SMs, increasing compute roof beyond the proportional $2\times$ design and achieving *beyond-linear* scaling. This rebalance is illustrated as "$-$L2 Cache" and "$+$SM" relative to the proportional scaling point, yielding more than $2\times$ performance.

The root cause of this gap between linear scaling of distributed execution and ideal performance scaling is *L2 isolation* induced by communication-centric philosophy of distributed execution. Exploiting a remote die's L2 for data reuse requires repeatedly accessing the same remote data, which directly conflicts with the goal of minimizing inter-die communication. Distributed execution therefore confines each die to its own local L2. This pins per-die operational intensity at the single-die balance point and prevents beyond-linear scaling even as total resources scale. On conventional multi-GPU systems, this isolation is inescapable: their limited inter-chip bandwidth cannot support repeated remote accesses that L2 sharing demands. Wafer-scale GPUs, with their substantially richer inter-die fabric, create a new opportunity to break this isolation. Section II substantiates these observations with detailed roofline analyses.

The goal of this work is to break L2 isolation and achieve beyond-linear scaling. While distributed execution treats remote access as overhead to be minimized, we take the opposite view: remote access is not the problem but the opportunity. Our **key insight** is that *the remote accesses that distributed execution strives to eliminate, when architecturally managed through dedicated orchestration, are exactly the key to unlocking beyond-linear scaling*. When multiple SMs on different dies repeatedly access the same data through SM-direct remote loads, every access is served by the one L2 slice where that data reside, achieving cross-die L2 sharing. Effective L2 capacity for data reuse thus spans all participating dies, breaking L2 isolation and enabling beyond-linear scaling. This reframes remote loads from a communication overhead to be eliminated into the critical primitive to enable L2 sharing. Single-GPU execution also performs remote loads, but without architectural management the resulting traffic overwhelms interconnect.

Architecturally managing SM-direct remote loads is non-trivial, with two critical challenges: ① Architecture-unaware task scheduling and data layout cause all-to-all remote traffic with high volume, high latency, and poor load balance. ② Existing GPU latency-hiding mechanisms cannot tolerate the long and diverse remote-access latencies at wafer scale, severely degrading compute utilization. To address these challenges, we propose HALO (**H**ardware-**A**ware **L**oad **O**rchestration), a hardware-software co-design framework that makes SM-direct remote loads efficient for practical L2 sharing at wafer scale. ❶ *Remote-access scheduling* at both operator and system levels reduces traffic volume, latency, and imbalance through topology-aware data placement, complemented by lightweight hardware support. ❷ *Speculative pipelining* defers shared-memory allocation to eliminate redundant occupation. This creates sufficient pipeline depth to hide long remote-access latency.

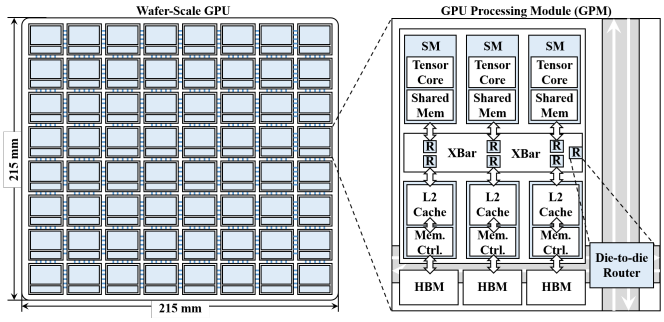

Fig. 2. Organization of a wafer-scale GPU. 64 GPMs are interconnected in an $8{\times}8$ 2D mesh on a single wafer substrate.

In summary, this paper makes the following contributions:

- We analyze existing wafer-scale LLM training solutions, reveal the gap between their linear scaling and ideal beyond-linear scaling, identifying L2 isolation as the root cause.
- We propose HALO, a hardware-software co-design framework for wafer-scale GPU LLM training that makes SM-direct remote loads efficient for L2 sharing. HALO introduces remote-access scheduling to reduce communication overhead, along with speculative pipelining to hide long-latency remote accesses.
- Evaluated on cycle-accurate simulator with diverse LLMs, HALO achieves $1.24\times$ average speedup over state-of-the-art distributed execution, reaching 98% of theoretical beyond-linear bound.

II. Background and Motivation

A. Wafer-Scale GPUs

Enabled by advanced packaging technologies such as TSV [6], [14], [19], CoW [10], [11], [52], and RDL [8], [9], [20], [21], wafer-scale chips have emerged as a promising scale-up solution [22], [42], [54], [50], [43], [48], [47], [34], [49]. Wafer-scale integration mainly follows two routes. Monolithic designs, e.g., Cerebras [22], fabricate compute and SRAM on one wafer-scale substrate but cannot place HBM stacks beside each die. Chiplet-based integration [42], [40], [34] instead assembles known-good dies on a wafer-level substrate, improving yield and enabling co-packaged HBM. This route is practical: wafer-level GPU prototype [34] and TSMC SoW-X [40] integrates dies with HBM. Following prior architectural studies [48], [47], [50], [34], [49], [43], we adopt an HBM-equipped chiplet organization. It improves flexibility and yield while providing the memory capacity required by LLMs, addressing capacity limitations of SRAM-only design. This is also consistent with TSMC SoW-X direction [40].

As shown in Figure 2, we model a wafer-scale GPU as an 8×8 2D mesh of 64 GPU processing modules

(GPMs) on a $215\,\mathrm{mm} \times 215\,\mathrm{mm}$ wafer substrate. Advanced packaging provides 5 TB/s per-link die-to-die bandwidth, as achieved in CoWoS-based production multi-die GPUs [32], [44] and scaled to wafer scale like SoW-X [40], with $\sim$100 ns latency [42]. This is a substantial improvement over conventional chip-to-chip interconnects [31], [33]. Each GPM contains one compute die and HBM stacks. The compute die uses a NoC to connect SMs, memory partitions with L2 caches and memory controllers, and die-to-die routers for cross-die memory accesses. This organization matches prior wafer-scale and multi-die GPU designs [32], [1], [34], [49]. It can be programmed either transparently as a single GPU or with distributed execution, where each die acts as an independent GPU and communication uses optimized communication operators as in multi-GPU systems.

B. Revisiting the Scaling of Wafer-Scale GPUs

1) Linear Scaling of Distributed Execution: The most natural approach is to program the entire wafer as a single GPU using the same interface as a conventional GPU, so that existing programs transparently scale without awareness of the multi-die structure. NVIDIA B200 adopts this approach at the two-die scale: the address space is interleaved across both dies, and kernels launch over the entire GPU without software awareness of where each thread block executes or where data reside [32]. With 10 TB/s NV-HBI, communication is not a bottleneck. At wafer scale, however, the dramatic increase in die count fundamentally changes the situation. Global all-to-all accesses now span tens of dies, causing die-to-die traffic to surge. On a mesh topology with limited bisection bandwidth, even the high-bandwidth wafer fabric cannot prevent communication from becoming the bottleneck. The larger die count also increases access latency substantially. Our experiments in Section IV-B1 confirm that compute utilization falls below 30%.

Although wafer-scale GPU can support shared address space and direct remote access of single-GPU execution, the prohibitive communication overhead forces LLM systems to retreat to distributed execution in practice. The system is instead treated as a tightly integrated multi-GPU system, where communication-centric optimizations such as collectives and compute-communication fusion substantially reduce overhead. On our wafer-scale GPU (configuration in Section IV-A), communication time is less than 10% of compute and is effectively hidden, enabling linear scaling: when resources, including total die area and HBM bandwidth, increase to $N\times$, performance also improves by $N\times$. In multi-GPU LLM literature, linear scaling is widely regarded as the optimum.

2) Beyond-Linear Scaling Potential: In fact, such linear scaling is only the optimum of treating additional dies as independent GPU instances, rather than the optimum of ideal single-GPU scale-up. Under distributed execution, each die performs majority of its computation locally, only inputs/outputs are exchanged with communication. Consequently, if a single die is configured at the *sweet spot*, a multi-die GPU under distributed execution remains at the same per-die sweet spot: each die still executes the same local computation independently. Therefore, distributed execution behaves as multiple independent instances, whose best-case performance scales linearly with die count, as shown in Figure 1(a).

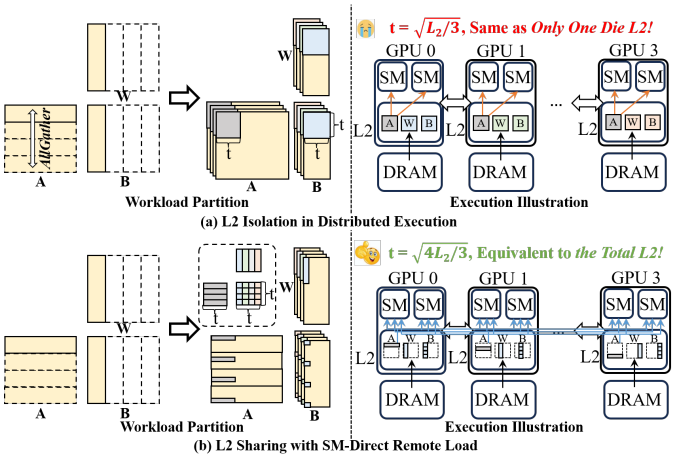

Fig. 3. L2 isolation vs. L2 sharing, illustrated with a GEMM example. (a) Under distributed execution, each die exploits only its local L2 for data reuse, confining the effective L2 capacity to a single die. (b) With SM-direct remote loads, SMs load directly from remote L2-DRAM, extending effective L2 capacity across all dies. Local W and B accesses are omitted.

However, an ideal single GPU should exhibit *beyond-linear* scaling: increasing resources, including total die area and HBM bandwidth, to $N\times$ can yield more than $N\times$ performance, as illustrated in Figure 1(b). Roofline model in Figure 1(c) makes this point clear. Consider a single-die GPU with an SM-L2-DRAM configuration that lies at the sweet spot, i.e., at the intersection of the compute roof and the memory-bandwidth roof. When die area and DRAM bandwidth are doubled, if SM count and L2 capacity are also doubled proportionally, performance also doubles, and the balance point remains at the same operational intensity. However, the enlarged L2 shifts the effective operating point to the right, transforming the scaled GPU into a compute-bound design. In other words, *memory-system redundancy* emerges. To achieve ideal scaling, the GPU must rebalance compute and memory resources by reducing L2 cache capacity ("$-$L2 Cache") and reallocating the freed area to additional SMs ("+SM") so as to return to the sweet spot. Therefore, under ideal single-GPU scaling, doubling die area and DRAM bandwidth achieves $>2\times$ performance, i.e., ideal single-GPU scaling is beyond linear. This indicates a gap between the linear scaling of distributed execution and the ideal beyond-linear scaling.

3) L2 Isolation: The Root Cause: The root cause of this gap is *L2 isolation* induced by the communication-centric philosophy of distributed execution. Reducing communication is the primary objective, which fundamentally precludes exploiting remote L2 caches for data reuse: reusing a remote L2 requires repeatedly accessing the same remote data. Consider AllGather-GEMM as an example in Figure 3(a). With separate collective and compute operators, remote data are first copied into local DRAM and then repeatedly read, so each die exploits only its own L2 for data reuse. Some compute-communication fused kernels alter the dataflow so that matrix A is loaded

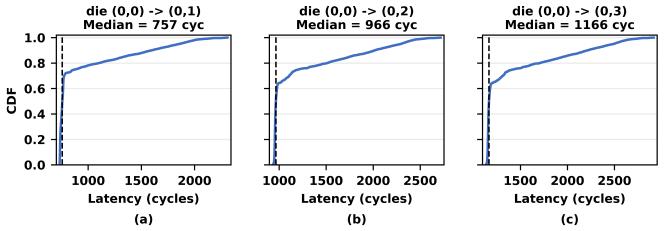

Fig. 4. CDF of remote-access latency for 1-, 2-, and 3-hop transactions.

only once into the SM and reused as each TB (Thread Block) iterates along the remaining dimension. However, this shifts the repeated access to matrix W and B, which still exploit only local L2 caches. This merely changes *which* data are repeatedly accessed without breaking the isolation itself. Under L2 isolation, when SM count, L2 capacity, and DRAM bandwidth all scale by N, arithmetic intensity cannot rise. The scaled GPU remains pinned at the balance point and cannot release L2 area to SMs. Beyond-linear scaling is therefore unachievable under distributed execution.

C. Design Philosophy and Challenges

The above analysis identifies L2 isolation as the architectural root cause of the gap to beyond-linear scaling. On conventional multi-GPU systems, this isolation is inescapable: their limited inter-chip bandwidth cannot sustain the repeated remote accesses that L2 sharing demands. Wafer-scale GPUs, with their substantially richer inter-die fabric, create a new opportunity. Our design philosophy is to *embrace remote access rather than eliminate it, and architecturally manage the resulting traffic through dedicated architectural orchestration to unlock beyond-linear scaling*. As shown in Figure 3(b), when multiple SMs on different dies repeatedly access the same data through SM-direct remote loads, every access is served by the one L2 slice where that data reside, achieving cross-die L2 sharing. Effective L2 capacity for data reuse thus spans all participating dies, breaking L2 isolation and enabling beyond-linear scaling. Single-GPU execution also performs remote loads, but without architectural management the resulting traffic overwhelms interconnect.

Making SM-direct remote loads efficient for practical L2 sharing is non-trivial. Two challenges must be addressed to achieve architecturally managed remote access:

C1: Architecture-unaware scheduling introduces excessive communication overhead. Without topology and workload awareness, scheduling and data layout create high-volume, high-latency, poorly balanced global all-to-all traffic. For example, global address interleaving [30], [31], [32] spreads accesses across memory partitions on all dies, causing long-distance traffic that occupies many links, making bandwidth the bottleneck and incurring high latency. It also fails to align with execution patterns, leading to imbalanced data transfer and memory accesses. HALO addresses this with operator- and system-level *remote-access scheduling* that aligns data placement and scheduling with compute and topology.

C2: Existing latency-hiding mechanisms cannot tolerate long remote-access latency. SM-direct remote loads across an L2-sharing domain of dimension d traverse $2(d-1)$ die-to-die links at $\sim$100 ns each. For d=4, this adds 600 ns beyond the $\sim$500 ns local round trip. Figure 4 shows median remote latency of 757–1166 cycles for 1–3 hops, with 2500–3000-cycle tails from queuing. Although GPUs hide memory latency via multi-stage software pipelining, each stage reserves a shared-memory tile, limiting depth. Our experiments show that it requires at least 12 stages for >98% compute utilization, but H100 shared memory supports only 7 stages, yielding merely 68.5% utilization. HALO addresses this with *speculative pipelining*, which defers shared-memory allocation to provide enough pipeline depth for long remote latencies.

III. HALO DESIGN

A. Overview

As established in Section II, architecturally managed remote access is the key to making SM-direct remote loads efficient for practical L2 sharing at wafer scale. Our proposed HALO addresses the two challenges identified in Section II through dedicated hardware-software co-design:

- To address **C1** (communication overhead), HALO introduces *remote-access scheduling*, as detailed in Section III-B, which manages remote-access traffic through compute-aligned interleaving at operator level and domain-partitioned execution at system level.
- To address **C2** (latency hiding), HALO introduces *speculative pipelining*, as detailed in Section III-C, an architectural design that employs on-demand shared-memory allocation to create sufficient pipeline depth for hiding the long remote memory access latency at wafer scale.

B. Remote-Access Scheduling

Our proposed remote-access scheduling addresses C1 by managing SM-direct remote loads through operator-level data placement and system-level partitioning.

1) Insight: Compute-Topology Co-Alignment: Reducing communication overhead of SM-direct remote loads requires jointly optimizing traffic volume, latency, and load balance. Global spreading causes excessive traffic volume and long latency, while simple partitioning reduces them but creates imbalance. The key observation is that operators in LLM training exhibit regular, tile-based access patterns. It allows communication traffic to be *managed* by aligning data placement with both compute pattern and mesh topology, achieving low traffic volume, short latency, and balanced load simultaneously. Remote-access scheduling exploits this at two levels: *compute-aligned interleaving* confines and balances tile-load traffic at operator level, while *domain-partitioned execution* limits communication scope by partitioning wafer into L2-sharing domains.

2) Compute-Aligned Interleaving: We characterize the communication cost of SM-direct remote loads in GEMM along the three dimensions below:

- **Maximum link load** $load_{max}$: the maximum amount of data transmitted on any single link, with the total memory-

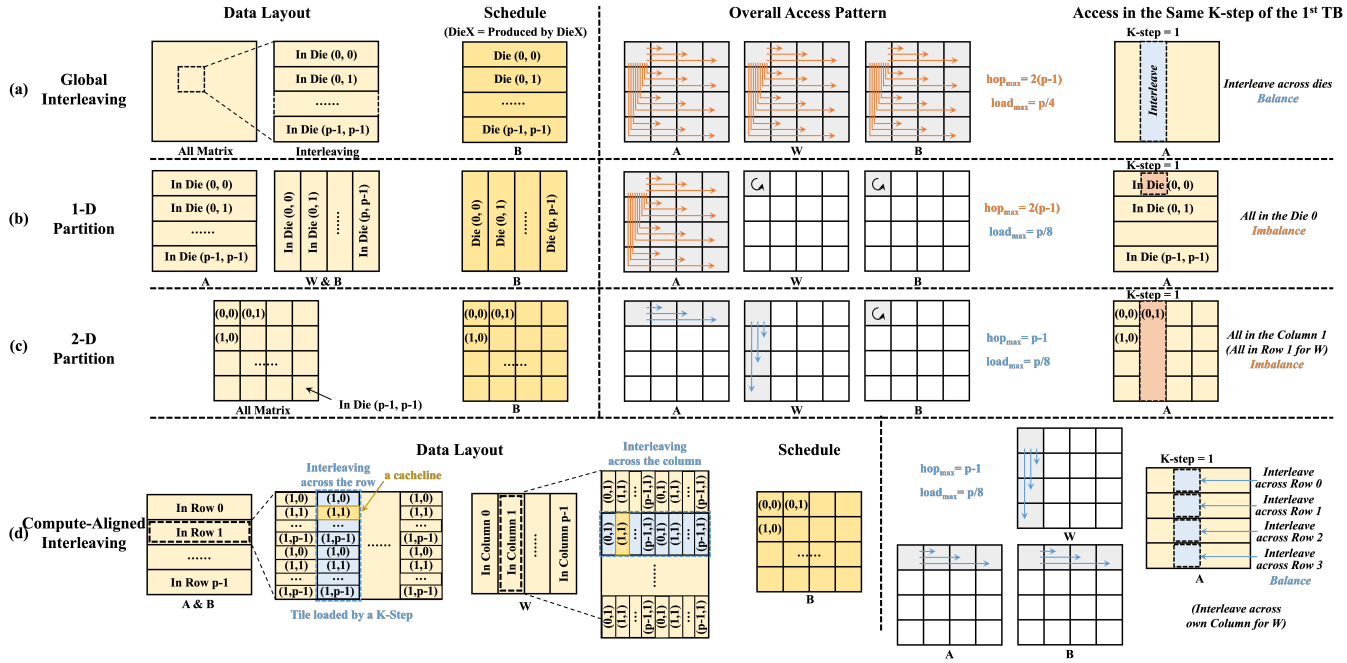

Fig. 5. Derivation of our proposed compute-aligned interleaving, evaluated by three dimensions, including maximum link load ($load_{max}$), maximum hop count (hop_{max}), and access balance. (a) Global interleaving spreads all accesses across all dies: balanced but with excessive traffic volume ($load_{max}$=$p/4$) and latency (hop_{max}=$2(p-1)$). (b) 1-D partition halves traffic volume but retains full latency and introduces severe imbalance. (c) 2-D partition reduces latency to $p-1$ but still suffers from imbalance. (d) Compute-aligned interleaving achieves low traffic volume, short latency, and balance simultaneously by fine-grained interleaving within a topology-confined region.

access volume of one die normalized to 1. This captures *traffic volume*.

- **Maximum hop count** hop_{max}: the longest path traversed by any remote memory access. This captures *latency*.
- **Access balance**: uniformity of memory accesses from TBs with the same index on each die at the same *K-step*. A *K-step* denotes an iteration over the tiled GEMM reduction (K) dimension, where a TB loads the corresponding A and W tiles and accumulates their product into its output tile B. This captures *load imbalance*.

These objectives appear to conflict: global interleaving achieves balance but maximizes traffic volume (e.g., $load_{max}$) and latency, while partition reduces traffic volume and latency but introduces imbalance. We show that all three can be simultaneously achieved by *aligning fine-grained interleaving with compute access pattern within a topology-confined region*. We arrive at this design through a systematic exploration of data-layout design space shown in Figure 5.

Global Interleaving. Global sub-page interleaving, as in NVIDIA GPUs [30], [31], [32], implicitly balances load but spreads all traffic across all dies, as shown in Figure 5(a). For a $p \times p$ mesh with uniform all-to-all accesses, $load_{max}$ is $p/4$, and hop_{max} is $2(p-1)$, corresponding to diagonal accesses. $p/4$ comes from the central bisection. Each source-destination pair carries normalized traffic $1/p^2$. Across the central bisection, $p^2/2$ source dies communicate with $p^2/2$ destination dies on the other side, giving one-direction bisection traffic $(p^2/2)(p^2/2)(1/p^2) = p^2/4$. This traffic is shared by p links across the bisection, yielding $load_{max} = p/4$. Because of globally interleaved address, accesses are implicitly balanced [26]. However, severe traffic volume ($load_{max} = p/4$) and high latency ($hop_{max} = 2(p-1)$) make this impractical at wafer scale.

1-D Partition. To reduce communication cost, we first consider a partition inspired by tensor parallelism: 1-D partition localizes one input operand and output, so only *half* of memory accesses are remote, halving $load_{max}$ to $p/8$ as shown in Figure 5(b). However, hop_{max} remains $2(p-1)$ because remote accesses still span all dies, so long latency persists. It also creates imbalance: at *K-step*=1 for the first TB, all dies access the same first-row tile of matrix A, concentrating all remote accesses on die $(0,0)$.

2-D Partition. The failure of 1-D partition to reduce latency stems from its inability to avoid long-distance accesses such as diagonal ones. To address this, we narrow the access-spreading region by partitioning both operands in two dimensions, as in Figure 5(c). Each die accesses only dies in the same row and column, reducing hop_{max} to $p-1$. This narrowing also reduces the number of traversed links, maintaining $load_{max}$ at $p/8$. Unlike prior 2-D distributed execution, which localizes computation to avoid remote access, our design uses SM-direct remote loads to enable cross-die L2 sharing. However, this coarse-grained partition still suffers from imbalance: at *K-step*=1, accesses to matrix A all fall on the first-column dies, while accesses to W all fall on the first-row dies.

Compute-Aligned Interleaving. The analysis above reveals that coarse-grained partitioning inherently causes imbalance: achieving balance requires distributing accesses within each tile load evenly, as global interleaving does, but within a confined region. Our compute-aligned interleaving achieves

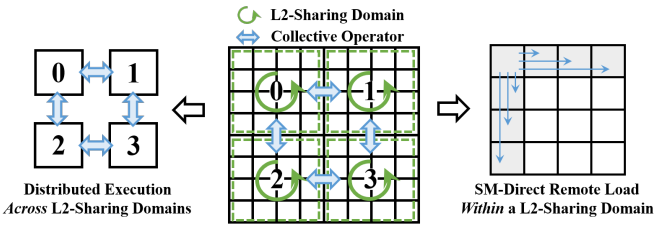

Fig. 6. Domain-partitioned execution partitions the wafer into L2-sharing domains (four 4×4 domains shown).

precisely this, as shown in Figure 5(d). Unlike coarse-grained 2-D partitioning, we interleave rows $[(M/p)i, (M/p)(i{+}1))$ of matrix A across the dies in the i-th die row at row granularity, so that all memory accesses corresponding to a tile load are evenly distributed across all dies in that row. Similarly, matrix W is interleaved across a die column at column granularity. The output matrix B adopts the same layout as A to ensure the subsequent GEMM receives correctly laid-out input. The layout of B does not materially affect communication cost, because it is written only once and this cost is negligible compared with the repeated accesses to A and W. Under this layout, $load_{max}$ and hop_{max} remain halved as in 2-D partition, while the fine-grained, compute-aligned interleaving ensures that each tile load is evenly distributed across the dies in a row or column, achieving load balance.

For FlashAttention, Q, K, V, and O all follow this layout. Within each L2-sharing domain, computation is scheduled head by head to maximize L2 sharing, since different heads share no data. Non-GEMM operators such as LayerNorm and Softmax have little data reuse, so each die processes its local partition with local accesses. Their small runtime does not reduce the overall benefit. In the backward, GEMM operand roles are transposed, so we insert a lightweight layout transformation between forward and backward. It moves each element once, takes less than 2% of total time, and is outweighed by the benefit of compute-aligned interleaving.

3) Domain-Partitioned Execution: Compute-aligned interleaving optimizes the communication pattern at the operator level, but when the L2-sharing domain spans the entire wafer, the absolute communication overhead remains non-trivial due to the large number of dies. To further reduce this overhead, we introduce domain-partitioned execution, a system-level scheduling strategy that limits the scope of L2 sharing to a subregion of the wafer.

Execution Model. As shown in Figure 6, domain-partitioned execution divides the wafer into L2-sharing domains. Within each domain, SM-direct remote loads with compute-aligned interleaving enable cross-die L2 sharing. Across domains, distributed execution performs only communication-light data exchanges. This reduces $load_{max}$ and hop_{max} from $p/8$ and $p{-}1$ to $d/8$ and $d{-}1$, where d is the domain dimension, trading a modest L2-sharing scope reduction for much lower communication overhead.

Data Exchange Traffic. Cross-domain transfers occur only between dies with the same in-domain position. In Figure 6, die $(0,0)$ exchanges A only with die $(0,4)$ and W only with die $(4,0)$. The exchanged data preserves compute-aligned interleaving, so intra-domain execution is unchanged. Computation within a domain and communication across domains are overlapped: as soon as a remote chunk arrives, its computation starts, pipelining inter-domain transfer with intra-domain execution.

4) Architectural Support: As shown in Figure 8, our proposed remote-access scheduling requires the extension to the TMA unit for compute-aligned interleaved layouts and router extension for traffic-class isolation.

TMA Extension. Existing TMA assumes standard matrices in contiguous address spaces, while compute-aligned interleaving distributes each matrix across dies. We extend tensor descriptor of TMA with `base_per_die[d]`, the base address on each die, and `interleave_dim`, the interleaving dimension. These fields initialize a *die base address table* at kernel launch. The TMA address generator is also extended to resolve each element to its die and local address. For row-wise interleaving: For an element at position (row, col) within the loaded tile under row-wise interleaving:

$$\begin{aligned}\texttt{die\_id} &= \texttt{row} \bmod p,\\ \texttt{offset} &= \lfloor \texttt{row}/p \rfloor \times \texttt{stride} + \texttt{col},\\ \texttt{addr} &= \texttt{base\_per\_die}[\texttt{die\_id}] + \texttt{offset},\end{aligned}$$

where p is the number of dies along the interleaving dimension. Column-wise interleaving swaps row and column. Each request carries `addr` and `die_id` for routing. The extension is confined to TMA address generator without modifying the memory hierarchy.

Traffic-Class Isolation. To reduce interference between intra- and inter-domain traffic, each request carries *D-Flag* as in Figure 8. At each D2D router output port, the flag selects either *intra-queue* for L2-sharing traffic or *inter-queue* for cross-domain exchange. A weighted arbiter allocates bandwidth between them using a weight derived from traffic volumes and configured at kernel launch.

C. Speculative Pipelining

Even after remote-access scheduling reduces communication overhead, remote memory access latency still remains fundamentally elevated compared with local accesses, as characterized in Section II. If not hidden, this latency causes pipeline bubbles that severely degrade compute utilization. GPUs hide memory latency through multi-stage software pipelining, but existing mechanisms manage shared memory too conservatively, limiting pipeline depth far below what wafer-scale latencies demand. Speculative pipelining addresses C2 by *speculatively* issuing more tile-load requests than physical shared memory can simultaneously hold. Such a speculation eliminates redundant resource occupation and therefore creates sufficient pipeline depth to hide long remote-access latency.

1) Key Insight: On-Demand Allocation: The root cause of insufficient pipeline depth is the overly conservative shared-memory management in existing GPUs. As Figure 7(a) shows, the pipeline depth required to hide latency is $\lceil \text{latency}/\text{Comp} \rceil{+}1$, which rises sharply with remote latency.

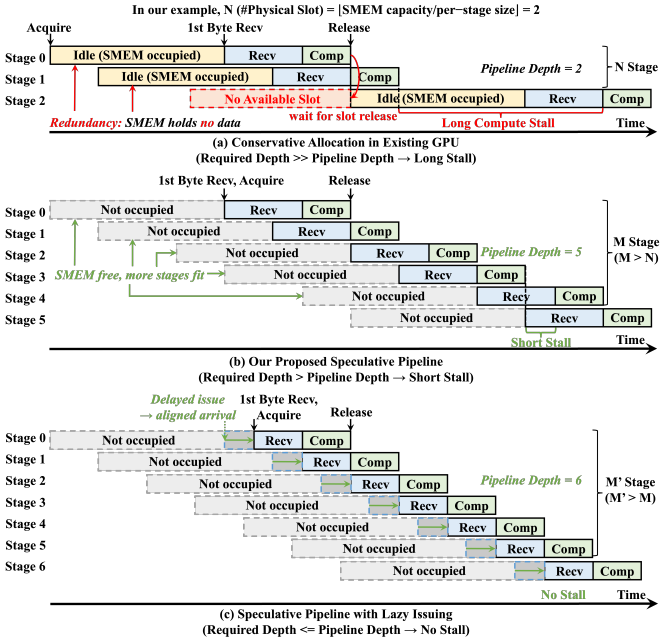

Fig. 7. Speculative pipelining illustration. (a) Conservative allocation limits pipeline depth to N, causing compute stalls. (b) On-demand allocation enables $M>N$ stages but a residual stall remains from arrival-time spread. (c) Lazy-issuing compresses the spread, eliminating stall.

In existing implementations, shared memory for a given stage must be acquired *before* the tile-load instruction is issued, but this space stays idle, holding no valid data, until the first byte returns to the SM. This issue-to-first-byte occupation is *entirely redundant* and limits effective depth under fixed capacity. On-demand allocation eliminates this redundancy by deferring the acquisition until the first byte of a tile load arrives, as in Figure 7(b). Thus, outstanding tile loads can exceed physical shared-memory capacity, enabling more stages. The approach is *speculative*: it bets earlier stages will free slots before later responses arrive. A large arrival-time spread still causes stalls, lazy-issuing in Section III-C3 mitigates this. Overflow handling in Section III-C2 preserves correctness when slots are temporarily exhausted.

2) Virtual Pipeline Stages: On-demand allocation requires the hardware to dynamically map tile-load responses to physical shared-memory slots at response arrival time. This timing is not deterministic and cannot be managed by software. We therefore introduce *virtual pipeline stages*, a hardware-managed abstraction that allows the software pipeline depth to exceed the physical shared-memory capacity. Virtual pipeline stages provide a virtual shared-memory space and virtual barriers, allowing the programmer to write multi-stage pipelines exactly as before, while the hardware transparently performs on-demand physical resource allocation.

Programming Model. The programmer specifies M virtual pipeline stages ($M \gg N$, where $N = \lfloor \text{SMEM capacity}/\text{per-stage size} \rfloor$), allocates virtual shared-memory buffers and virtual barriers for all M stages, and writes the pipelining identically to the standard case, only replacing physical shared-memory and mbarrier references with their virtual counterparts. Hardware transparently manages the mapping from virtual to physical resources on demand.

Allocation Protocol. Physical shared memory is partitioned into N fixed-size slots, each paired with a physical mbarrier. A VSHMEM table in SM manages the dynamic mapping from M virtual stages to the N physical slots. When a kernel acquires a virtual barrier, no physical resource is allocated. The VSHMEM table records the pending acquisition and the mapping from outstanding TMA requests to the virtual-stage index. When the *first response* for a virtual stage arrives at the SM, a free physical slot and its paired mbarrier are allocated, the virtual-to-physical mapping is recorded, and the response data is written to the physical slot. Subsequent responses for the same stage are routed via recorded mapping. Once all responses arrive, the physical mbarrier completes and the virtual barrier's phase bit is updated. The consumer waits on the virtual barrier, accesses data in the physical slot, and upon completion releases the virtual barrier, freeing both physical slot and mbarrier for reuse.

Overflow Recovery. Because the number of virtual stages exceeds the number of physical slots, overflow can occur: all N physical slots may be occupied when a new response arrives, leaving no buffer for the data. The VSHMEM table marks the corresponding virtual stage as *critical*, granting it priority allocation as soon as a physical slot is freed. The TMA requests for that stage are retransmitted at cacheline granularity with a high-priority flag. Deadlock cannot arise because each occupied slot is guaranteed to eventually complete: all its in-flight responses will arrive through the deadlock-free mesh network, the consumer will process the data, and the slot will be released. The freed slot is then immediately assigned to the highest-priority critical stage. This mechanism ensures forward progress and program correctness.

Warm-up. HALO throttles the pipeline depth to avoid transient overflow during warming up. A TB starts with the active pipeline depth equal to the number of physical slots, as in a conventional pipeline. Each time a stage releases a physical slot, the active pipeline depth is increased by one until reaching virtual pipeline stage M. This prevents the initial long-latency stages from being launched simultaneously before any physical slot can turn over.

3) Lazy-Issuing: To allow rapid turnover of physical slots, slot occupation time must be minimized. Occupation spans from the first-byte arrival to the consumer completion. Arrival-time spreading dominates the time: data is interleaved across dies at different distances, so individual transactions of a tile load have vastly different latencies, prolonging the interval from first- to last-byte arrival.

Lazy-issuing reduces this spread by deliberately delaying the issuance of short-latency requests so that all requests complete at approximately the same time. Each request is delayed by the difference between the static latency to the farthest die and the computed static latency of that particular request, eliminating the arrival-time spread from static-latency differences. To further align the spread caused by queuing delay, each die maintains counters that track the observed

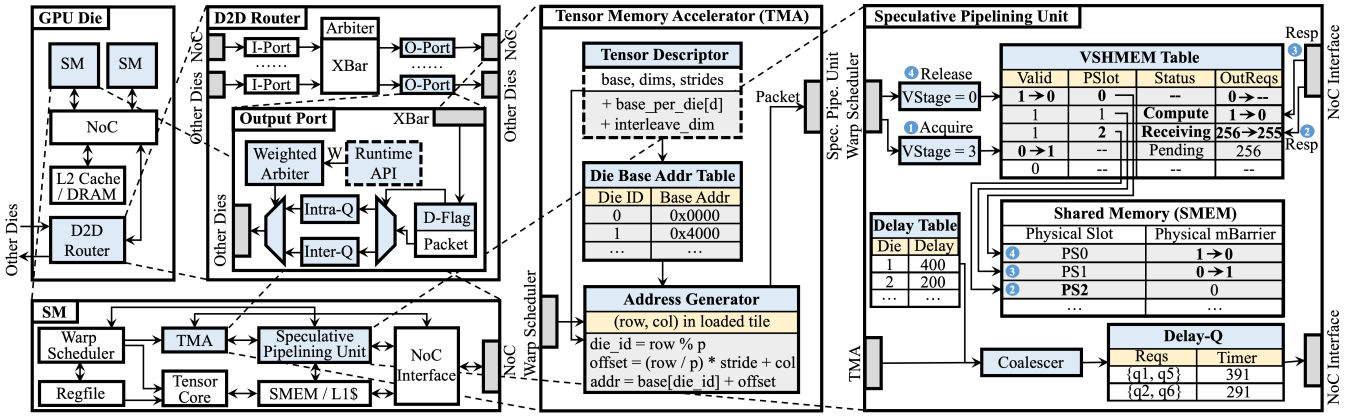

Fig. 8. Architectural design of HALO. TMA is extended with a die base address table and an interleaving-aware address generator. D2D router output port is augmented with a D-Flag and weighted arbiter. A speculative pipelining unit manages virtual-to-physical stage mapping via VSHMEM table and aligns request arrival times through lazy-issuing hardware.

access latency to all other dies. Once latency reaches a steady state, the per-request delay also accounts for the measured queuing-latency difference, further tightening alignment. By shortening per-stage occupation time, lazy-issuing enables fewer physical slots to sustain more virtual pipeline stages, eliminating the residual stall as shown in Figure 7(c).

4) Architectural Support: Our proposed speculative pipelining requires three architectural supports: ISA extensions to expose virtual pipeline stages, a VSHMEM table to manage the virtual-to-physical mapping, and lazy-issuing logic to align request arrival times.

ISA Extension. Four PTX instructions that constitute TMA-based pipelining require extension to operate on virtual addresses: `cp.async.bulk.tensor` (data transfer), `mbarrier.arrive.expect_tx` (producer acquire), `mbarrier.try_wait.parity` (producer/consumer wait), and `mbarrier.arrive` (consumer release). We add a virtual state-space qualifier, analogous to the existing `shared::cta` and `shared::cluster` qualifiers in PTX. When operating on virtual addresses, these instructions interact with the VSHMEM table rather than directly accessing physical shared memory and mbarriers.

VSHMEM Table. The VSHMEM table maintains one entry per virtual pipeline stage. Each entry stores four fields: a `Valid` bit, a physical-slot index `PSlot`, a `Status` word, and an outstanding-response counter `OutReqs`. As illustrated in Figure 8, each virtual stage follows a four-step lifecycle: ① **Acquire.** The warp scheduler acquires a virtual barrier for stage i. The VSHMEM table sets `Valid`←1, `PSlot`←unassigned, `Status`←PENDING, and `OutReqs`←the total expected response count. No physical shared memory is consumed at this point. ② **Allocate & Receive.** When the first network response for stage i arrives, the table allocates a free physical slot and its paired physical mbarrier, records the mapping in `PSlot`, and transitions `Status` to RECEIVING. Each subsequent response is routed to the mapped physical slot via an associative lookup on the virtual-stage index, and `OutReqs` is decremented. ③ **Compute.** When `OutReqs` reaches zero, all tile data have arrived. The physical mbarrier completes and signals the consumer warp. `Status` transitions to COMPUTE, and the consumer reads data directly from the physical slot. ④ **Release.** The consumer finishes and releases the virtual barrier. The table sets `Valid`←0, frees the physical slot and its mbarrier, making them available for reuse by subsequent stages. With M=12 virtual stages, the table contains only 12 entries. The associative lookup on each returning response completes in a single cycle, lies entirely off the SM's compute critical path, and is triggered asynchronously by network responses.

Overflow Handling Workflow. Overflow recovery is in cacheline granularity. The workflow is as follows: 1) When a response arrives, hardware checks its virtual-stage entry in VSHMEM table. If the entry is PENDING and no physical slot is free, the response overflows, the entry stays unmapped and is marked CRITICAL. 2) The overflow immediately triggers a retry of its request, injected into network with a high priority. 3) Once a slot is released, VSHMEM assigns it to the critical virtual stage and moves it to RECEIVING. 4) The retried response is accepted in RECEIVING; if the entry is still PENDING, it triggers another retry.

Lazy-Issuing Hardware. As shown in Figure 8, the lazy-issuing mechanism consists of delay table and delay queue. A *delay table* records the calculated delay to each destination die, which is calculated as the difference between the maximum latency across all destinations and the latency to the specific target die, as measured by dedicated latency counters. For a 4×4 domain this amounts to 16 entries of 2 bytes each. During warming up, the delay table is updated from the latency counters whenever a stage releases a physical slot. *Delay queue* buffers delayed requests after coalescing. The coalescer merges requests from the same TMA instruction and destination die when their addresses follow a stride pattern. Each entry contains a 74-bit request descriptor: `Valid` (1b), `TMAId` (5b), `DstDie` (6b), `BaseAddr` (48b), `Stride` (8b), and `Count` (6b), plus a 12-bit `Timer`. When the timer expires, the queue regenerates requests from `BaseAddr`, `Stride`, and `Count`. In our kernels, each 128-request TMA targets four destinations, thus

Wafer-Scale GPU		Per-Die Configuration		
Parameter		Parameter	Sweet-spot	HALO
Dies	64	SMs per die	64	80
Topology	8×8 mesh	FP16 (TFLOPS)	959	1199
D2D bandwidth	5 TB/s	L2 cache	50 MB	4 MB
D2D latency	100 ns	HBM bandwidth	512 GB/s	

TABLE I
HARDWARE CONFIGURATION. BOTH SWEET-SPOT DIE AND HALO DIE OCCUPY THE SAME DIE AREA.

coalescing into four entries. With $M{=}12$ virtual stages and two TMA loads per stage, the bound is $12{\times}2{\times}4 = 96$ entries per SM, totaling $96{\times}86$ bits $\approx$1KB. All structures operate asynchronously relative to the SM's warp-scheduler critical path and do not affect SM frequency or pipeline timing.

D. From L2 Sharing to Beyond-Linear Scaling

Together, remote-access scheduling and speculative pipelining make SM-direct remote loads efficient for practical L2 sharing at wafer scale. The effective L2 capacity now extends to the aggregated L2 capacity across all dies within the same L2-sharing domain, as discussed in Figure 3. Therefore, following the analysis in Section II, when the resources, including total die area and HBM bandwidth provided by multiple dies, scale to the wafer scale, the wafer-scale GPU can achieve a *beyond-linear* performance scaling. We quantify the achievable gain in Section IV.

IV. EVALUATION

A. Methodology

1) Simulation Infrastructure: We evaluate HALO using a cycle-accurate wafer-scale GPU simulator that integrates a customized Accel-Sim [17] with BookSim2 [13]. We extend Accel-Sim's Hopper model to better capture TMA and multi-stage software pipelining with asynchronous barriers. We also parallelize per-die simulation and synchronize at cycle boundaries for acceleration. Validation against H100 shows GEMM errors within 6% across diverse shapes.

2) Hardware Configuration: Table I lists hardware parameters. Our wafer-scale GPU has 64 dies in 8×8 2D mesh, with 512 GB/s HBM per die. D2D link has 5 TB/s bandwidth with 100 ns latency.

Sweet-spot die. Due to area constraint at wafer scale, each GPU die cannot replicate the full H100 design. We therefore adopt a training-optimized die that doubles the tensor-core count per SM while reducing the total SM count to 64, achieving comparable peak throughput (959 vs. 989 TFLOPS in FP16) in a smaller area. This design trend of maximizing datapath density while reducing per-SM control overhead aligns with the industry-wide shift toward larger execution units in training-optimized accelerators. L2 cache is set to 50 MB, occupying approximately 23% of total die area, preserving the original compute-to-L2 ratio of H100. This die operates near *sweet spot* of roofline model. Distributed execution baselines all use wafer-scale GPUs built from this sweet-spot die.

Halo die. To realize the beyond-linear scaling, our proposed HALO reduces per-die L2 from 50 MB to 4 MB. Released area, approximately 18% of total die area, is conservatively reallocated to additional SMs, yielding 80 SMs per die, a 25% increase over the sweet-spot die. With the same per-SM throughput, the theoretical performance upper bound of HALO is $80\,/\,64 = 1.25\times$ over the sweet-spot baseline. Both die configurations occupy the same total die area, ensuring an iso-area comparison. Although each die retains only 4 MB of local L2, L2 sharing across a $4{\times}4$ domain provides an effective L2 capacity of $4\,\text{MB} \times 16 = 64\,\text{MB}$ for data reuse, comparable to the $1.25\times$ 50 MB on the sweet-spot die. HALO and single-GPU execution baselines adopts this die.

Name	Hidden Size	FFN Hidden Size	Attention Heads	Sequence Length	Batch Size
Llama	16384	53248	128	2048	8
GPT3	12288	49152	96	2048	8
Llama-L	65536	212992	256	4096	16
GPT3-L	49152	196608	192	4096	16

TABLE II
LLM WORKLOAD CONFIGURATIONS.

Area model. Our area estimates are grounded in publicly available die analyses of NVIDIA H100 [31], where the L2 cache subsystem (SRAM arrays, tag logic, and controllers) occupies approximately 20% of total die area. We adopt a conservative reallocation ratio: of the 46 MB of freed L2 area, only the SRAM array portion (which dominates L2 area) is assumed usable for SM placement. The remaining tag logic, controllers, and associated routing and power-delivery infrastructure are excluded. We further account for physical design constraints: SM macros have a coarser placement granularity than SRAM arrays, and the freed region must accommodate the SM's local interconnect and register-file area without violating metal-layer routing density limits. These constraints mean that not all of the 18.4% freed area translates directly into SM count, making the resulting 16 additional SMs a conservative estimate. To quantify sensitivity, we note that if the reclaimable area were 15% instead of 18.4%, the SM count would increase to only 74 (a $1.16\times$ bound), while 21% reclaimable area would yield 84 SMs (a $1.31\times$ bound). Beyond-linear scaling opportunity exists across this entire range.

3) Workloads: We evaluate HALO on four LLM training workloads from two model families: Llama 3.1 405B [5] (LLAMA) and GPT-3 175B [2] (GPT3). To assess scalability to larger models, we derive scaled-up variants, Llama 3.1 Large (LLAMA-L) and GPT-3 Large (GPT3-L). Table II lists detailed configurations. GEMM kernels use a CUTLASS-like implementation exploiting TMA and WGMMA instructions with warp specialization for multi-stage pipelining.

4) Baselines: We compare HALO against six baselines spanning three execution paradigms:

- *Multi-GPU*: **TP-64G** [31] applies tensor parallelism across 64 GPUs, each comprising a single sweet-spot die.
- *Single-GPU Execution* on wafer-scale GPU: **Global Interleaving** [30], [31], [32] is the address-interleaving scheme for multiple memory partitions, where we do not adopt the automatic L2 replication. **LADM** [16] is a locality-centric data and TB management scheme for multi-die execution.
- *Distributed Execution* on wafer-scale GPU: **2D-TP** is a widely adopted approach in WSE [22], [7], here we apply it

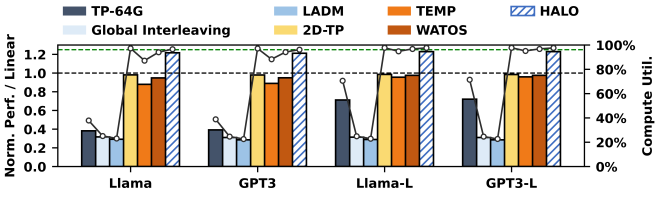

Fig. 9. End-to-end LLM training performance and compute utilization, normalized to ideal linear scaling of 64 sweet-spot dies. HALO exceeds linear scaling on all workloads, approaching the theoretical beyond-linear bound.

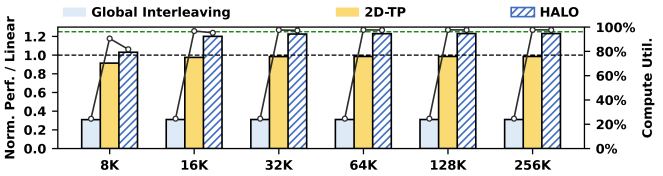

Fig. 10. GEMM performance across dimensions (M=N=K) from 8K to 256K. HALO consistently outperforms both Global Interleaving and 2D-TP.

to wafer-scale GPU. **TEMP** [48] proposes a novel TP and **WATOS** [47] proposes systematic coordination for LLM training on wafer-scale chips.

B. Overall Performance

1) End-to-End LLM Training: To evaluate the overall effectiveness of HALO, we compare end-to-end LLM training performance across all baselines and four workloads. Figure 9 presents the results normalized to ideal linear scaling, i.e., 64× performance of a single sweet-spot die. The right y-axis also shows compute utilization.

Our proposed HALO can achieve 1.21–1.23× performance speedup over ideal linear scaling across all workloads, reaching 96.9–98.3% of the theoretical beyond-linear scaling bound. Compute utilization consistently exceeds 95%, confirming that remote-access scheduling and speculative pipelining effectively manage communication overhead and keep the 80 SMs on the HALO die productively busy.

HALO on a single wafer outperforms TP-64G, a multi-GPU baseline with 64 GPUs, demonstrating the inherent advantage of wafer-scale integration. Compared to the single-GPU execution baselines, HALO achieves 3.91× and 4.25× average speedup over Global Interleaving and LADM. Compute utilization of Global Interleaving and LADM falls below 30%, as unmanaged all-to-all traffic severely overwhelms the interconnect, and LADM's locality optimization is even harmful due to the introduced communication imbalance.

Compared to distributed execution baselines, HALO achieves 1.24×, 1.32×, and 1.27× average speedup over 2D-TP, TEMP, and WATOS respectively. Although these baselines achieve near-linear scaling with only 1.8–7.9% performance loss through communication-centric optimization, all of them are fundamentally bounded by linear scaling due to L2 isolation, as analyzed in Section II. HALO surpasses this barrier through cross-die L2 sharing. These results validate that HALO's L2-sharing approach achieves practical beyond-linear scaling on real LLM workloads.

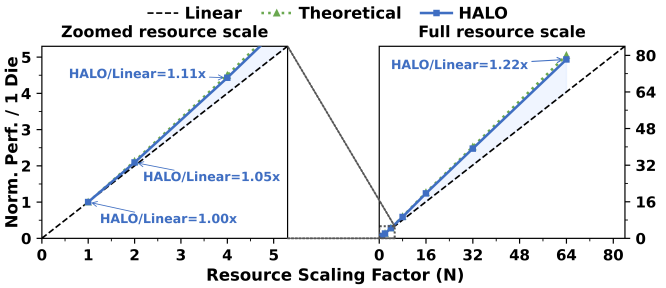

Fig. 11. Scalability of HALO, demonstrating beyond-linear performance scaling and near-theoretical performance.

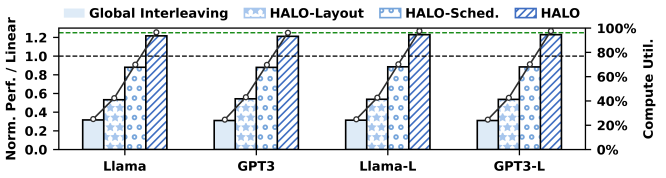

Fig. 12. Cumulative ablation study. Starting from Global Interleaving, each HALO component is incrementally added: compute-aligned interleaving, domain-partitioned execution, and speculative pipelining.

2) GEMM Performance: To evaluate operator-level performance of HALO, we sweep square GEMM sizes from 8K to 256K (M=N=K) and compare HALO with Global Interleaving and 2D-TP. As Figure 10 shows, HALO consistently achieves 3.33–3.97× and 1.13–1.25× speedups over the two baselines respectively. Speedup grows with GEMM size as deeper-pipeline startup overhead is amortized. These results show that HALO's benefit is robust across operator sizes, not tied to specific workload configurations.

3) Scalability Evaluation: To validate the beyond-linear performance scaling of HALO, we evaluate different numbers of dies, with the die count corresponding to the resource scaling factor N. We consider N (mesh shape) = 1 (1*1), 2 (1*2), 4 (2*2), 8 (2*4), 16 (4*4), 32 (4*8), and 64 (8*8), and perform a design-space exploration to identify the best SM-L2 configuration and L2-sharing domain size for each N. Figure 11 presents the results, with each point averaged across the four LLM workloads. "Linear" denotes linearly scaled performance relative to the single-die performance, whereas "Theoretical" denotes the theoretical performance under full utilization of each design. The results demonstrate that HALO achieves beyond-linear scalability with near-theoretical performance.

C. Ablation Study

To quantify individual contribution of each HALO component, we perform a cumulative ablation study starting from Global Interleaving and incrementally adding each technique, as shown in Figure 12.

Starting from Global Interleaving with a low compute utilization of 24.8% on average due to unmanaged all-to-all communication overhead, we incrementally add HALO's techniques. **+ Compute-Aligned Interleaving** (HALO-Layout) raises performance by 1.72× by confining remote accesses within the same row and column with fine-grained interleaving, substantially reducing traffic volume and latency.

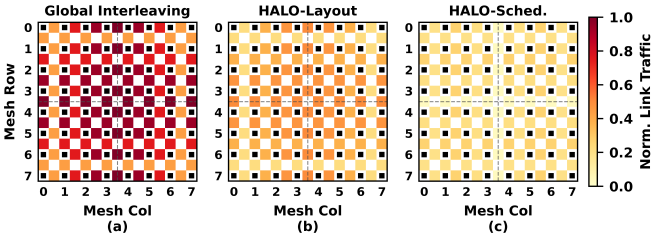

Fig. 13. Per-link traffic heatmap across the 8×8 mesh, comparing three scheduling strategies. Compute-aligned interleaving halves the maximum link load relative to global interleaving, and domain-partitioned execution (d=4) further confines traffic to within each domain.

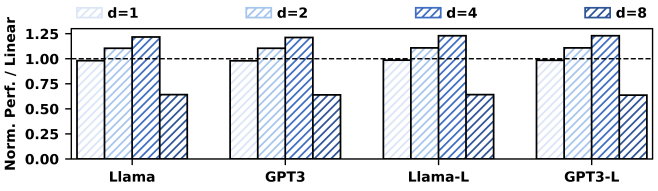

Fig. 14. Sensitivity to L2-sharing domain size d on a fixed 64-die wafer. d=4 optimally balances L2-sharing scope against communication overhead. Small domains waste area on L2, while large domains suffer excessive traffic.

+ Domain-Partitioned Execution (HALO-Sched.) further improves performance by 1.64× by partitioning the wafer into L2-sharing domains, further reducing the remaining communication overhead and bringing utilization to 70.1%. At this point, the remaining performance gap is due to the elevated remote-access latency causing pipeline bubbles. **+ Speculative Pipelining** (HALO) delivers the final 1.39× boost by eliminating these pipeline bubbles, pushing utilization above 97.0% and enabling HALO to fully exploit the additional SMs on its 80-SM die. This ablation demonstrates that all three components are essential: remote-access scheduling alone is insufficient without latency hiding, and speculative pipelining alone cannot compensate for excessive communication overhead.

D. Remote-Access Scheduling Analysis

1) Traffic Volume Reduction: To validate that compute-aligned interleaving and domain-partitioned execution reduce traffic as designed in Section III-B, we visualize the per-link traffic distribution across the 8×8 mesh under different scheduling strategies. Figure 13 visualizes the results. The maximum link load directly limits system performance, as it determines the bandwidth bottleneck.

With compute-aligned interleaving, the maximum link load is reduced by half compared to global interleaving, consistent with the theoretical analysis ($load_{max}$ from $p/4$ to $p/8$). Upon adding domain-partitioned execution with d=4, traffic is further reduced by narrowing the L2-sharing domain, reducing $load_{max}$ from $p/8$ to $d/8$. These results confirm that remote-access scheduling effectively controls traffic volume as predicted by the analytical model, and that the combination of operator-level and system-level optimization is necessary to bring communication overhead to a manageable level.

2) Domain Size Sensitivity: The L2-sharing domain size d governs the trade-off between L2-sharing scope and communication overhead: a larger domain provides more effective

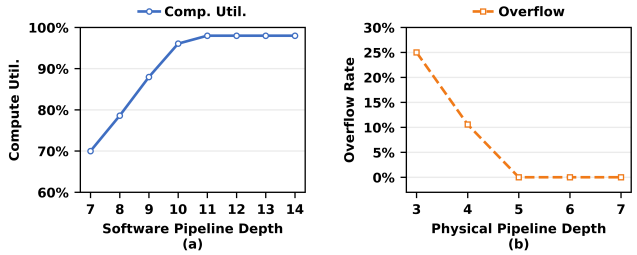

Fig. 15. Pipeline depth and physical resource efficiency. (a) ≥12 stages are needed for >98% utilization, far beyond the 7 stages existing GPUs support. (b) With 12 stages, only 5 physical slots suffice for full utilization.

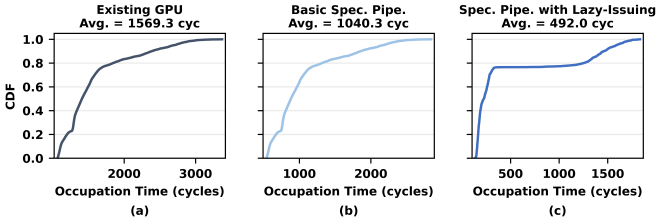

Fig. 16. CDF of per-stage shared-memory occupation time. On-demand allocation reduces occupation by 33.7% over conservative allocation. Adding lazy-issuing achieves a 68.6% total reduction, enabling rapid slot turnover.

L2 capacity but incurs higher traffic, while a smaller domain limits communication but requires retaining more local L2, leaving less area for SMs. To identify the optimal operating point, we sweep d and re-balance the SM-L2 configuration for each size. Figure 14 presents the results.

Smaller domains (d=2) cannot achieve optimal performance because the large per-die L2 requirement occupies die area that could otherwise be allocated to SMs. Larger domains (d=8, full wafer) also underperform due to increased communication overhead that degrades compute utilization. The optimal point is d=4, which balances sufficient L2 sharing (effective 64 MB) with manageable communication cost. This validates our proposed domain-partitioned execution design and our selected configuration in our evaluation.

E. Speculative Pipelining Analysis

1) Pipeline Depth Requirement: To determine the minimum pipeline depth required to hide the long remote memory access latency characterized in Section II, we vary software pipeline depth from 7 to 14 while providing sufficient physical resources, as shown in Figure 15(a).

Utilization increases monotonically from 69.9% at depth 7 to >98% at depth≥12, demonstrating that at least 12 stages are essential for hiding the remote-access latency. On existing GPUs where pipeline depth is bounded by $\lfloor$SMEM capacity/per-stage size$\rfloor$, the maximum achievable depth with 227 KB shared memory on NVIDIA H100 is only 7 stages. This gap between the required 12 stages and the achievable 7 stages underscores the necessity of speculative pipelining: without it, compute utilization is limited to 69.9%, leaving over 30% of the computational potential unutilized.

2) Physical Resource Sensitivity: To validate that our proposed speculative pipelining can achieve the required pipeline depth with limited physical resources, we fix the logical

software pipeline depth at 12 and vary the number of physical pipeline stages, as shown in Figure 15(b).

With only 3 physical slots available, the speculative pipelining can already achieve a low overflow rate of 25.0%. As the number of physical stages increases to 5, the overflow rate approaches zero. This demonstrates that speculative pipelining enables a Hopper-like shared-memory capacity (7 pipeline slots) to fully sustain 12 software stages, and that shared memory can potentially be reduced to only 5 slots to save area. The result validates the key advantage of speculative pipelining: the software pipeline depth is no longer bounded by physical shared-memory capacity.

3) Occupation Time Reduction: To understand how on-demand allocation and lazy-issuing each contribute to reducing per-stage shared-memory occupation time, the key metric that determines physical-slot turnover rate, we compare three strategies. Figure 16 shows the results. On existing GPUs, the average occupation time is 1569 cycles, dominated by the round-trip latency from load issuance to data arrival. Basic speculative pipelining reduces this to 1040 cycles (33.7% reduction) by deferring resource allocation to first-byte arrival, eliminating the redundant occupation during the issue-to-arrival interval. Adding lazy-issuing further reduces occupation to 492 cycles (68.6% reduction from existing GPUs) by aligning arrival times of requests with different latencies, compressing the first-to-last-byte arrival spread. This tighter occupation window means each physical slot turns over faster, confirming lazy-issuing as essential for sustaining deep pipelines with limited physical resources. The remaining long tail primarily arises during the transient warm-up phase and is handled by warm-up throttling. In steady state, long-occupation stages are rare and occur during congested windows in which subsequent stages also arrive later. Consequently, these long tails do not create sufficient simultaneous slot pressure to cause overflow with five physical stages.

F. Power Evaluation

Power is a critical design objective for wafer-scale GPUs. We evaluate power of HALE using our extended AccelWattch [15] to support Hopper architecture and SoW-X links [40]. We calibrate the GPU die model against H100 SXM GPU and configure it to model our dies. For die-to-die links, we use publicly available SoW-X data [40] to estimate the link power. We also adopt tile-level model for fast exploration, which is also followed by Section IV-G.

We evaluate the iso-area and iso-frequency setting used in main experiments to characterize the power of HALO's iso-area design. As shown in Figure 17(a), HALO consumes 30.3kW on average, only 9.0% more than 2D-TP's 27.8kW. This increase is primarily due to the higher SM count of HALO die, while smaller L2 offsets part of the cost. The contribution of increased inter-die traffic volume is small relative to those of other components. Because performance gain exceeds power increase, HALO improves power efficiency by 14.3% over 2D-TP on average. We further evaluate an iso-area and iso-power setting by reducing the frequency of HALO until its

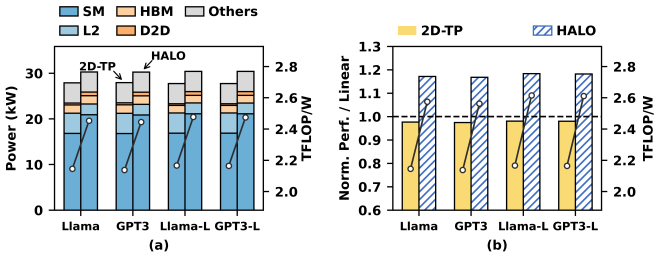

Fig. 17. Power evaluation of HALO, demonstrating HALO's efficiency. (a) Power consumption and efficiency under iso-frequency setting. (b) Performance and power efficiency under iso-power setting.

power matches that of 2D-TP. HALO requires only a modest frequency reduction to 0.965×. Figure 17(b) shows that, even under this iso-power constraint, HALO still achieves a 1.19× average speedup over 2D-TP and improves power efficiency by 20.3% on average. These results demonstrate HALO's advantage under both iso-area and iso-power constraints.

G. Discussion

We also evaluate HALO with diverse workloads, die-to-die bandwidths, topologies, and baseline configurations to demonstrate its generality. We use a tile-level model to enable fast generalization and evaluation. We adopt this for both new design points and those evaluated with detailed simulation in the main experiments to ensure fair comparisons. Similar results to their detailed simulations validates the fidelity.

1) Diverse Workloads: We evaluate HALO on diverse workloads to validate its generality, including inference (prefill and decode), MoE layer, and RWKV. We adopt Llama 3.1 405B [5] for inference, as in main experiments, DeepSeek-V3 MoE layer [23] for MoE, and RWKV-70B-style recurrent block extrapolated from public RWKV-14B [36] scaling pattern for RWKV. Figure 18 shows that HALO outperforms distributed execution baselines on all these workloads, with speedups of 1.12×, 1.19×, and 1.23×, demonstrating its generality. Speedups for inference and MoE layer are lower than for LLM training because decode provides no data reuse and each expert computation has relatively fine granularity, respectively.

2) Diverse Die-to-Die Bandwidth: We perform a sensitivity study of HALO across diverse die-to-die bandwidths from 1 to 10 TB/s. For each bandwidth, we conduct a design space exploration to identify the best SM-L2 configuration and L2-sharing domain size. Figure 19 shows that HALO outperforms distributed execution when bandwidth exceeds 1 TB/s, with speedups increasing as bandwidth grows, where speedups in average are 1.01×, 1.24×, and 1.27× for 2, 5, and 10 TB/s. These results demonstrate HALO's performance benefits under high bandwidth provided by wafer-scale integration. It also explains why prior multi-chip research has not explored a HALO-style approach: limited inter-chip bandwidth cannot support cross-die L2 sharing, highlighting the novelty of HALO for wafer-scale GPUs.

3) Diverse Topology: We perform a topology sensitivity study of HALO using two additional feasible wafer-scale interconnects: FRED-like fabric [38] and mesh-switch fab-

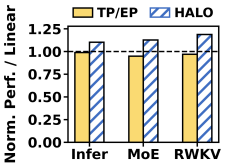

Fig. 18. Evaluation across diverse workloads.

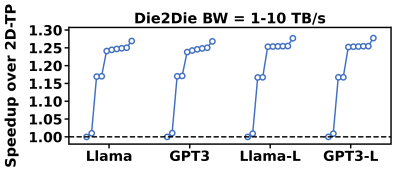

Fig. 19. Evaluation across diverse die-to-die bandwidths.

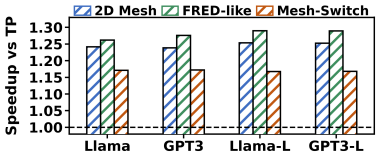

Fig. 20. Evaluation across diverse topologies.

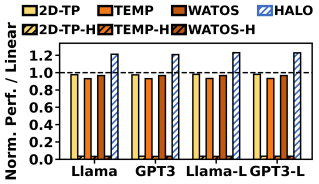

Fig. 21. Evaluation across diverse baseline die configurations.

ric [51]. FRED-like fabric is conceptually similar to the high-radix fat-tree NVLink/NV-Switch fabric used in multi-chip systems. We model 32 GPU dies, because switch fabric occupies approximately half of the wafer area [38], and a fabric bandwidth of 5 TB/s. The baseline uses 1D-TP, whereas HALO uses the full wafer as L2-sharing domain. We adapt compute-aligned interleaving accordingly: for $B{=}AW$, W and B are localized to each die's output shard, while A is fine-grain interleaved. For mesh-switch fabric, we model ten $2{\times}2$ mesh groups [51] connected through a central switch. Its baseline uses hierarchical TP, with 1D-TP across groups and 2D-TP within each group, whereas HALO confines L2 sharing to each local $2{\times}2$ group. Figure 20 shows that HALO improves performance across diverse topologies, with average speedups of $1.24\times$, $1.28\times$, and $1.17\times$, demonstrating its generality.

4) Diverse Baseline Die Configuration: We evaluate HALO against diverse baseline die configurations to validate our baseline selection. In addition to the sweet-spot die used by the distributed execution baselines, we also evaluate these baselines using the Halo die. Figure 21 shows that baselines using the Halo die, denoted by "-H", perform substantially worse than those using the sweet-spot die, dropping by 96.4% in average. Under distributed execution, L2 caches remain isolated, so each die can exploit only its local L2 for local computation. The small local L2 capacity of the Halo die therefore results in poor performance. A fair comparison should provide HALO and each baseline with the same die area and their respective best design configurations, rather than equalizing the number of SMs. The sweet-spot die is therefore a more appropriate configuration for the distributed execution baselines than the Halo die.

H. Hardware Overhead

We evaluate hardware overhead of HALO under 12nm technology. We implement the components in RTL and perform synthesis with commercial design tools. HALO introduces three hardware components: the extended TMA address generator from Section III-B4, the traffic-class allocator from Section III-B4, and the speculative pipelining unit from Section III-C4. Per die, TMA extension occupies $3.13{\times}10^{-2}\,\mathrm{mm}^2$, traffic allocator $9.77{\times}10^{-4}\,\mathrm{mm}^2$, and speculative pipelining unit $9.38{\times}10^{-2}\,\mathrm{mm}^2$. The total area cost is $0.126\,\mathrm{mm}^2$ per die, less than 0.02% of die area. This minimal overhead validates that HALO is practical to implement.

V. Related Work

Multi-die and wafer-scale chip architectures. Multi-die GPUs have been extensively studied since MCM-GPU [1]. Subsequent works propose NUMA-aware memory management, scheduling, and caching optimizations [26], [53], [16], [57], where L1.5 cache is widely adopted to reduce SM-direct remote access overhead but introduces redundancy [57]. General-purpose wafer-scale GPU architectures [34], [49] adopt single-GPU execution but achieve only sub-linear scaling due to architecture- and workload-unaware traffic management. LLM-centric solutions for training [48], [47], inference [22], [42], [7], [43], [50], [54] adopt distributed execution to achieve linear scaling. HALO departs from both paradigms by *architecturally managing* cross-die remote loads, enabling deliberate L2 sharing that converts memory-system redundancy into additional compute for beyond-linear scaling.

LLM training parallelism. Modern LLM training employs multi-dimensional parallelism: tensor parallelism [41], pipeline parallelism [28], [12], data parallelism [37], [58]. Alpa [59] automates inter- and intra-operator parallelism, and MeshSlice [27] proposes efficient 2D tensor parallelism for 2D topology. There are also some works exploring architectural innovations to accelerate parallel execution of LLMs on multi-GPUs with hardware-accelerated computation-communication overlap [35], [60] and in-switch computing [18], [55], [56]. However, these techniques partition computation across independent GPU instances and communicate through collective operations, inherently isolating the L2 cache. HALO's introduces a fundamentally different execution model based on SM-direct remote loads within a domain for L2 sharing.

VI. Conclusion

This paper presents HALO, a hardware-software co-design framework that enables beyond-linear scaling on wafer-scale GPUs through breaking L2 isolation with architecturally managed SM-direct remote loads. By integrating remote-access scheduling and speculative pipelining, HALO achieves $1.24\times$ average speedup over SOTA distributed execution on LLM training, reaching 98% of the theoretical beyond-linear bound at less than 0.02% die-area overhead.

Acknowledgement

We sincerely thank the anonymous MICRO'26 reviewers for their valuable suggestions. This work is partially sponsored by Hong Kong Research Grants Council (RGC) CRF-YCRG C6003-24Y and GRF 16216825, and Shanghai QiYuan Innovation Foundation QY2025-QN-SJTU-011. It was partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

REFERENCES

[1] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[2] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[4] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv e-prints*, pp. arXiv–2407, 2024.

[5] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[6] H. Guo, S. Cao, L. Li, and X. Zhang, "A review on the mainstream through-silicon via etching methods," *Materials Science in Semiconductor Processing*, vol. 137, p. 106182, 2022.

[7] C. He, Y. Huang, P. Mu, Z. Miao, J. Xue, L. Ma, F. Yang, and L. Mai, "{WaferLLM}: Large language model inference at wafer scale," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, 2025, pp. 257–273.

[8] S. Hou, C. H. Lee, T.-D. Wang, H. C. Hou, and H.-P. Hu, "Supercarrier redistribution layers to realize ultra large 2.5 d wafer scale packaging by cowos," in *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*. IEEE, 2023, pp. 510–514.

[9] C.-H. Hsia, S. Park, S. Watanabe, M. Arnold, Z. El-Mekki, M. Delande, E. Shafahian, P. K. Gowda, H. Struyf, and A. Radisic, "Wafer-level electrochemical deposition and processing of nanotwinned cu rdl," in *2024 IEEE International Interconnect Technology Conference (IITC)*. IEEE, 2024, pp. 1–3.

[10] Y.-C. Hu, Y.-M. Liang, H.-P. Hu, C.-Y. Tan, C.-T. Shen, C.-H. Lee, and S. Hou, "Cowos architecture evolution for next generation hpc on 2.5 d system in package," in *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*. IEEE, 2023, pp. 1022–1026.

[11] P. Huang, C. Lu, W. Wei, C. Chiu, K. Ting, C. Hu, C. Tsai, S. Hou, W. Chiou, C. Wang *et al.*, "Wafer level system integration of the fifth generation cowos®-s with high performance si interposer at 2500 mm2," in *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*. IEEE, 2021, pp. 101–104.

[12] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[13] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2013, pp. 86–96.

[14] B. Jiao, J. Qiao, S. Jia, R. Liu, X. Wei, S. Yun, Y. Kong, Y. Ye, X. Du, L. Yu *et al.*, "Low stress tsv arrays for high-density interconnection," *Engineering*, vol. 38, pp. 201–208, 2024.

[15] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO-54: 54th Annual IEEE/ACM International symposium on microarchitecture*, 2021, pp. 738–753.

[16] M. Khairy, V. Nikiforov, D. Nellans, and T. G. Rogers, "Locality-centric data and threadblock management for massive gpus," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1022–1036.

[17] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486.

[18] B. Klenk, N. Jiang, G. Thorson, and L. Dennison, "An in-network architecture for accelerating shared-memory multiprocessor collectives," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 996–1009.

[19] J. H. Lau, "Recent advances and trends in multiple system and heterogeneous integration with tsv interposers," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 13, no. 1, pp. 3–25, 2023.

[20] J. H. Lau, G. C.-F. Chen, J. Y.-C. Huang, R. T.-S. Chou, C. C.-L. Yang, H.-N. Liu, and T.-J. Tseng, "Hybrid substrate by fan-out rdl-first panel-level packaging," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 8, pp. 1301–1309, 2021.

[21] J. H. Lau, C.-T. Ko, K.-M. Yang, C.-Y. Peng, T. Xia, P. B. Lin, J. Chen, P. P.-C. Huang, H.-N. Liu, T.-J. Tseng *et al.*, "Panel-level fan-out rdl-first packaging for heterogeneous integration," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 10, no. 7, pp. 1125–1137, 2020.

[22] S. Lie, "Wafer-scale ai: Gpu impossible performance," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–71.

[23] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[25] G. H. Loh, Y. Xie, and B. Black, "Processor design in 3d die-stacking technologies," *Ieee Micro*, vol. 27, no. 3, pp. 31–48, 2007.

[26] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the socket: Numa-aware gpus," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 123–135.

[27] H. Nam, G. Gerogiannis, and J. Torrellas, "Meshslice: Efficient 2d tensor parallelism for distributed dnn training," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 821–834.

[28] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[29] M. Neisser, "International roadmap for devices and systems lithography roadmap," *Journal of Micro/Nanopatterning, Materials, and Metrology*, vol. 20, no. 4, pp. 044 601–044 601, 2021.

[30] NVIDIA, "Nvidia a100 tensor core gpu." *https://www.nvidia.com/en-us/data-center/a100*, 2020.

[31] NVIDIA, "Nvidia h100 tensor core gpu." *https://www.nvidia.com/en-us/data-center/h100*, 2022.

[32] NVIDIA, "Nvidia blackwell architecture technical brief." *https://resources.nvidia.com/en-us-blackwell-architecture*, 2024.

[33] NVIDIA, "Nvidia gb200 nvl72." *https://www.nvidia.com/en-us/data-center/gb200-nvl72/*, 2024.

[34] S. Pal, D. Petrisko, M. Tomei, P. Gupta, S. S. Iyer, and R. Kumar, "Architecting waferscale processors-a gpu case study," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 250–263.

[35] S. Pati, S. Aga, M. Islam, N. Jayasena, and M. D. Sinclair, "T3: Transparent tracking & triggering for fine-grained overlap of compute & collectives," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 1146–1164.

[36] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, L. Derczynski *et al.*, "Rwkv: Reinventing rnns for the transformer era," in *Findings of the association for computational linguistics: EMNLP 2023*, 2023, pp. 14 048–14 077.

[37] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[38] S. Rashidi, W. Won, S. Srinivasan, P. Gupta, and T. Krishna, "Fred: A wafer-scale fabric for 3d parallel dnn training," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 34–48.

[39] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 14–27.

[40] P.-C. Shih, A.-J. Su, K.-H. Tam, T.-C. Huang, K. Chuang, and J. Yeh, "Sow-x: A novel system-on-wafer technology for next generation ai server application," in *2025 IEEE 75th Electronic Components and Technology Conference (ECTC)*. IEEE, 2025, pp. 1–6.

[41] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[42] E. Talpes, D. Williams, and D. D. Sarma, "Dojo: The microarchitecture of tesla's exa-scale computer," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE, 2022, pp. 1–28.

[43] X. Tang, J. Hou, D. Jiang, T. Wei, J. Liu, J. Deng, H. Wang, Q. Yang, H. Shang, C. Li *et al.*, "Moentwine: Unleashing the potential of wafer-scale chips for large-scale expert parallel inference," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–15.

[44] TSMC, "Tsmc's chip on wafer on substrate." *https://3dfabric.tsmc.com/english/dedicatedFoundry/technology/cowos.htm*, 2025.

[45] W. J. Turner, J. W. Poulton, J. M. Wilson, X. Chen, S. G. Tell, M. Fojtik, T. H. Greer, B. Zimmer, S. Song, N. Nedovic *et al.*, "Ground-referenced signaling for intra-chip and short-reach chip-to-chip interconnects," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2018, pp. 1–8.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[47] H. Wang, Z. Wang, H. Wang, J. Hou, T. Wei, C. Li, Y. Hu, and S. Yin, "Watos: Efficient llm training strategies and architecture co-exploration for wafer-scale chip," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–19.

[48] H. Wang, T. Wei, Z. Wang, D. Jiang, Q. Yang, J. Liu, J. Hou, C. Li, J. Deng, Y. Hu *et al.*, "Temp: A memory efficient physical-aware tensor partition-mapping framework on wafer-scale chips," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–18.

[49] D. Xu, Y. Li, Y. Sun, J. Ren, and Y. Sun, "Hdpat: Hierarchical distributed page address translation for wafer-scale gpus," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–12.

[50] Z. Xu, D. Kong, J. Liu, J. Li, J. Hou, X. Dai, C. Li, S. Wei, Y. Hu, and S. Yin, "Wsc-llm: Efficient llm service and architecture co-exploration for wafer-scale chips," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1–17.

[51] Q. Yang, T. Wei, S. Guan, C. Li, H. Shang, J. Deng, H. Wang, C. Li, L. Wang, Y. Zhang *et al.*, "Pd constraint-aware physical/logical topology co-design for network on wafer," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 49–64.

[52] S.-F. Yang, W.-C. Wang, Y.-T. Lin, C.-C. Hung, H.-Y. Tung, and J. Hsieh, "Signal integrity designs at organic interposer cowos-r for hbm3-9.2 gbps high speed interconnection of 2.5 d-ic chiplets integration," in *2024 IEEE 74th Electronic Components and Technology Conference (ECTC)*. IEEE, 2024, pp. 1098–1103.

[53] V. Young, A. Jaleel, E. Bolotin, E. Ebrahimi, D. Nellans, and O. Villa, "Combining hw/sw mechanisms to improve numa performance of multi-gpu systems," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 339–351.

[54] X. Yu, D. Jiang, J. Deng, J. Liu, C. Li, S. Yin, and Y. Hu, "Cramming a data center into one cabinet, a co-exploration of computing and hardware architecture of waferscale chip," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 631–645.

[55] C. Zhang, Q. Zhang, Z. Zhou, Y. Diao, H. Wang, Z. Zhou, Z. Tu, Z. Li, G. Sun, Z. Song, Z. Ji, J. Leng, and M. Guo, "Towards compute-aware in-switch computing for llms tensor-parallelism on multi-gpu systems," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–15.

[56] Q. Zhang, C. Zhang, Z. Zhou, H. Wang, Z. Zhou, Z. Tu, G. Sun, Z. Xie, Y. Diao, Z. Ji, J. Leng, G. He, and M. Guo, "Accelerating moe with dynamic in-switch computing on multi-gpus," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2026, pp. 921–935.

[57] S. Zhang, M. Naderan-Tahan, M. Jahre, and L. Eeckhout, "Sac: Sharing-aware caching in multi-chip gpus," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[58] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023.

[59] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing *et al.*, "Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 559–578.

[60] Z. Zhou, C. Zhang, S. Zhang, Q. Zhang, H. Wang, Z. Zhou, Z. Tu, G. Sun, Y. Diao, Z. Ji, J. Leng, G. He, and M. Guo, "Moe-hub: Taming software complexity for seamless moe overlap with hardware-accelerated communication on multi-gpu systems," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2026, pp. 2459–2474.