Accelerating MoE with Dynamic In-Switch Computing on Multi-GPUs

Qijun Zhang[1], Chen Zhang[2*], Zhuoshan Zhou[2], Haibo Wang[3], Zhe Zhou[3], Zhipeng Tu[3], Guangyu Sun[4], Zhiyao Xie[1], Yijia Diao[2], Zhigang Ji[2], Jingwen Leng[2,5], Guanghui He[2], Minyi Guo[2]
Hong Kong University of Science and Technology[1], Shanghai Jiao Tong University[2],
Huawei Technologies Co. Ltd.[3], Peking University[4], Shanghai Qi Zhi Institute[5]
qzhangcs@connect.ust.hk, {chenzhang.sjtu, zs.zhou, diao_yijia, zhigangji, guanghui.he}@sjtu.edu.cn, eezhiyao@ust.hk,
{wanghaibo33, zhouzhe22, tuzhipeng3}@huawei.com, gsun@pku.edu.cn, {leng-jw, guo-my}@cs.sjtu.edu.cn

***Abstract*—Mixture-of-Experts (MoE) has been adopted by many leading large models to reduce computational requirements. However, frequent inter-GPU communication in MoE expert parallelism (EP) becomes a performance challenge. We observe substantial redundant inter-GPU data transfers in MoE that can be potentially addressed by in-switch computing. Unfortunately, the existing solution, NVLink SHARP (NVLS), can only support static collectives with regular patterns, incapable of dynamic communication with irregular patterns in MoE.**

To bridge the functionality gap, we propose DySHARP, an integral dynamic in-switch computing solution to accelerate MoE, encompassing both communication primitives and communication-aware scheduling: 1) Dynamic multimem addressing co-designs ISA, architecture, and runtime, as a dynamic extension to NVLS, reducing redundant traffic. However, the resulting traffic reduction is inherently asymmetric between two directions, preventing it from directly translating into speedup. 2) Token-centric kernel fusion deeply fuses the dispatch-computation-combine pipeline, resolving this asymmetry to translate traffic reduction into actual speedup. Compared with the state-of-the-art solution, DySHARP achieves up to 1.79× speedup.

I. Introduction

In recent years, the development of large language models has brought groundbreaking advances to many fields, including natural language processing [27], [52], computer vision [7], [25], and reasoning [6], [55]. To enhance model capabilities, the parameter count of models has been continuously scaling up [17], leading to a significant surge in the computational demands required for model training. To reduce these computational requirements, many leading large models, including DeepSeek [6], GPT [40], [41], Llama [27], Qwen [55], and Pangu [49], have opted to adopt Mixture-of-Experts (MoE) architecture [20] for model parameter scaling. Compared to the dense Transformer [52], MoE splits Feed-Forward Network (FFN) layer into multiple experts. Each token dynamically activates only a small subset of experts, enabling a substantial reduction in computational overhead.

With the ever-increasing computational and memory demands, *expert parallelism* (EP) [20] that distributes experts across GPUs is proposed to train MoE on multi-GPU systems. However, since a token may activate experts on remote GPUs, EP requires frequent inter-GPU communication, including *Dispatch* and *Combine* communication operators. This

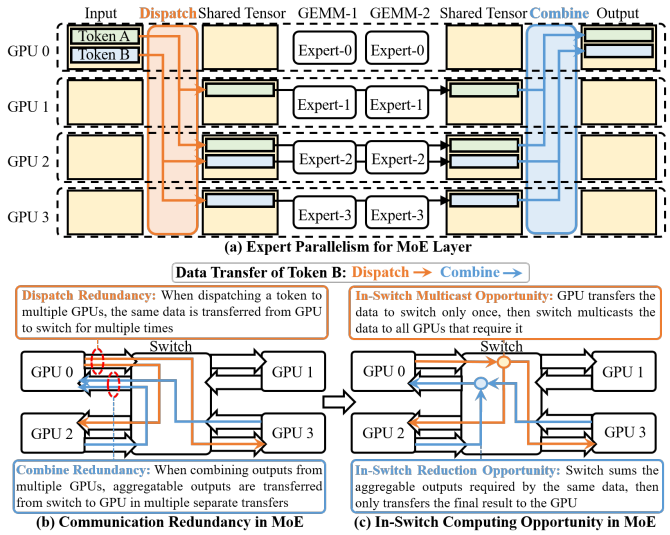

Fig. 1. In-switch computing opportunity in MoE. MoE has significant redundant transfer that can be potentially addressed with in-switch computing.

frequent inter-GPU communication becomes a performance bottleneck in MoE execution [10], [12], [43], [46], [53], [57], consuming 50-80% execution time [57].

We observe that both the Dispatch and Combine involve a fundamental communication inefficiency that no existing work tackles: redundant data movement across GPUs. As illustrated in Fig. 1(b), 1) When dispatching a token to multiple GPUs, the same data is transferred from GPU to switch multiple times. 2) When combining outputs from multiple GPUs, aggregatable outputs required by the same data are transferred from switch to GPU as multiple separate transfers. Profiling DeepSeek-v3 on a simulated GH200 NVL32 [33] shows a near 50% communication redundancy out of total traffic.

In-switch computing, which has been integrated in NVLink/NVSwitch interconnection [33], [36] through NVLink SHARP (NVLS) [19], can potentially address these redundancies through in-switch multicast and in-switch reduction, as illustrated in Fig. 1(c). 1) With in-switch multicast, the GPU can only transfer the data to the switch once, then the switch multicasts the data to all GPUs, eliminating the dispatch redundancy. 2) With in-switch reduction, the switch can sum aggregable outputs and only transfer the final result to the GPU, eliminating the combine redundancy. However, despite

* Chen Zhang is the corresponding author.

the promising opportunity, the existing NVLS design is fundamentally static, restricted to the static collectives with regular communication patterns where the target sets are fixed and addresses are symmetric. Such a static NVLS design is thus incapable of accelerating dynamic operators with irregular communication patterns in MoE, with varying target sets and asymmetric addressing. A software-based workaround reinterpreting MoE communication as static collectives generates substantial useless traffic, which reaches 340% in our profiling, negating the benefits of in-switch computing.

This functionality gap motivates us to propose a *dynamic* in-switch computing solution to eliminate this communication redundancy in MoE. Achieving this requires not only communication primitives to reduce redundant traffic but also communication-aware scheduling to translate the reduction into actual speedup. However, designing such a solution faces two challenges: ① The existing multi-GPU system design lacks top-down architectural support to enable the dynamic in-switch computing. ② Isolated dataflow schedule, where Dispatch and Combine are executed in isolation, incurs directional bandwidth imbalance, causing low overall bandwidth utilization. To address these challenges, DySHARP proposes an integral solution encompassing both communication primitives and communication-aware scheduling: ❶ Dynamic multimem addressing, a dynamic extension of NVLS's multimem addressing. Packet carries a single multimem address and a lightweight target list, with each GPU managing its memory locally. This supports irregularity of dynamic communication with high efficiency. ❷ Token-centric kernel fusion to co-schedule operators. It utilizes token-level data dependency to pipeline the whole Dispatch-Computation-Combine chain. This token-paced pipeline merges complementary asymmetric communication patterns to improve bandwidth utilization. The two techniques work as an *integral* solution, where neither alone is sufficient: through dynamic multimem addressing, DySHARP eliminates nearly half of the total traffic, but the resulting reduction is inherently asymmetric between two directions, preventing it from directly translating into speedup. Token-centric kernel fusion resolves this asymmetry, translating the traffic reduction into actual speedup and achieving complete redundancy elimination. To the best of our knowledge, this is the first work to accelerate dynamic communication in MoE with in-switch computing.

In summary, this paper makes the following contributions:

- We conduct an in-depth analysis of the opportunity of leveraging in-switch computing to accelerate MoE's dynamic communication and the limitations of existing NVLS.
- We propose DySHARP, the first dynamic in-switch computing framework that introduces dynamic multimem addressing and token-centric kernel fusion to accelerate dynamic communication operators with fully exploited in-switch computing capabilities.
- We evaluate DySHARP extensively under diverse workload configurations on a simulated GH200 NVL32 systems, demonstrating up to 1.79× speedup compared to the SOTA MoE acceleration solution.

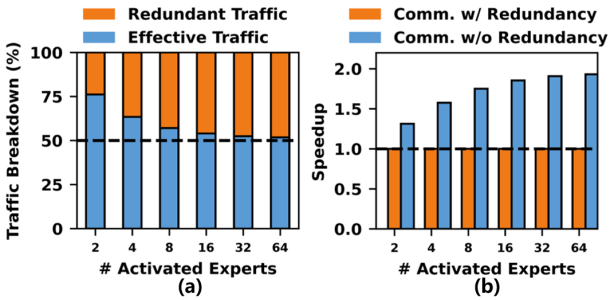

Fig. 2. The quantification of (a) redundant data transfer and (b) acceleration opportunity of redundancy elimination of DeepSeek-V3 on a simulated GH200 NVL32-like 32-GPU system. It demonstrates significant potential for eliminating redundancy with in-switch computing.

II. Background and Motivation

A. MoE Structure and Expert Parallelism

MoE layer consists of multiple small FFNs each referred to as an expert, where each token only dynamically activates $topk$ experts selected by the gate network for computation, reducing computational cost. As illustrated in Fig. 1(a), each token is assigned to $topk$ experts, the assigned experts execute FFN computations (GEMM-1 and GEMM-2), and the $topk$ outputs are aggregated to produce the final result.

To train MoE on multi-GPU systems, expert parallelism (EP) distributes experts across GPUs, introducing two inter-GPU communication operations as shown in Fig. 1(a): *Dispatch* sends tokens to GPUs hosting their activated experts, and *Combine* aggregates expert outputs back. This frequent communication becomes a performance bottleneck, accounting for 50-80% of MoE layer execution [57]. Our experiment shows communication consumes 70.4% of MoE layer execution in DeepSeek-V3 on simulated GH200 NVL32. Newer GPUs like NVIDIA Blackwell [35] and Rubin [39] are expected to see computational capacity outpace growth in communication, exacerbating this ratio.

B. Communication Redundancy in MoE

A fundamental inefficiency of MoE communication lies in the large amount of redundant data movement across GPUs. In Dispatch, a single token often needs to reach multiple GPUs, where the same data is transmitted multiple times from source GPU to switch. For example, in Fig. 1(b), Token B on GPU0 must be sent to GPU 2 and 3, causing two identical but separate transfers over the GPU0-switch link. Similarly, in Combine, aggregatable expert outputs of a token from multiple GPUs are individually sent back, creating multiple separate transfers from switch to source GPU that the token originally resides. Intermediate outputs from both GPU 2 and 3 contribute to the output of Token B, causing two aggregatable but separate transfers over the switch-GPU0 link. Fig. 2(a) quantifies the redundant data transfer of DeepSeek-v3 with different numbers of activated experts on a 32-GPU system similar to the GH200 NVL32 [33]. It shows that there is significant redundant data transfer that accounts for nearly 50% of the total traffic when the number of activated experts is over 8.

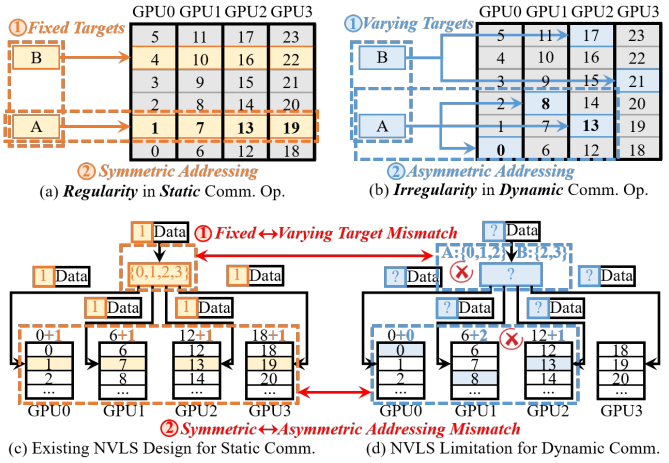

Fig. 3. Communication pattern and NVLS applicability for static and dynamic communications. NVLS's customization for the regularity of static collectives leads to limitations for supporting dynamic communications with irregularity.

Prior works [10], [12], [20], [30], [46], [53], [57], [59] mitigate MoE's communication overhead through optimized libraries or computation-communication overlap, but none tackle the fundamental problem: redundant transfers of identical or aggregatable data.

C. Opportunity and Limitation of In-Switch Computing

1) Redundancy Elimination with In-Switch Computing: Theoretically, redundancies introduced in Sec. II-B can be eliminated through in-switch computing [9], [11], [19], [45], [56], which augments interconnect fabric with lightweight processing primitives. Since Hopper [31], NVIDIA has integrated in-switch computing in multi-GPUs interconnected via NVLink/NVSwitch with NVLink SHARP (NVLS) [19]. NVLS introduces *multimem* instructions to utilize in-switch computing for redundant transfer elimination: `multimem.st` accelerates AllGather through in-switch multicast, and `multimem.ld_reduce` speeds up Reduce-Scatter via in-switch reduction. This technique can also potentially address communication redundancy in MoE, as illustrated in Fig. 1(c):

- *In-switch multicast* eliminates the redundancy in the dispatch operator: Only a single copy of the data is transferred from the source GPU to the switch. The switch then multicasts it to all destination GPUs requiring that data. As shown in Fig. 1(c), GPU 0 sends the token to the switch only once. Switch multicasts the token to GPU 2 and 3.
- *In-switch reduction* eliminates the redundancy in the combine operator: The multiple intermediate outputs are accumulated within the switch. Only the final result is transferred from the switch back to the source GPU. The data from GPU 2 and 3 are aggregated within the switch, and only the final accumulated result is transferred to GPU 0.

Therefore, with the capability of multicast and reduction inside the switch, the redundant data transfer can be eliminated. Fig. 2(b) quantifies the ideal acceleration opportunity with redundancy eliminated, indicating a significant ideal communication speedup near 2× with ≥8 activated experts.

2) Limitation of Existing In-Switch Computing: While in-switch multicast and reduction are appealing to eliminate this communication redundancy, existing solution (NVLS) is fundamentally **static**, restricted to static collective operators like AllGather and Reduce-Scatter, and thus incapable of accelerating dynamic communication operators in MoE.

This limitation stems from NVLS's customization for the regularity inherent in static collective operations. As illustrated in Fig. 3(a), this regularity manifests in two forms:

- Fixed target sets: All tokens of the operator always communicate with the same group of GPUs.
- Symmetric addressing: A token resides at identical memory offsets across GPUs.

This regularity enables NVLS to introduce multimem addressing, depicted in Fig. 3(c): Packets carrying only one address and rely on preconfigured target set to determine destinations, minimizing header overhead for high bandwidth efficiency.

However, such a NVLS cannot support the MoE's dynamic communication operators with the irregular pattern. The irregularity is shown in Fig. 3(b):

- Varying targets: each token may be routed to a different subset of experts, e.g., Token A → GPUs {0, 1, 2} while Token B → GPUs {2, 3}.
- Asymmetric addressing: tokens are independently allocated on each GPU in a dynamic approach, leading to divergent memory offsets, e.g., Token A are mapped to offsets {0, 2, 1} across GPUs {0, 1, 2}.

This communication pattern mismatch between dynamic communication in MoE and static collectives NVLS customized for makes NVLS unable to support MoE Dispatch and Combine: Preconfigured target set in switch cannot support varying targets, and carrying only one address in packet cannot determine the destinations under asymmetric addressing.

A naïve workaround reinterprets MoE communication as static collectives: using AllGather to emulate Dispatch and Reduce-Scatter for Combine. However, this forces all GPUs to send/receive data irrespective of actual need, generating useless traffic. Profiling DeepSeek-V3 on a GH200 NVL32-like system reveals that this translation introduces 340% useless traffic, negating the potential benefits of in-switch computing.

D. Design Philosophy and Challenges

To bridge this functionality gap, we propose DySHARP. The core philosophy is to provide a *dynamic* in-switch computing solution to eliminate communication redundancy. However, designing such a system still faces two challenges:

C1: Lack of Top-Down Architectural Support. The existing multi-GPU system design lacks architectural support for such a dynamic in-switch computing. To address this problem, DySHARP introduces **dynamic multimem addressing**, a dynamic extension to the existing multimem addressing framework. Dynamic multimem addressing introduces top-down enhancement to the existing multi-GPU system, including packet format, ISA, microarchitecture of both GPU and switch, and CUDA runtime. This full-stack extension

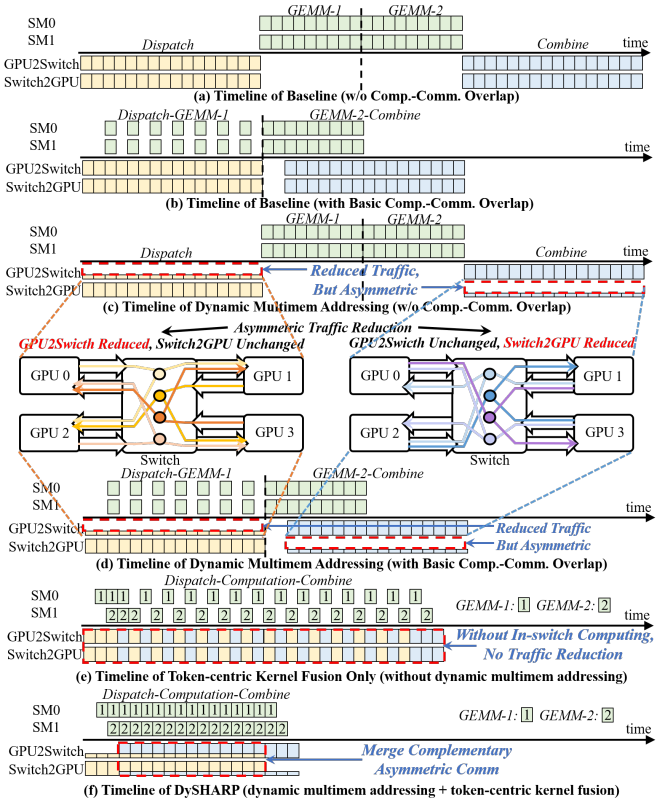

Fig. 4. How the two techniques work as an integral solution. Dynamic multimem addressing reduces traffic but inherently introduces asymmetric reduction across directions. Token-centric kernel fusion resolves this asymmetry, translating traffic reduction into overall speedup. Neither alone is sufficient.

enables the functionality of dynamic in-switch computing. By eliminating redundant traffic via in-switch multicast and reduction, dynamic multimem addressing reduces nearly half of the total communication traffic.

C2: Low Utilization of Isolated Dataflow Schedule. Even with architectural support, executing dispatch and combine as isolated operators creates *directional* bandwidth imbalance. In-switch multicast suppresses GPU→switch traffic but leaves switch→GPU heavy, and vice versa for reduction. The under-optimized direction dominates, resulting in low overall utilization. Overlap schemes [10], [12], [53], [57] can be composed with in-switch computing, but two communication kernels still remain isolated. DySHARP proposes **token-centric kernel fusion** to pipeline the whole Dispatch-Computation-Combine chain. This enables concurrent Dispatch and Combine, where complementary traffic patterns balance bidirectional bandwidth and significantly improve utilization. Fig. 4 illustrates how the two techniques work as an integral solution. As shown in Fig. 4(b), in-switch computing inherently introduces traffic reduction that is asymmetric between two directions: in-switch multicast reduces GPU-to-switch traffic in Dispatch, while in-switch reduction reduces switch-to-GPU traffic in Combine. When Dispatch and Combine are executed in isolation, the unreduced direction becomes the bottleneck, preventing the traffic reduction from directly translating into speedup. Token-centric kernel fusion resolves this asymmetry, translating the traffic reduction into end-to-end speedup and achieving complete redundancy elimination, as shown in Fig. 4(d). Neither technique alone is sufficient: without traffic reduction, token-centric kernel fusion alone offers no improvement over existing techniques, as shown in Fig. 4(c). Detailed analysis is in Sec. IV-A, with quantitative validation in Sec. VI-A3.

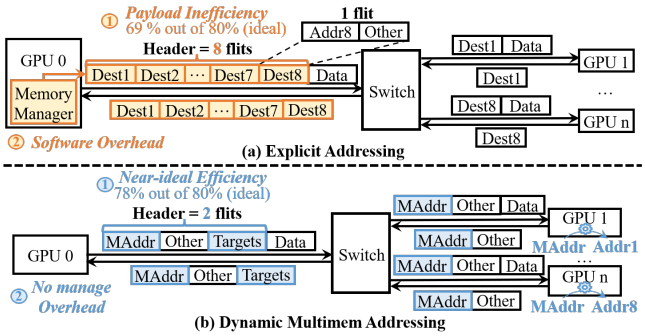

Fig. 5. Two potential solutions for dynamic in-switch computing. (a) The straightforward solution is explicit addressing, but it has low payload efficiency and high software overhead. (b) Dynamic multimem addressing that we employ achieves near-ideal payload efficiency and no software overhead.

III. Dynamic Multimem Addressing

A. Key Idea of Dynamic Multimem Addressing

To enable dynamic in-switch computing that supports the irregular dynamic communication in MoE, there are two potential solutions. The first solution is a straightforward approach that abandons multimem addressing and reverts to general shared-memory in-switch operations, which explicitly embeds all destination addresses, shown as explicit addressing in Fig. 5(a). The second solution is to extend existing multimem addressing, still carrying only one address that represents multiple destinations of a request, which is our proposed dynamic multimem addressing, as shown in Fig. 5(b).

For explicit addressing, while it can support varying targets and asymmetric addressing, it is inefficient: 1) *Payload inefficiency*: explicit destinations inflate packet headers and reduce payload efficiency; e.g., targeting eight GPUs requires eight destination flits in both request and response, dropping efficiency from an ideal 80% to 69%. 2) *Software overhead*: sender must track remote memory states, e.g., per-expert token counters, and precompute destination addresses, causing extra synchronization (over 5% performance loss [59]) and consuming 10-20% of GPU compute resources [6]. In comparison, dynamic multimem addressing is more promising. With only one address in the packet, it achieves high payload efficiency. Without considering detailed addressing in target GPUs, software overhead is also eliminated. Therefore, we employ dynamic multimem addressing in DySHARP.

The core of designing dynamic multimem addressing is *how to define the carried address that can represent the irregular multiple destinations of an operation.* We derive this address based on our understanding that Dispatch and Combine are the dynamic counterparts of AllGather and Reduce-Scatter. As shown in Fig. 6 (one expert per GPU), two properties follow:

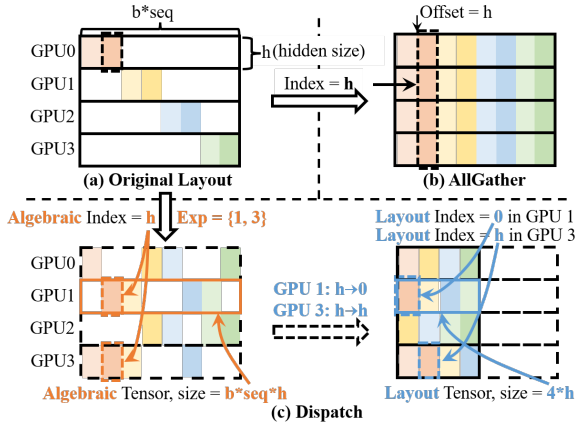

Fig. 6. Comparison between Dispatch and AllGather. Dispatch/Combine are dynamic variants of AllGather/Reduce-Scatter, still with identical algebraic index across GPUs but per-GPU managed asymmetric memory layout.

1) **Algebraic index is identical across GPUs.** In AllGather, each token is always broadcast to all GPUs and lands at the same index in the result tensor of each GPU. Algebraically, the only transformation from AllGather to Dispatch is that each token is sent only to a dynamically selected subset. Therefore, the *algebraic index*, i.e., the index in the resulting *algebraic tensor*, is still identical across GPUs.
2) **Memory layout is per-GPU managed and asymmetric.** Because only a subset of GPUs receives a given token, the algebraic tensor is fragmented and must be compacted into a dense *layout tensor* to be stored in memory. This compaction is performed through stacking tokens within each GPU, so the resulting *layout index*, i.e., the index in the layout tensor, is naturally asymmetric across GPUs.

These properties give us the answer to question above, as illustrated in Fig. 5(b): *with a lightweight target expert list and an algebraic-layout mapping, the algebraic index can represent multiple destinations of an operation.* It drives two design points of our proposed dynamic multimem addressing:

- **Customized Packet**: A customized packet carries a single multimem address, whose offset is the algebraic index, and a target expert list. Packet format, ISA, and microarchitecture of GPU and switch are extended for support.
- **Index Managing**: A hardware memory manager is proposed in Hub. This manager performs algebraic-layout index mapping to translate multimem address to virtual address, whose offset is layout index, for memory access.

Our design introduces only minor, non-intrusive modifications to the existing hardware and software stack, building upon current datapaths without altering any original functionalities. This ensures low design complexity and minimal overhead, while preserving full support for other workloads. Design details will be introduced in the following subsections.

B. Packet Format Extension

As introduced in Sec. III-A, a customized packet format should be supported to carry a single multimem address and an additional target expert list. We extend NVLink data-link packet format to support dynamic multimem addressing framework, as shown in Fig. 7. In *flit0*, we replace 64-bit address with: 1) 48-bit multimem address for algebraic index supporting 128TB address space, 2) 1-bit *stage* (Dispatch/Combine), and 3) 15-bit *target count*. Following *flit0*, *target extension* flits encode destination expert IDs (16 bits each, eight per flit). Subsequent byte-enable and payload flits remain unchanged. Compared with explicit addressing that embeds full destination addresses, adopting such a multimem-style format preserves header compactness for near-ideal payload efficiency.

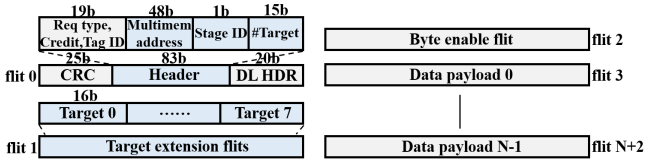

Fig. 7. Our extended data link layer packet format for DySHARP based on the original NVLink packet format. Packet has only a single multimem address and an additional target list.

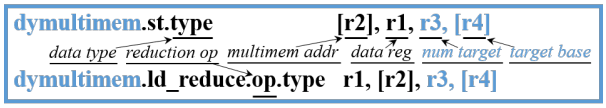

Fig. 8. ISA extension for dynamic multimem addressing. We derive dymultimem instructions based on multimem instructions, with two additional registers required to specify the target list of the operation.

C. ISA Extension

ISA should also be extended to provide a programming interface feasible for our packet extension. Because the request packet carries a single multimem address and a target expert list to support varying targets, ISA should provide this information, enabling source GPU to issue such a request packet.

Therefore, we introduce dymultimem instructions, an extension based on multimem instructions, as illustrated in Fig. 8. Extended instructions include `dymultimem.st` for multicast in Dispatch and `dymultimem.ld_reduce` for reduction in Combine. Similar to original multimem instructions, dymultimem instructions use a register (`r2`) as multimem address, whose offset is the algebraic index, and another register (`r1`) to hold the data operand for `.st` or receive the reduced value for `.ld_reduce`. Since NVLS's `multimem.ld_reduce` does not support weighted reduction and adding weighting would incur high hardware complexity, our `dymultimem.ld_reduce` retains reduction without weighting. Instead, we support weighted sum in Combine by applying weights in the epilogue of the preceding GEMM, before reduction. Concretely, each expert scales its output o_i by the gating weight w_i in GEMM-2's epilogue, so the subsequent unweighted reduction $\sum_i (w_i \cdot o_i)$ yields the desired weighted sum.

As extension, to fetch target expert list, each instruction additionally specifies `r3` for the target count and `r4` for the base address of a contiguous target list. Targets can be fetched from global or shared memory, where list is usually loaded to shared memory in advance to reduce overhead.

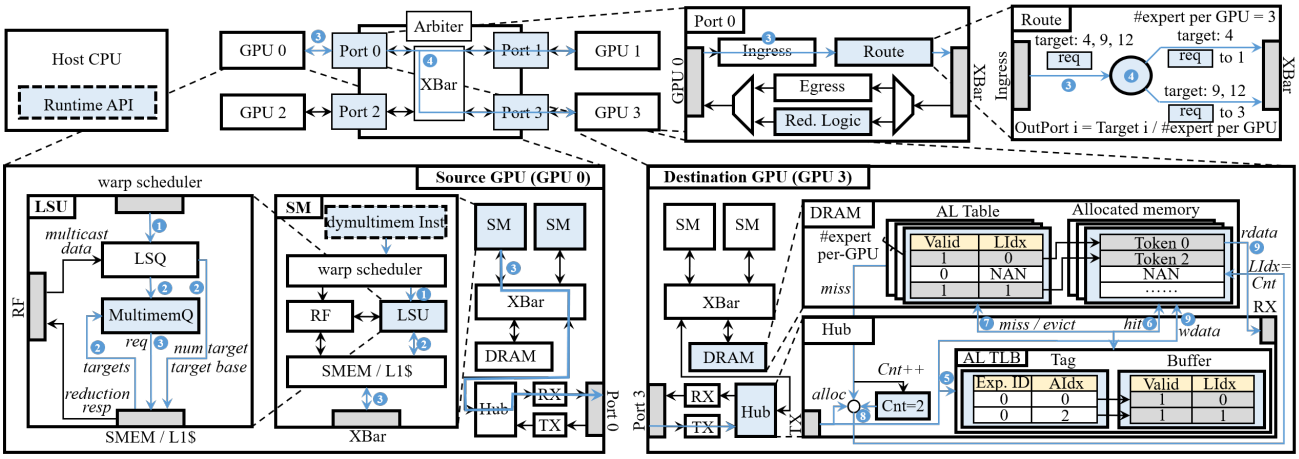

Fig. 9. Detailed architectural design and workflow of the dynamic multimem addressing framework. 1) Source GPU includes a new LSU design in SM to fetch targets for dynamic multimem instructions. 2) Switch enhances forwarding and reduction, aware of the target list. 3) Destination GPU introduces a hardware memory manager in the Hub, performing multimem-virtual translation through mapping algebraic index to layout index.

D. Microarchitecture Extension

As illustrated in Fig. 9, DySHARP introduces three components for the two design points, as introduced in Sec. III-A.

- For the customized packet, 1) source GPU[1] includes new LSU design in SM to support target fetching for dynamic multimem instructions. 2) Switch enhances the forwarding and reduction, aware of the target list.
- For the index managing, 3) destination GPU introduces a hardware memory manager in the Hub. This manager performs multimem-virtual translation through mapping algebraic index to layout index locally within each GPU.

1) Architectural Support in Source GPU: We extend the SM's LSU to fetch the target expert list and assemble dymultimem requests. Unlike regular memory instructions that read operands and immediately issue, a `dymultimem.*` first fetches its targets from memory. LSU uses target count and base pointer encoded in instruction to read the contiguous target list from shared or global memory. As shown in Fig. 9, we introduce a small *MultimemQ* separate from original LSQ to hold instructions that have issued their target fetching request. Once target list is ready, LSU issues the complete request packet to on-chip network and then to inter-GPU network.

2) Architectural Support in Switch: Within the switch, the *Route* module is augmented to forward dymultimem packets by their targets as shown in Fig. 9. For each target i, the output port is computed as $\text{OutPort}_i = \text{Target}_i/\#\text{expert\_per\_GPU}$. Switch replicates the request per output port and trims each replica to include only the targets to the port. For `dymultimem.ld_reduce` (Combine), the Reduction Logic records the number of targets associated with the request and decrements this counter as partial responses arrive; completion is detected when it reaches zero, which then returns the reduced result to the source GPU. This target-aware replication and completion tracking match the packet format in Sec. III-B.

[1]We call the GPU that a token resides on before Dispatch as source GPU of a token, and the GPU that a token is dispatched to as destination GPU.

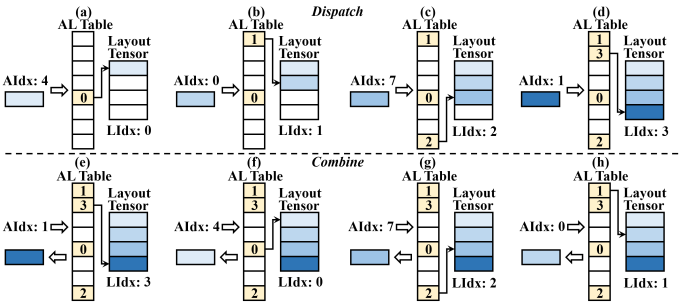

Fig. 10. Illustration of hardware memory manager workflow for Dispatch and Combine, where AIdx is algebraic block index and LIdx is layout block index. Manager allocates a layout block to a new algebraic block during Dispatch, and then shares the same algebraic-layout mapping for Combine.

3) Architectural Support in Destination GPU: At the destination GPU, a hardware memory manager performs algebraic-layout index mapping to translate the multimem address to a virtual address. This translation is performed before virtual-physical translation in GPU's Link MMU [14], [19].

This manager performs translation with the token vector as its granularity. *Dispatch:* During Dispatch, the manager dynamically allocates the layout block to the algebraic block for arriving tokens sent by `dymultimem.st` request packets. The layout block is allocated in an accumulative approach, ensuring the fragmented algebraic tensor to be stored in a dense layout tensor. *Combine:* During Combine, with the algebraic-layout mapping built during Dispatch, the manager translates the algebraic index of `dymultimem.ld_reduce` request packets to the layout index to load the data as responses.

AL Management with AL Table. The hardware memory manager performs algebraic-layout mapping (AL) management with an AL Table in GPU's DRAM. AL Table is depicted in Fig. 9, with each entry representing a mapping from an algebraic block to a layout block. The entry contains a `Valid` field to indicate whether a layout block has been allocated for this algebraic block, and a `LIdx` field to store the layout block index (LIdx). To find the layout block for an algebraic block, algebraic block index (AIdx) is used to index AL Table to

get its LIdx. When multiple experts reside on a single GPU, the AL Table incorporates multiple independent sub-tables, with each sub-table separately managing the layout tensor of a distinct expert. Each AL Table entry is 4B (1b `Valid` + 31b `LIdx`), resulting in a table size of $4\times$nToken bytes. Even when processing 1M tokens, this consumption is only 4MB per layer, which is small compared to GPU DRAM (40s-100s GB).

Fig. 10 illustrates how AL management is performed. *Dispatch:* Fig. 10(a)-(d) show the layout block allocation during Dispatch. When a new token, with an unseen AIdx of the `dymultimem.st` request packet, arrives, the manager allocates the next available layout block for it. The mapping is registered by writing the LIdx to the AIdx-th entry of the AL Table. A counter is adopted to track the next available layout block. *Combine:* Fig. 10(e)-(h) show the algebraic-layout mapping used by the Combine. Because the expert computation does not change the token order, the algebraic-layout mapping is unchanged before and after expert computation, leading the Combine to share the same AL Table as Dispatch. When a `dymultimem.ld_reduce` request packet arrives, the manager looks up the AL Table to get LIdx to be accessed.

MV Translation. Hardware memory manager performs multimem-virtual address (MV) translation based on algebraic-layout mapping. For multimem address $MAddr$, AIdx is $(MAddr - MBase)/bsize$. After AL management resolves LIdx, virtual address $VAddr$ is $VBase + LIdx * bsize + MAddr \% bsize$, where $MBase$ and $VBase$ are base addresses of multimem and virtual spaces. MV Translation is decoupled from AL management because Dispatch and Combine share the same algebraic-layout mapping but operate on different virtual address spaces.

AL TLB. Analogous to a conventional TLB, we introduce AL TLB to accelerate AL Table lookups. As shown in Fig. 9, AL TLB uses the concatenation of Expert ID and AIdx as its tag. Each entry stores this tag along with its AL Table entry. The tag section employs Content-Addressable Memory (CAM) for fast lookup. The resulting index from tag matching accesses the buffer section implemented with SRAM. During access, 1) on a hit, the LIdx is directly got. 2) On a miss, the AL Table is accessed and the entry is brought into the AL TLB. In our software implementation for Dispatch and Combine, elements within the same token vector are typically accessed contiguously. This access pattern has strong temporal locality for AL TLB lookups, enabling a high AL TLB hit rate.

4) Architectural Workflow: The workflow is depicted as Step ①–⑨ in Fig. 9. On source GPU, ① a dymultimem instruction enters LSQ, then ② LSU fetches its target list from memory and ③ issues complete dymultimem request through on-chip and inter-GPU networks to switch. ④ Switch computes output ports per target and forwards replicated packets. At destination GPU, ⑤ the request queries AL TLB: ⑥ on a hit, MV translation directly yields the virtual address; ⑦ on a miss, AL Table is accessed, with ⑧ new layout blocks allocated on first touch during Dispatch. ⑨ The translated virtual address is used for memory access: `dymultimem.st` writes data, and `dymultimem.ld_reduce` reads and returns a response that is aggregated in the switch.

```
// ... Allocate memory and prepare data ...
// class CUDymulticastObjectProp {... int bsize, int stage}
McProp.bsize = hsize; McProp.stage = 2; // Two sets of multimem regions (Dispatch and Combine)
cuDyMulticastCreate(McObj, McProp); // Create dynamic multicast object and add devices
for(int device = 0; device < num_devices; device++)
  cuMulticastAddDevice(McObj, device);
// Register memory spaces (multimem and virtual address space sizes are different)
for(int expert = 0; expert < num_experts; expert++) {
  cuDyMulticastBindAddr(McObj, 0, ntoken, mdispatch[expert], nactive[expert], 0); // Dispatch
  cuDyMulticastBindAddr(McObj, 0, ntoken, mcombine[expert], nactive[expert], 1); // Combine
}
cuMemMap();
// ... Execute Dispatch and Combine kernel ...
```

Fig. 11. Code snippet with our extended CUDA Runtime API. We extend the existing CUDA Runtime API to support dynamic multimem addressing.

E. Runtime Extension

We also extend the existing CUDA Runtime API for multicast object management to support the dynamic multimem addressing. Fig. 11 shows a CUDA code snippet utilizing the extended Runtime API. We introduce `CUDymulticastObjectProp` and its `cuDyMulticastCreate` function. The extension specifies the block size `bsize` and `stage`, the number of multimem region sets sharing the same algebraic-layout index mapping, as the vector length `hsize` and 2 (Dispatch and Combine), respectively. We also extend the API with `cuDyMulticastBindAddr` to specify the sizes of the multimem and virtual address space, as `ntoken` and `nactive[expert]` in our example. The `nactive[expert]` represents the number of tokens to be dispatched to a given expert. This value is determined by token routing, which is generated by gating network *before* Dispatch.

IV. Token-Centric Kernel Fusion

Building on dynamic multimem addressing, we address low bandwidth utilization under isolated dataflow scheduling, as introduced in Sec. II-D, with *token-centric kernel fusion*, which co-schedules the full *Dispatch-Computation-Combine* pipeline to co-execute Dispatch and Combine that have complementary communication patterns. Sec. IV-A outlines the key idea. Sec. IV-B2 presents a *token tracker* that captures fine-grained, token-level dependencies across operators. Sec. IV-C introduces a *token-centric scheduler* that exploits these dependencies to pipeline operators, improving bandwidth utilization.

A. Key Idea of Token-Centric Kernel Fusion

Token-centric kernel fusion treats the MoE layer as a *token-paced pipeline* rather than four isolated operators. As illustrated in Fig. 12(a), the insight is that readiness can be determined at *token/tile* granularity, so operation can be performed as soon as their inputs for a given token (or a tile of $tsize$ tokens) become available, without waiting for operator-wide completion. Concretely, by explicitly tracking these token-level dependencies and scheduling at *readiness boundaries*, Dispatch and Combine proceed *concurrently*.

As previewed in Sec. II-D, Fig. 4(a)–(f) provides a detailed illustration of how kernel fusion *translates the traffic reduction* of dynamic multimem addressing into *speedup*. Without dynamic multimem addressing, the two baselines in Fig. 4(a)(b),

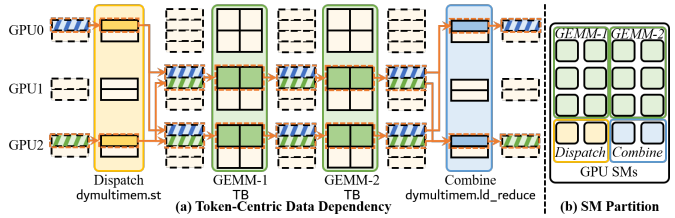

Fig. 12. (a) Token-centric data dependency chain across Dispatch, GEMM-1, GEMM-2, and Combine. (b) SM partition for pipelined execution.

i.e., DeepEP (baseline without overlap) and COMET (baseline with basic overlap), suffer from significant communication bottlenecks. Dynamic multimem addressing reduces GPU→switch traffic for Dispatch and switch→GPU traffic for Combine, yielding DySHARP-Basic (dynamic multimem addressing without overlap) and DySHARP-COMET (dynamic multimem addressing with basic overlap) in Fig. 4(c)(d). However, this traffic reduction does not directly lead to speedup because of the asymmetry between two directions in communication pattern. Hence we propose token-centric kernel fusion to translate traffic reduction into overall speedup. As depicted in Fig. 4(f), this is achieved by co-executing Dispatch and Combine concurrently, thereby merging complementary asymmetric communication patterns in Fig. 4(c)(d). This concurrent execution is fine-grained pipelining the *whole* Dispatch-Computation-Combine flow. Importantly, token-centric kernel fusion *alone* in Fig. 4(e) does *not* yield speedup over the SOTA baseline COMET, it must be *integrated* with in-switch computing together to unlock the full performance potential. We provide detailed experimental analysis in Sec. VI-A3.

B. Token Tracker

The token tracker is proposed to detect *readiness boundaries*. It detects when a tile of tokens from Dispatch becomes consumable for its consumer GEMM TBs, and when a token with its $topk$ expert outputs are ready for its Combine.

1) Token Tracker Design: The tracker monitors the dependency chains as illustrated in Fig. 12(a).

- **Dispatch⇒GEMM-1:** When $tsize$ dispatched tokens for an expert have arrived through `dymultimem.st`, the corresponding row of GEMM-1 TBs is ready and can be issued immediately. Each expert locally counts arrived tokens. When counter reaches $tsize$, a row of GEMM-1 TBs is marked ready to issue.
- **GEMM-1⇒GEMM-2:** A GEMM-2 TB row becomes ready when the corresponding GEMM-1 TB row completes. Tracker monitors TB completion of GEMM-1 and notifies the scheduler when a row of TB completes.
- **GEMM-2⇒Combine:** For each token, when all its $topk$ expert outputs are produced, Combine for this token can be executed via `dymultimem.ld_reduce`. When a GEMM-2 TB row finishes its $tsize$ outputs, it notifies the source GPU of these tokens. Source GPU counts the number of notifications received for each token, and is ready when counter reaches $topk$.

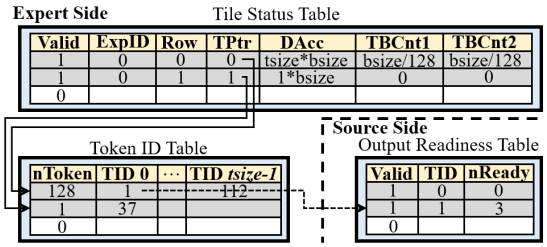

Fig. 13. Architecture design of token tracker. Token tracker introduces Tile Status Table for **Dispatch⇒GEMM-1** and **GEMM-1⇒GEMM-2** readiness tracking, and cooperates with proposed Token ID Table and Output Readiness Table to track **GEMM-2⇒Combine** readiness.

2) Token Tracker Architectural Support: To implement the above readiness tracking, the tracker uses three lightweight tables, as shown in Fig. 13.

Tile Status (TS) Table monitors the status of each $tsize$-token tile corresponding to a GEMM TB row. It tracks 1) the readiness of **Dispatch⇒GEMM-1**, 2) the readiness of **GEMM-1⇒GEMM-2**, and 3) the completion of GEMM-2 TB row to assist **GEMM-2⇒Combine** readiness tracking.

Each entry TS Table includes a `Valid` field to indicate if the entry is valid, an `ExpID` field to identify the expert that the $tsize$ tokens belong to, and a `Row` field to record the row of TB these tokens correspond to. 1) To track the readiness of **Dispatch⇒GEMM-1**, TS Table uses `DAcc` field to track the number of `dymultimem.st` access to the address region for these $tsize$ tokens. Reaching $tsize*bsize$ indicates the arrival of dispatched $tsize$ tokens, marking the GEMM-1 TB row ready to issue. 2) TS Table includes a `TBCnt1` field to track **GEMM-1⇒GEMM-2** readiness. This field tracks the number of completed GEMM-1 TB of this row. Completion of all TBs in this row indicates the readiness of the corresponding GEMM-2 TB row. 3) Similar to `TBCnt1`, `TBCnt2` counts the number of completed GEMM-2 TB of this row. The completion of the row starts the notification to source GPUs for **GEMM-2⇒Combine** readiness tracking. The TS Table resides on-chip and can be offloaded to DRAM on overflow.

Token ID (TID) Table and **Output Readiness (OR) Table** are for **GEMM-2⇒Combine** readiness tracking. TID Table records tokens of each token tile, which are the tokens to be notified at the completion of GEMM-2 TB row. It records the number of tokens in this tile as `nToken` and each token ID as `TID`, which is registered when allocating the layout block to the algebraic block. Due to size and low access frequency, this table is placed in DRAM. OR Table receives this notification to track the readiness of each token for Combine. OR Table entry adopts a counter `nReady` to track the readiness of token `TID`. OR Table resides on-chip and can be offloaded to DRAM.

When TS Table detects the completion of a GEMM-2 TB row, tracker notifies source GPUs of these tokens for their readiness. Tracker indexes TID Table with `TPtr` to collect completed token IDs, and then sends notifications to source GPUs of these tokens. When source GPU receives a notification, it increments `nReady` of the token entry. `nReady` reaching $topk$ indicates the readiness of this token for Combine.

To guarantee visibility of produced data before accessing, all state updates in token tracker's tables are performed after written data is visible to all SMs, i.e., when the acknowledge is detected, indicating stored data has arrived at LLC/DRAM.

C. Token-Centric Scheduler

The scheduler realizes token-paced pipeline based on readiness detection of the tracker. It allows `Dispatch` and `Combine` to run concurrently, merging asymmetric traffic of in-switch computing. This scheduler is implemented in software through megakernel that employs persistent thread blocks (TBs) to bypass hardware TB scheduler [26]. Original TBs are represented as *tasks*, and the action of *issuing a TB to an SM* is emulated by a persistent TB fetching a task from the task list.

1) SM Partitioning: As shown in Fig. 12(b), to achieve pipelining, SMs are partitioned into four groups dedicated to Dispatch, GEMM-1, GEMM-2, and Combine. A modified TB scheduler issues TBs of each kernel to its SM group. GEMM-1 and GEMM-2 can share SMs when one has no ready TB.

2) Readiness-Gated Schedule: Consistent with Sec. IV-A, operation is gated by readiness besides resource availability. To check readiness for synchronization, the kernel polls the field indicating readiness in the token tracker's tables using a dedicated load instruction within a loop until they are ready:

- `GEMM-1`/`GEMM-2`: a row of TB is issued only when the tracker marks the corresponding row *ready* based on TS Table and the target SM group has capacity.
- `Combine`: communication kernels query token readiness, i.e., if the `nReady` of OR entry reaches $topk$, before issuing `dymultimem.ld_reduce` for that token.

As shown in Fig. 4(f), because readiness is checked at token/tile granularity, MoE layer is executed as a token-paced pipeline. Dispatch, dominating GPU→switch, and Combine, dominating switch→GPU, naturally run in parallel, merging asymmetric pattern to improve bandwidth utilization.

V. Experimental Methodology

A. Hardware Configuration

In our experiment, we simulate the NVIDIA GH200 NVL32 [33], a 32-GPU system interconnected via nine NVSwitch with a fully connected fat-tree topology. We integrate BookSim2 [16] and our customized Accel-Sim [18] to simulate our system in a cycle-accurate approach, with each GPU configured based on the NVIDIA H200 specifications [32]. For DeepSeek-V3, we extend the latest version of Accel-Sim, which supports basic Hopper features, to simulate high-performance FP8 kernels. To enable multi-GPU simulation, we support concurrent execution across GPUs connected through a switch-based network through BookSim2.

NVLink is modeled using real device parameters of NVLink 4.0 [34]. The bidirectional bandwidth of NVLink is configured to 900 GB/s, and the latency of a single NVLink is configured to 250ns, where the round-trip latency is 1 µs. Flit size is set as 16B. For our modeled NVSwitch, each input port provides sixteen 256-depth virtual channels, with eight for requests and eight for responses. Port reduction buffer size is set to 64KB. For architectural support of dynamic multimem addressing, the MultimemQ in LSU consists of 32 entries, and AL TLB in Hub is configured to 512 entries. For token-centric kernel fusion, both TS Table and OR Table have 1024 entries. Our simulator is validated against DGX-H100, with average errors within 6% for GEMM and DeepEP communication operators across diverse shapes/volumes.

B. Benchmark

Name	Hidden Size	MoE Hidden Size	Attention Heads	Sequence Length	Number of Experts	$topk$ Candidates
Small (S)	2048	512	32	2048	64	{8, 16, 32}
Medium (M)	4096	1024	64	4096	128	{8, 16, 32}
Large (L)	7168	2048	128	8192	256	{8, 16, 32}

TABLE I
MODEL CONFIGURATIONS ADOPTED IN EVALUATION.

DeepSeek-V3 [6] stands as one of the most competitive MoE-based LLMs. We therefore refer DeepSeek-V3 for our evaluation. Table I details the model configurations adopted in our evaluation. In addition to the official DeepSeek-V3 model configuration, denoted as *Large* (L), we configure two additional model sizes: *Small* (S) and *Medium* (M). The number of activated experts, $topk$, is 8 in DeepSeek-V3. We also evaluate $topk = 16/32$ to cover broader sparsity ranges, which may be potentially adopted in larger future models. Our evaluation focuses on communication-heavy MoE training, with data parallelism for attention layers and expert parallelism for MoE layers within a NVL32 node. For end-to-end training, the 16-way pipeline parallelism is adopted across 16-NVL32 nodes [6]. Following ByteDance's observation for typical training jobs [1], [57], we model the token distribution across experts as a normal distribution with a standard deviation (std) of 0.032.

C. Baseline

DySHARP is evaluated against seven baselines: 1) **DeepEP** [59] is the state-of-the-art communication library for Dispatch and Combine, where no in-switch computing is utilized. 2) **NVLink SHARP (NVLS)** [19] is the existing in-switch computing solution for static collective operations. Dispatch/Combine are replaced with AllGather/Reduce-Scatter as a workaround. 3) **FasterMoE** [10] and 4) **Tutel** [12] are coarse-grained computation-communication overlapping solutions for MoE. 5) **CCFuser** [53] and 6) **COMET** [57] are fine-grained overlapping solutions, supporting Dispatch-GEMM and GEMM-Combine overlapping. 7) **DualPipe** [6] is an overlap strategy designed for cross-node pipeline.

VI. Experimental Results

A. End-to-End and MoE-Layer Performance

1) End-to-End and MoE-Layer Speedup: Fig. 14 presents end-to-end model training speedup achieved by DySHARP compared with baselines across various model configurations and $topk$. This evaluation includes both attention and MoE layer, and covers both forward and backward propagation. We denote configuration `Config` with $topk = k$ as `Config-k`. DySHARP achieves speedups of up to 2.31×, 5.12×, 2.11×, 1.98×, 1.85×, 1.79×, and 1.88× over DeepEP, NVLS, FasterMoE, Tutel, CCFuser, COMET, and DualPipe respectively,

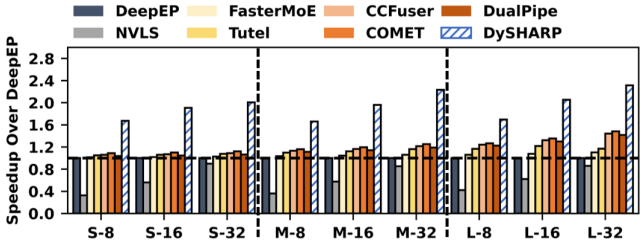

Fig. 14. End-to-end model training speedup across different configurations.

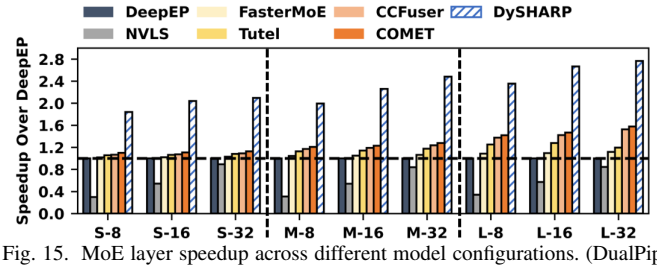

Fig. 15. MoE layer speedup across different model configurations. (DualPipe is excluded because it is model-level cross-layer optimization.)

with geometric means of 1.93×, 3.38×, 1.84×, 1.72×, 1.63×, 1.59×, and 1.66×. Fig. 15 further isolates performance comparison for communication-intensive MoE layer, encompassing Dispatch-Computation-Combine. DualPipe is excluded because it is model-level cross-layer optimization. Compared with other six baselines, DySHARP achieves speedups of up to 2.77×, 6.93×, 2.48×, 2.32×, 2.01×, and 1.94×, respectively, with geometric means of 2.26×, 4.25×, 2.14×, 1.96×, 1.84×, and 1.78×. This demonstrates DySHARP's significant performance advantage, attributed to dynamic multimem addressing for redundant data transfer elimination and token-centric kernel fusion to merge asymmetric communication.

2) Discussions and Analysis: DySHARP outperforms DeepEP, FasterMoE, Tutel, CCFuser, COMET, and DualPipe primarily by eliminating redundant data transfers and reducing memory management overhead. Unlike these baselines with communication redundancy discussed in Sec. II-B, DySHARP leverages dynamic in-switch multicast and reduction to eliminate this redundancy, boosting performance. It also avoids software-controlled memory management and associated metadata transmission, e.g., token arrival counts, further reducing overhead. Fine-grained computation–communication overlap is also an advantage over baselines.

DySHARP's performance advantage over the existing in-switch computing solution, NVLS, is from eliminating useless data transfers. This approach of replacing dynamic communication with static counterparts incurs large amounts of useless data transfer. In contrast, DySHARP can natively support dynamic communication without useless transfer.

3) Ablation Studies for Speedup Source Analysis: We quantitatively validate the speedup sources analyzed in Fig. 4(a)–(f). In addition to DeepEP, COMET, and DySHARP, we further implement three variants for ablation study: 1) DySHARP-Basic (Fig. 4(c)): dynamic multimem addressing without computation–communication overlap; 2) DySHARP-COMET (Fig. 4(d)): DySHARP-Basic with COMET's overlap; and 3) kernel fusion only (Fig. 4(e)): token-centric fusion without dynamic multimem addressing.

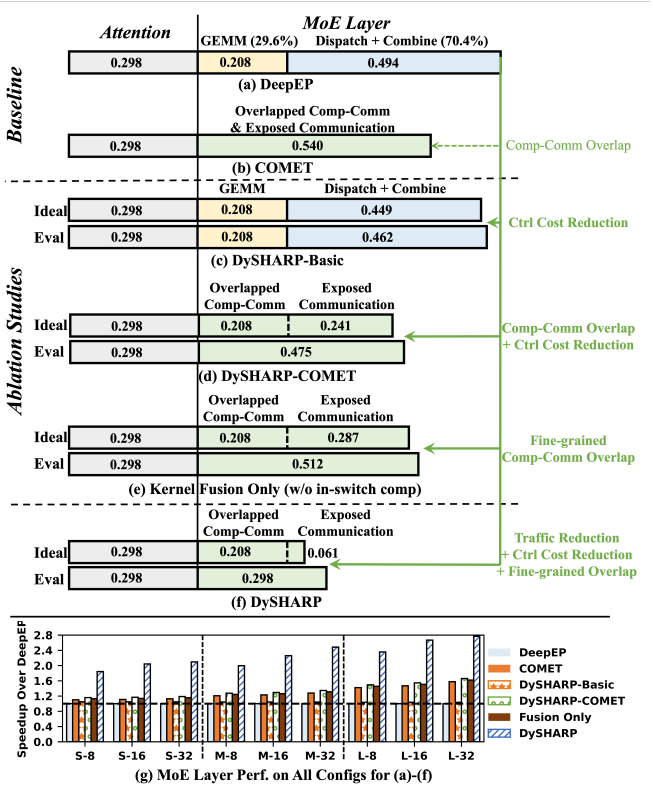

Fig. 16. Quantitative time breakdown (normalized to DeepEP) and ablation studies on official DeepSeek-V3 configuration (L-8), validating Fig. 4(a)-(f).

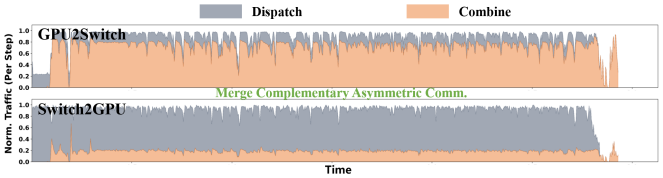

Fig. 17. Illustration of merging complementary asymmetric communication.

Fig. 16(a)–(f) shows the time breakdown on DeepSeek-V3 (Large-8). In Fig. 16(a)(b), DeepEP and COMET exhibit a severe communication bottleneck. With dynamic multimem addressing, DySHARP-Basic and DySHARP-COMET reduce traffic but do not directly lead to speedup as shown in Fig. 16(c)(d). This problem is due to asymmetric traffic reduction between the two directions. As an integral solution, DySHARP in Fig. 16(f) utilizes token-centric kernel fusion to merge complementary asymmetric communication by co-executing Dispatch and Combine concurrently, transforming traffic reduction enabled by dynamic multimem addressing into speedup. This merging of complementary communication can be observed in Fig. 17. Moreover, kernel fusion *alone* in Fig. 16(e) *cannot* provide speedup over the SOTA baseline COMET, it must be integrated together with in-switch computing to unlock full potential. We also evaluate on all configurations, with results shown in Fig. 16(g).

B. Detailed Performance Analysis

In this section, we individually analyze the effectiveness of dynamic multimem addressing and token-centric kernel fusion.

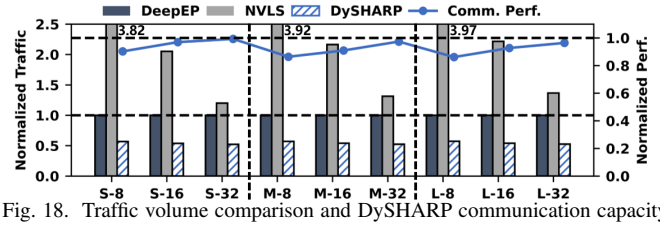

Fig. 18. Traffic volume comparison and DySHARP communication capacity.

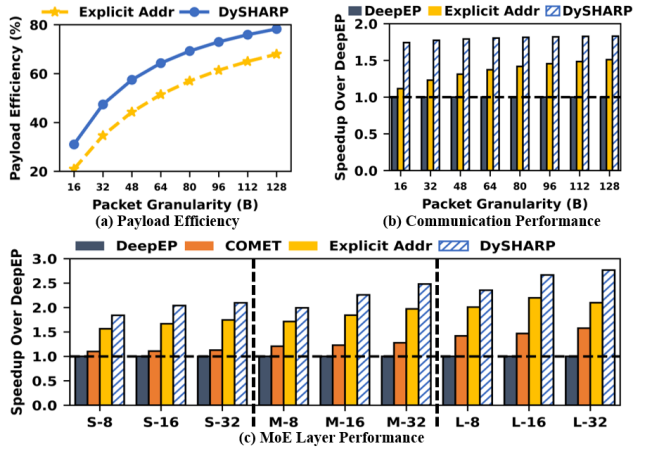

Fig. 19. Comparison between DySHARP and explicit addressing on (a) payload efficiency, (b) communication, and (c) MoE layer performance.

1) Impact of Dynamic Multimem Addressing: Dynamic multimem addressing reduces redundant data transfers in dynamic communication via in-switch computing, while avoiding useless data transfers present in existing in-switch computing solutions. Fig. 18 compares the data transfer traffic of DeepEP, NVLS, and DySHARP, demonstrating DySHARP's effectiveness in significantly reducing data movement. Due to the substantial volume of useless data transfers, applying NVLS as a workaround results in increased data movement compared to DeepEP. DySHARP reduces traffic by nearly 50% compared to DeepEP by eliminating redundant transfers. To inspect the communication capability of dynamic multimem addressing, we concurrently execute Dispatch and Combine operators without computation to measure the pure communication performance. Fig. 18 demonstrates the pure communication performance normalized to the ideal calculated with traffic volume and bandwidth. DySHARP can, on average, achieve over 90% performance of the ideal, indicating the high performance of such a dynamic multi-destination operation.

We analyze advantage of dynamic multimem addressing compared to straightforward explicit addressing that explicitly encodes all destinations within request packet. Fig. 19(a) compares payload efficiency (the proportion of data flits to total transmitted flits) under different data transfer granularities when targeting 8 destinations. Results demonstrate DySHARP's consistently higher payload efficiency than explicit addressing. Fig. 19(b) further compares performance of pure communication operators, highlighting the gain from high payload efficiency. We also adapt token-centric kernel fusion for explicit addressing and evaluate MoE layer performance. Fig. 19(c) shows results, validating DySHARP's advantage. These results verify discussion in Sec. III-A.

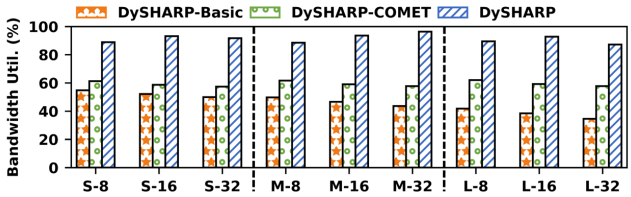

Fig. 20. Bandwidth utilization comparison. Token-centric kernel fusion improves utilization over non-overlap and the SOTA overlap solutions.

2) Impact of Token-Centric Kernel Fusion: Token-centric kernel fusion improves overall bandwidth utilization by enabling token-paced pipeline of the Dispatch-Computation-Combine workflow. Fig. 20 compares the bandwidth utilization of full DySHARP against DySHARP-Basic and DySHARP-COMET. Without token-centric kernel fusion, DySHARP-Basic exhibits low bandwidth utilization due to non-overlapped computation-communication and asymmetric communication. While DySHARP-COMET achieves overlap, isolated Dispatch and Combine are still asymmetric. DySHARP with token-centric kernel fusion merges asymmetric Dispatch and Combine, transforming traffic reduction into speedup.

C. Sensitivity Analysis

1) Performance of Different Numbers of GPUs: We evaluate the performance of DySHARP against baselines across different system scales. Using Small-8 and Medium-8 model configurations, we compare DySHARP with DeepEP and COMET across GPU counts of 4-64. Fig. 21 presents our evaluation on MoE layer. We simulate the 64-GPU node as an extension of NVL32, where the only difference is the interconnect. We simulate such a system with doubled number of NVSwitch (18 NVSwitch). Each NVSwitch has 64 ports, and each port is connected to a GPU, providing full bandwidth for the GH200 chip that has 18 ports. The results show that as the number of GPUs increases, DySHARP consistently outperforms both DeepEP and COMET, with the gap progressively widening. This highlights DySHARP's strong scalability and demonstrates its potential for future larger-scale SuperPODs.

2) Performance of Different Sequence Lengths: We further evaluate the performance of DySHARP against baselines across varying sequence lengths. Fig. 22 presents the comparison results on MoE layer under sequence lengths of 1024-16384. Results demonstrate that DySHARP achieves the shortest execution time regardless of sequence length. As the length increases, the execution times of both DeepEP and COMET rise rapidly, while DySHARP's execution time increases more moderately. This indicates that DySHARP's advantage over baselines becomes more pronounced with longer sequences.

3) Performance of Different Token Distribution: The number of tokens routed to each device is different. Therefore, we evaluate sensitivity to token distribution. Following evaluation setup of ByteDance's COMET [57], we vary standard deviation of token distribution across experts from 0.01 to 0.05, based on normal distribution $std = 0.032$ for a typical training job as introduced in Sec. V-B. Result demonstrates that DySHARP always achieves remarkable speedups over baselines on MoE layer, regardless of token distribution variations.

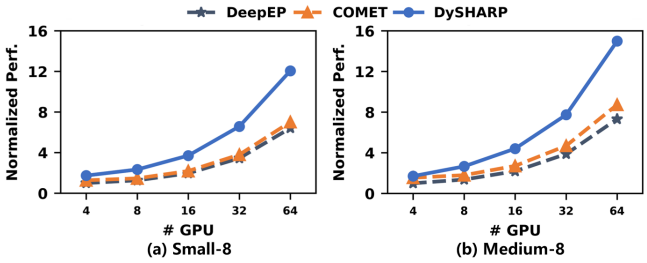

Fig. 21. Performance sensitivity to the number of GPUs.

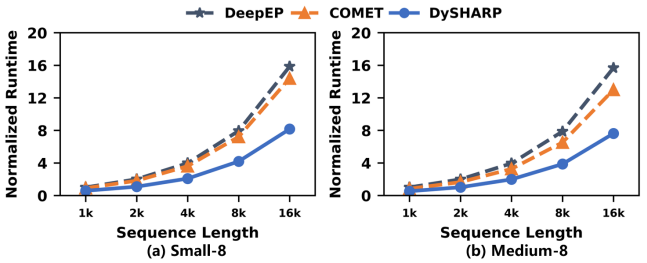

Fig. 22. Performance sensitivity to the sequence length.

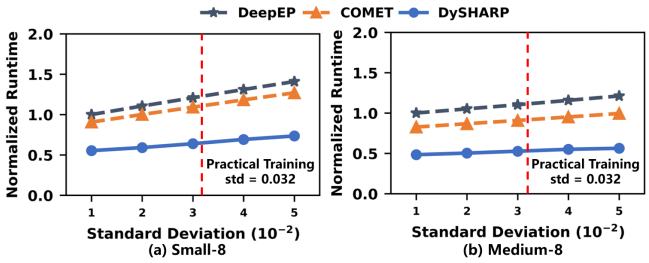

Fig. 23. Performance sensitivity to the token distribution for training.

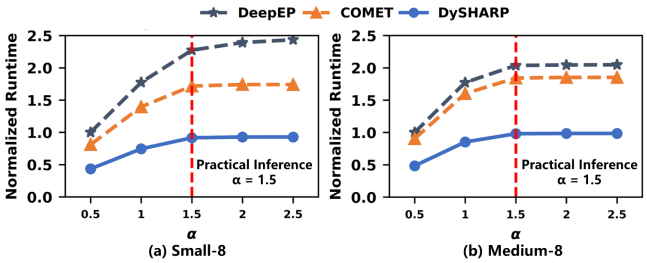

Fig. 24. Performance sensitivity to the token distribution for inference.

We also evaluate token distribution during inference, different from training [22]. Our preliminary study reveals a power-law distribution, consistent with recent work [47] showing that inference token distribution can be modeled as power-law with $\alpha \approx 1.5$. Accordingly, we model inference token distribution as a power-law with α of 0.5-2.5. Fig. 24 shows that, while imbalance prolongs all methods, DySHARP consistently achieves substantial speedup under inference distributions.

D. Hardware Overhead

We evaluate the hardware overhead of our architectural supports under TSMC 12nm technology [51]. To evaluate hardware overhead, we implement our components in RTL and synthesize them using Synopsys Design Compiler® [48]. SRAM macros are generated with the Memory Compiler of library [51]. The tables are implemented as 16-bank dual-port SRAM (1R1W) to meet DySHARP's concurrent read/write requirements. For the switch, by building upon the datapath of existing NVLS, our extension is a lightweight control

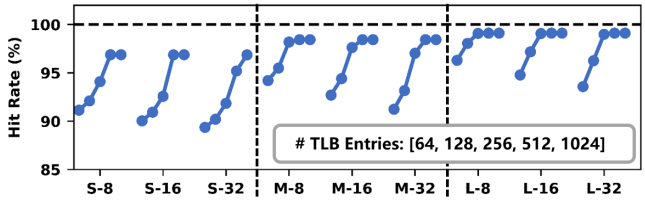

Fig. 25. Design space exploration of AL-TLB.

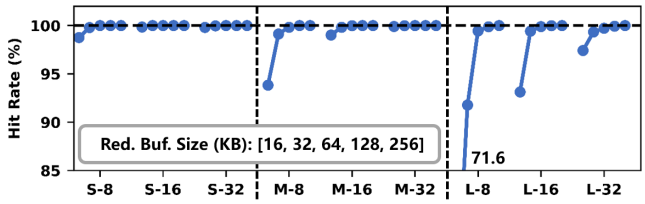

Fig. 26. Design space exploration of reduction buffer.

logic for routing calculation. This enhancement negligibly adds only one cycle to the datapath, without affecting data forwarding. Area overhead of this logic is less than $0.01mm^2$. This overhead is less than 0.1% of NVSwitch die [13], [19]. For the GPU, the additional architectural supports require only $0.198mm^2$, which is about 0.024% of the H100 GPU die area. The evaluation indicates that our architectural supports are feasible for hardware implementation.

We further explore design space of DySHARP by evaluating hit rates of AL-TLB in Hub and reduction buffer in switch under different sizes. For reduction buffer, a hit means a packet does not trigger eviction. Fig. 25 and 26 show the hit rates of AL-TLB and reduction buffer when varying the sizes. It indicates that 512-entry is a sweet spot for AL-TLB, which can achieve near-ideal hit rates while maintaining small overhead. For the reduction buffer, the sweet spot is 64KB, guaranteeing almost no eviction that increases traffic.

VII. Discussion

A. Evaluation of End-to-end Inference

We further evaluate DySHARP for end-to-end inference, covering both prefill and decode stages. Prefill, like training, is communication-intensive and benefits from DySHARP traffic reduction. Although decode is memory-bound with small batches, its latency sensitivity makes DySHARP's fine-grained synchronization and reduced software control cost impactful. Results in Fig. 27, where LLM decodes 512 tokens after prefill, confirm DySHARP's superior inference performance.

B. Evaluation on Other Models and Other Platform

Name	Hidden Size	MoE Hidden Size	Attention Heads	Sequence Length	Number of Experts	*topk*
GPT-OSS-120B	2880	2880	64	4096	64	4
Qwen3-235B	4096	1536	128	4096	128	8

We evaluate DySHARP for other leading MoE models, including GPT-OSS-120B [40] and Qwen3-235B [55], as shown in table. Results in Fig. 28 demonstrate DySHARP's superior end-to-end performance on diverse models.

We evaluate on GH200 NVL32 because single-node systems integrate an increasing number of GPUs [33], [36], [39] [33],

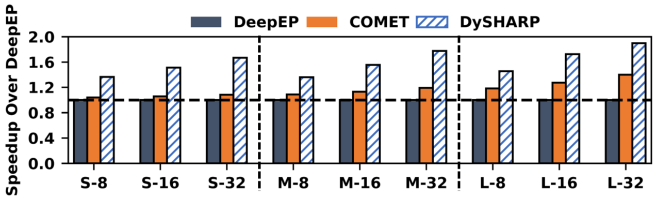

Fig. 27. End-to-end speedup for inference.

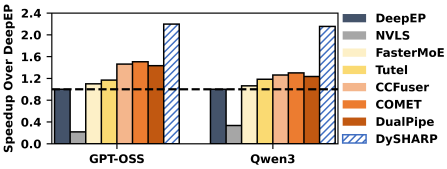

Fig. 28. Evaluation of other leading MoE models.

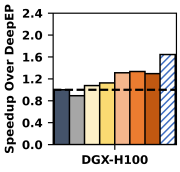

Fig. 29. Evaluation on other platform.

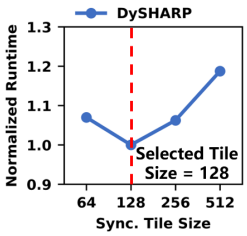

Fig. 30. Study on our optimal kernel fusion tile size.

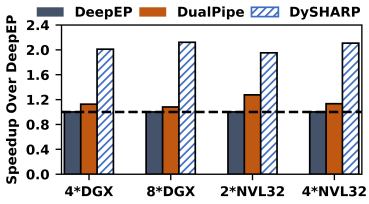

Fig. 31. Evaluation of DySHARP's multi-node extension.

[36], [39]. DySHARP also applies to regular small nodes like DGX-H100 (8 GPUs). We perform end-to-end evaluation of Large-8 configuration on DGX-H100. Results in Fig. 29 confirm DySHARP's advantage on regular small nodes.

C. Study on Tile Size Choice for Kernel Fusion

Token-centric kernel fusion adopts the synchronization tile size of 128, the minimum granularity that preserves computation utilization, as it matches the GEMM tile size of 128. A smaller tile would force a suboptimal GEMM tile size and increase synchronization overhead, while a larger tile would coarsen overlap. We validate this on the Small-8 configuration, where smaller models are more sensitive to synchronization granularity. Results in Fig. 30 confirm our choice.

D. Extension to Multi-Node System

InfiniBand (IB) [38] networks with Quantum Switch exhibit similar communication redundancy, making multi-node extension feasible. Motivated by the call for a unified network interface [6], [58], our extension abstracts the whole cluster as a shared memory system: programmers keep using `dymultimem.st` and `dymultimem.ld_reduce` for cross-node in-switch computing, where NVSwitch and Quantum IB Switch coordinate global routing. Taking multicast (`dymultimem.st`) as an example, the source GPU duplicates the request: one copy goes to NVSwitch for intra-node delivery, and the other is sent out-of-node, via extended IBGDA [37] for translation between two communication models, to the IB Switch, which multicasts it to all target nodes. At each remote node, the GPU with the same intra-node ID forwards the request via NVSwitch to destination GPUs. `dymultimem.ld_reduce` is similar. Small NVLink packets are aggregated before entering IB to improve performance [28]. Details are omitted for space.

We perform preliminary evaluation comparing to DeepEP and DualPipe for end-to-end training (Large-8). We apply expert parallelism across 4/8*DGX-H100 and 2/4*NVL32, and also adopt 16-way pipeline parallelism. Nodes are interconnected with IB. Results in Fig. 31 show the benefit of extended DySHARP over multi-node baselines. While DeepEP reduces inter-node traffic via hierarchical communication *across* intra- and inter-node networks, it cannot eliminate redundancy *within* switch-connected networks (Fig. 1) in both intra- and inter-node networks, which DySHARP effectively eliminates.

VIII. Related Work

Numerous prior works have proposed leveraging switches to accelerate operations such as AllReduce [5], [8], [9], [19], [23], [24], [45], [56], database queries [21], [50], MapReduce [3], [4], and DLRM communication [11]. However, most efforts focus on traditional inter-node networks. Only NVLS [19], CAIS [56], and TRACI [11] target inter-chip interconnects with memory semantics. Nevertheless, NVLS and CAIS only support static collectives, and TRACI is tailored to DLRM. Crucially, neither supports the dynamic communication inherent in MoE.

An increasing number of works aim to optimize the communication bottleneck in MoE. Some works [20], [30], [46], [59] provide highly optimized libraries. Recent works [12], [29], [44], [59] also consider hierarchical communication. Beyond optimizing the communication operators in isolation, several works [10], [12], [53], [57] explore overlapping computation and communication to reduce exposed communication overhead. FasterMoE [10] and Tutel [12] introduce coarse-grained overlap pipeline. COMET [57] and CCFuser [53] further achieve fine-grained overlap to enhance performance. Some works [2], [15], [42], [54] target overlapping in dense LLM but are inapplicable to MoE. None leverages in-switch computing to reduce communication redundancy in MoE.

IX. Conclusion

We propose DySHARP, an integral dynamic in-switch computing solution for MoE acceleration. It introduces dynamic multimem addressing and token-centric kernel fusion to achieve fully exploited in-switch computing capabilities. With such a full-stack solution, DySHARP achieves up to 1.79× speedup compared to the SOTA solution.

Acknowledgement

We sincerely thank the anonymous ISCA'26 reviewers and shepherd for their valuable suggestions. This work is supported by National Natural Science Foundation of China (NSFC) grant (62502305), Natural Science Foundation of Shanghai (NSFS) grant 25ZR1402275, the Shanghai QiYuan Innovation Foundation QY2025-QN-SJTU-011, Hong Kong Research Grants Council (RGC) CRF-YCRG C6003-24Y, and Shanghai Qi Zhi Institute Innovation Program SQZ202316.

REFERENCES

[1] ByteDance, "Flux: Fine-grained computation-communication overlapping gpu kernel library." *https://github.com/bytedance/flux*, 2025.

[2] C. Chen, X. Li, Q. Zhu, J. Duan, P. Sun, X. Zhang, and C. Yang, "Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 178–191.

[3] G. Chen, G. Zeng, and L. Chen, "P4com: In-network computation with programmable switches," *arXiv preprint arXiv:2107.13694*, 2021.

[4] L. Chen, G. Chen, J. Lingys, and K. Chen, "Programmable switch as a parallel computing device," *arXiv preprint arXiv:1803.01491*, 2018.

[5] D. De Sensi, S. Di Girolamo, S. Ashkboos, S. Li, and T. Hoefler, "Flare: Flexible in-network allreduce," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–16.

[6] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang, R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

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

[8] N. Gebara, "In-network aggregation for shared machine learning clusters," *Proceedings of Machine Learning and Systems (MLSys)*, 2021.

[9] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, L. Levi, A. Margolin, T. Ronen, A. Shpiner, O. Wertheim, and E. Zahavi, "Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. IEEE, 2016, pp. 1–10.

[10] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 120–134.

[11] G. Huang, H. Li, L. Qin, J. Huang, Y. Kang, Y. Ding, and Y. Xie, "Traci: Network acceleration of input-dynamic communication for large-scale deep learning recommendation model," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1880–1893.

[12] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram, H. Chau, P. Cheng, F. Yang, M. Yang, and Y. Xiong, "Tutel: Adaptive mixture-of-experts at scale," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 269–287, 2023.

[13] A. Ishii, D. Foley, E. Anderson, B. Dally, G. Dearth, L. Dennison, M. Hummel, and J. Schafer, "Nvswitch and dgx-2," in *Hot Chips*, 2018.

[14] A. Ishii and R. Wells, "The nvlink-network switch: Nvidia's switch chip for high communication-bandwidth superpods," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 2022, pp. 1–23.

[15] A. Jangda, J. Huang, G. Liu, A. H. N. Sabet, S. Maleki, Y. Miao, M. Musuvathi, T. Mytkowicz, and O. Saarikivi, "Breaking the computation and communication abstraction barrier in distributed machine learning workloads," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 402–416.

[16] 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.

[17] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[18] 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.

[19] 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.

[20] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[21] A. Lerner, R. Hussein, P. Cudre-Mauroux, and U. eXascale Infolab, "The case for network accelerated query processing." in *CIDR*, 2019.

[22] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating distributed {MoE} training and inference with lina," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 945–959.

[23] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 279–291.

[24] S. Liu, Q. Wang, J. Zhang, W. Wu, Q. Lin, Y. Liu, M. Xu, M. Canini, R. C. Cheung, and J. He, "In-network aggregation with transport transparency for distributed training," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 376–391.

[25] 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.

[26] L. Ma, Z. Xie, Z. Yang, J. Xue, Y. Miao, W. Cui, W. Hu, F. Yang, L. Zhang, and L. Zhou, "Rammer: Enabling holistic deep learning compiler optimizations with {rTasks}," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 881–897.

[27] Meta, "The llama 4 herd: The beginning of a new era of natively multimodal ai innovation," *https://ai.meta.com/blog/llama-4-multimodal-intelligence*, 2025.

[28] H. Muthukrishnan, D. Lustig, O. Villa, T. Wenisch, and D. Nellans, "Finepack: Transparently improving the efficiency of fine-grained transfers in multi-gpu systems," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 516–529.

[29] X. Nie, P. Zhao, X. Miao, T. Zhao, and B. Cui, "Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system," *arXiv preprint arXiv:2203.14685*, 2022.

[30] NVIDIA, "Doubling all2all performance with nvidia collective communication library 2.12," *https://developer.nvidia.com/blog/doubling-all2all-performance-with/nvidia-collective-communication/library-2-12/*, 2022.

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

[32] NVIDIA, "Nvidia h200 tensor core gpu." *https://www.nvidia.com/en-us/data-center/h200*, 2023.

[33] NVIDIA, "One giant superchip for llms, recommenders, and gnns: Introducing nvidia gh200 nvl32." *https://developer.nvidia.com/blog/one-*

giant-superchip-for-llms-recommenders-and-gnns-introducing-nvidia-gh200-nvl32, 2023.

[34] NVIDIA, "Introduction to nvidia dgx h100/h200 systems." *https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html*, 2024.

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

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

[37] NVIDIA, "Improving network performance of hpc systems using nvidia magnum io nvshmem and gpudirect async." *https://developer.nvidia.com/blog/improving-network-performance-of-hpc-systems-using-nvidia-magnum-io-nvshmem-and-gpudirect-async*, 2025.

[38] NVIDIA, "The nvidia quantum infiniband platform." *https://www.nvidia.com/en-us/networking/products/infiniband*, 2025.

[39] NVIDIA, "Inside the nvidia rubin platform: Six new chips, one ai supercomputer." *https://developer.nvidia.com/blog/inside-the-nvidia-rubin-platform-six-new-chips-one-ai-supercomputer*, 2026.

[40] OpenAI, "Gpt-oss." *https://github.com/openai/gpt-oss*, 2025.

[41] OpenAI, "Introducing gpt-5." *https://openai.com/index/introducing-gpt-5*, 2025.

[42] 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.

[43] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International conference on machine learning*. PMLR, 2022, pp. 18 332–18 346.

[44] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3505–3506.

[45] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtárik, "Scaling distributed machine learning with {In-Network} aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[46] L. Shen, Z. Wu, W. Gong, H. Hao, Y. Bai, H. Wu, X. Wu, J. Bian, H. Xiong, D. Yu, and Y. Ma, "Se-moe: A scalable and efficient mixture-of-experts distributed training and inference system," *arXiv e-prints*, pp. arXiv–2205, 2022.

[47] Z. Su, Q. Li, H. Zhang, W. Ye, Q. Xue, Y. Qian, Y. Xie, N. Wong, and K. Yuan, "Unveiling super experts in mixture-of-experts large language models," *arXiv preprint arXiv:2507.23279*, 2025.

[48] Synopsys, "Design compiler® rtl synthesis." *https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html*, 2021.

[49] Y. Tang, Y. Yin, Y. Wang, H. Zhou, Y. Pan, W. Guo, Z. Zhang, M. Rang, F. Liu, N. Zhang, B. Li, Y. Dong, X. Meng, Y. Wang, D. Li, Y. Li, D. Tu, C. Chen, Y. Yan, F. Yu, R. Tang, Y. Wang, B. Huang, B. Wang, B. Liu, C. Zhang, D. Kuang, F. Liu, G. Huang, J. Wei, J. Qin, J. Ran, J. Li, J. Zhao, L. Dai, L. Li, L. Deng, P. Qin, P. Zeng, Q. Gu, S. Tang, S. Cheng, T. Gao, T. Yu, T. Li, T. Bi, W. He, W. Mao, W. Huang, W. Liu, X. Li, X. Yu, X. Wu, X. He, Y. Du, Y. Xu, Y. Tian, Y. Wu, Y. Huang, Y. Tian, Y. Zhu, Y. Li, Y. Wang, Y. Gai, Y. Li, Y. Luo, Y. Ni, Y. Sun, Z. Chen, Z. Liu, Z. Liu, Z. Tu, Z. Ding, and Z. Zhan, "Pangu ultra moe: How to train your big moe on ascend npus," *arXiv preprint arXiv:2505.04519*, 2025.

[50] M. Tirmazi, R. Ben Basat, J. Gao, and M. Yu, "Cheetah: Accelerating database queries with switch pruning," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2407–2422.

[51] TSMC, "Tsmc 16nm and 12nm process technologies." *https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_16_12nm*, 2017.

[52] 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.

[53] H. Wang, Y. Xia, D. Yang, X. Zhou, and D. Cheng, "Harnessing inter-gpu shared memory for seamless moe communication-computation fusion," in *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 2025, pp. 170–182.

[54] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang, S. Kumar, T. Guo, Y. Xu, and Z. Zhou, "Overlap communication with dependent computation via decomposition in large deep learning models," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2022, pp. 93–106.

[55] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[56] C. Zhang, Q. Zhang, Z. Zhou, Y. Diao, H. Wang, Z. Zhou, Z. Tu, Z. Li, G. Sun, Z. Song *et al.*, "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.

[57] S. Zhang, N. Zheng, H. Lin, Z. Jiang, W. Bao, C. Jiang, Q. Hou, W. Cui, S. Zheng, L.-W. Chang, Q. Chen, and X. Liu, "Comet: Fine-grained computation-communication overlapping for mixture-of-experts," *arXiv preprint arXiv:2502.19811*, 2025.

[58] C. Zhao, C. Deng, C. Ruan, D. Dai, H. Gao, J. Li, L. Zhang, P. Huang, S. Zhou, S. Ma, W. Liang, Y. He, Y. Wang, Y. Liu, and Y. Wei, "Insights into deepseek-v3: Scaling challenges and reflections on hardware for ai architectures," in *2025 ACM/IEEE 52nd Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2025, p. 1731–1745.

[59] C. Zhao, S. Zhou, L. Zhang, C. Deng, Z. Xu, Y. Liu, K. Yu, J. Li, and L. Zhao, "Deepep: an efficient expert-parallel communication library," https://github.com/deepseek-ai/DeepEP, 2025.