

PARE: A Power and Area-Efficient Memory Controller for High-Bandwidth DRAM

Andre Green
NVIDIA, USA

University of California, Berkeley, USA
andre_green@berkeley.edu

Ben Keller
NVIDIA, USA
benk@nvidia.com

Niladrish Chatterjee
NVIDIA, USA
nchatterjee@nvidia.com

David Nellans
NVIDIA, USA
dnellans@nvidia.com

Mike O'Connor
NVIDIA, USA
moconnor@nvidia.com

Borivoje Nikolić
University of California, Berkeley, USA
bora@berkeley.edu

Abstract—High-throughput processors, like GPUs, fundamentally depend on high-bandwidth DRAM to feed their parallel compute engines. To maximize effective bandwidth, modern GPU memory controllers sort incoming requests to exploit DRAM row-buffer locality across interleaved streams from parallel compute engines. Historically, this deep sorting has improved both bandwidth utilization and energy efficiency by reducing unnecessary DRAM row activations. In this work, we show that the shifting access patterns of modern ML and HPC workloads, combined with improving DRAM access energy efficiency, are eroding the benefits of these deep-sorting memory controller designs. By combining architectural simulation with high-level synthesis, we demonstrate that simple energy-optimized memory controllers can achieve near-parity performance with prior complex sorting controllers for many important datacenter workloads. Such ultra-efficient designs are increasingly critical as emerging technologies like high-bandwidth stacked 3D DRAM will likely render the overhead of traditional sorting memory controllers prohibitively large for tightly integrated GPU systems. Drawing on these insights, we first present PARE, an efficiency-optimized GPU memory controller that removes row-aware transaction sorting entirely. PARE remains close to a sorting controller for most workloads, but exposes losses in two bandwidth-sensitive kernels caused by premature row-miss commitment. We then present MS-PARE, which retains only a narrow row-hit check to avoid this hazard. Relative to the conventional sorting controller, MS-PARE reduces area by 73.3% and average controller energy across 14 workloads by 55%, while maintaining near-parity performance and supporting the extreme DRAM bandwidth demanded by next-generation GPU architectures.

Index Terms—Memory controller, DRAM, GPU architecture, energy efficiency, high-bandwidth memory, memory scheduling, ML inference

I. INTRODUCTION

LLM inference is limited by memory bandwidth. Large models stream weights and KV-cache state from DRAM at a rate that GPU compute has outgrown, and successive HBM generations have narrowed that gap only partially [1], [2]. Future systems will need more bandwidth, and two integration paths are emerging to provide it. The nearer-term path keeps the 2.5D package but moves the HBM base die to an advanced logic node, enabling higher per-pin rates within the same package geometry [3]–[6] (F-HBM). The longer-term path stacks

DRAM directly on, or under, the GPU die, as in Samsung’s zHBM concept and recent 3D-stacked research designs [6]–[9] (F-3DMem). Unlike 2.5D integration, where connections are limited by bump pitch around the die perimeter, 3D integration exposes the full bonding area, enabling many more I/Os at lower per-pin rates and shortening the path between the DRAM array and the GPU. 3D integration introduces challenges (thermal management, yield, process support, and thinned-die reliability) but those are not the focus here. Both paths share a common bottleneck: the memory controller.

Both F-HBM and F-3DMem increase bandwidth by exposing more of the DRAM array through more, finer-grained channels, the same direction explored in Fine-Grained DRAM [10] and later 3D-stacked designs [9]. Throughput processors can exploit this channel-level parallelism well [9], [10], but controller cost scales with it. More channels mean more per-channel controllers, and their area becomes a hard constraint on the base die or along the edge of the compute die [11]. Contemporary HBM4 already moves the base die to a 4 nm logic process to accommodate higher channel bandwidth, doubled DQ TSVs, and timing-calibration and test logic [4]. Accommodating hundreds of conventional sorting controllers could therefore place substantial pressure on the available logic area in future memory systems.

Energy is an equally hard constraint. Figure 2 shows that a full-row HBM4 access costs about 2.60 pJ/bit, while F-3DMem reduces that to about 0.30 pJ/bit. The conventional sorting memory controller (SMC) we evaluate consumes 0.120 pJ/bit on average; the Power and Area-Efficient memory controller (PARE) reduces that to 0.056 pJ/bit (Section VIII-B). At 5 TB/s, those two controller points correspond to about 4.8 W and 2.2 W. While the SMC accounts for only 4.4% of total memory-access energy in HBM4 (combining DRAM-device and controller energy), its contribution rises to 29% in F-3DMem. Future architectures must therefore reduce both DRAM-device energy and the energy spent in the memory controller as bandwidth scales.

PARE addresses this by avoiding the cost of aggressive row-buffer locality mining. Conventional controllers derive

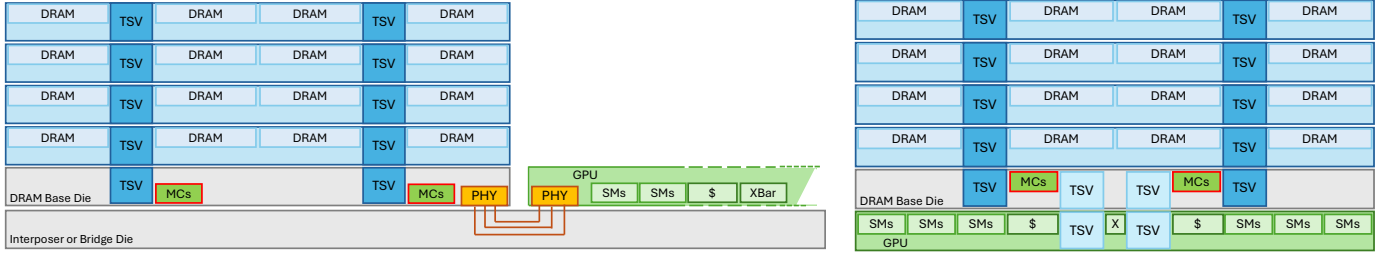


Fig. 1. F-HBM 2.5D and F-3DMem high-bandwidth stacked DRAM system overview.

bandwidth from two sources: harvesting row hits and exploiting bank-level parallelism. PARE removes most of the first while preserving the second. As row-activation energy improves across DRAM generations, the energy returned by extra row hits shrinks. Once natural bytes-per-activate are already sufficient for the available bank-level parallelism, additional row-hit extraction has limited throughput value. Our analysis across both HPC and ML workloads shows that many kernels already expose enough natural locality for a simpler controller to remain effective. Moving to a row-granularity interface, as in RoMe [12], is not a general GPU solution, since many workloads do not consume an entire row per activation and forcing full-row transfers incurs substantial overfetch. PARE keeps a conventional fine-grained interface and reduces controller overhead through large-access-granularity requests, shallow queues, and simpler scheduling logic. We make the following contributions in this work:

- We identify why conventional sorting memory controllers scale poorly into F-HBM and F-3DMem, quantify when additional row-hit mining stops paying off in energy and throughput, and analyze row-access behavior across both HPC and ML workloads to show why a simpler controller is often sufficient while row-granularity transfer is not a universal GPU solution.
- We present PARE, a reduced-complexity per-channel GPU memory controller that removes the main structures responsible for row sorting and starvation management in a conventional design, while maintaining the performance of ML inference. We then add a minimal row-hit guard to create Minimally-Sorting PARE (MS-PARE), which restores performance parity for the bandwidth-sensitive HPC kernels while retaining most of PARE’s area and energy savings.
- We implement PARE, MS-PARE, and a sorting baseline in HLS/RTL, carry all three through physical synthesis, and evaluate them in simulation. Relative to the sorting baseline, MS-PARE reduces controller area by 73.3% and average controller energy across 14 workloads by 55%, while maintaining near-parity GPU performance with a geomean change of -0.03% and a worst-case loss of 0.71%.

II. BACKGROUND

In this section, we briefly describe trends that will take us to higher bandwidth DRAM architectures and the resulting

impacts on memory controller architectures used in throughput processors in order to set the stage for PARE.

A. High-Bandwidth DRAM Properties

Future high-bandwidth DRAM systems can follow two paths. One is to continue scaling package-based HBM through more aggressive 2.5D integration. The other is to move to a 3D organization in which the DRAM stack is placed directly on the compute die. This paper studies both directions through two projected future high-bandwidth DRAM systems shown in Figure 1, a 2.5D approach (F-HBM) and a 3D approach (F-3DMem). These use a common DRAM architecture that doubles the bandwidth over HBM4, but differ in how the memory stacks are integrated with the processor. F-HBM follows the evolutionary HBM approach of increasing stack bandwidth within a 2.5D package. It builds on emerging HBM4 stacks which utilize a logic process base layer for high-performance PHYs and allow integration of memory controller logic [4], [6]. F-3DMem takes a different approach, stacking the DRAM directly on the GPU die, similar to Samsung’s zHBM concept and the Raptor 3D-DRAM accelerator [6], [7]. A base layer exists between the DRAM dies and the GPU that interfaces between the GPU and the DRAM stacks, decoupling the TSV layouts of the DRAM and the connections to the GPU below. The memory controllers can reside on this interface die.

a) Bandwidth Scaling through Channel Proliferation:

These future DRAMs increase stack bandwidth by increasing the number of channels, while narrowing each one. This organization is similar in spirit to Fine-Grained DRAM [10]. Rather than depending on bank-group interleaving to keep a wide channel busy, F-HBM and F-3DMem effectively create an independent channel from each bank group of an HBM4 DRAM. This results in four times as many independent channels to the memory controller. Since each channel is limited to the bandwidth of a single bank group (without the ability to interleave among multiple bank groups), the per-channel bandwidth is one half that of HBM4. Overall, 4 times more channels at half the bandwidth doubles the per-stack bandwidth of HBM4.

b) **Right-Sized Row Activations:** Row activation energy scales with row size, so smaller rows reduce overfetch [10], [13], [14]. We cut the row-size in half relative to HBM4 to 512 B in our future DRAM configurations. Prior fine-grained designs have pushed the row size down more aggressively to 256 B [9], [10], [14]. We take a more conservative approach, as

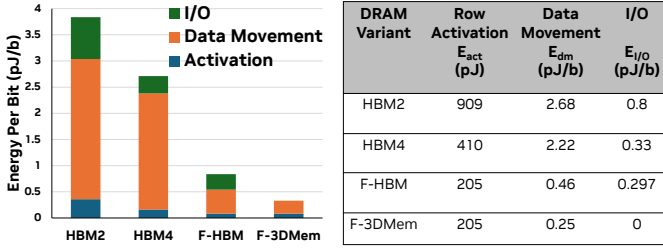


Fig. 2. DRAM access-energy breakdown.

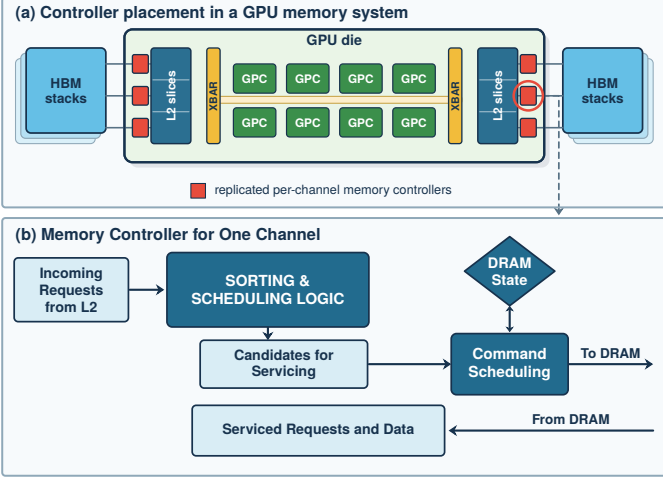


Fig. 3. GPU memory-controller placement and per-channel request flow. The expanded controller shows sorting and scheduling, DRAM-state feedback, command issue, and the serviced-data return path.

the HPC workloads studied in this paper generate an average of 317.8 bytes per activate and the LLM workloads studied generally read the entire 512 B row on each activate.

c) Implications for the Memory Controller: The main consequence of this future DRAM organization is on the controller. F-HBM and F-3DMem have 4 times as many independent memory channels. Replicating a conventional sorting controller for each of those channels is costly in terms of area. Furthermore, as the energy efficiency of the DRAMs shown in Figure 2 improve, particularly with F-3DMem, the energy overheads of the memory controller become a more significant portion of memory access energy.

We will show that the workloads we consider rarely benefit from memory scheduling approaches that reorder requests searching for row-hits in the memory access stream. These factors motivate PARE: an energy-efficient controller designed for many-channel future memory systems without the overheads of a conventional sorting scheduler.

B. Memory Controller Architectures

Figure 3 shows a GPU, which issues memory requests from SMs, through the on-chip interconnect and L2 slices, to a set of memory controllers. Each controller serves one DRAM channel, where it sorts requests, schedules the appropriate DRAM commands, and issues them to the DRAM.

The primary objective of a GPU memory controller is to maximize bandwidth. Conventional controllers do this in two ways. First, they harvest row-buffer locality by reordering requests so that multiple accesses to an open row can be issued back-to-back. This avoids repeated ACT/PRE sequences and increases useful transfers per activation [15], [16]. Second, they exploit bank-level parallelism by interleaving commands across independent banks so that one bank can make progress while another waits on DRAM timing constraints. These two mechanisms are complementary, but they do not cost the same to implement.

Most of the controller complexity sits on the row-hit side. To find row hits, the controller must track row IDs, search across candidate transactions, and often reorder requests aggressively while still enforcing fairness. It must also manage read-to-write transitions carefully by batching writes and draining them when the write queue reaches a high-water mark to avoid frequent bus turnarounds and their associated $tWTR$ penalties [17]. Periodic refresh adds another layer of media management, requiring REF commands within $tREFI$ limits while minimizing interference with live traffic.

Bank-level parallelism is preserved later in the pipeline, at the per-bank queues and command scheduler. Once transactions have been assigned to banks, the command scheduler arbitrates across those bank queues each cycle and issues whichever row or column commands are legal. This interleaving across banks keeps the bus occupied while individual banks wait on $tRCD$, $tRAS$, tRP , or refresh delays. As we show in Section VIII-B, a large fraction of the area and energy of a conventional controller is devoted to the first mechanism, namely finding and exploiting row hits. PARE targets that asymmetry: it reduces the cost of row-hit harvesting while preserving the command-scheduler machinery needed to exploit bank-level parallelism.

III. MEMORY CONTROLLER DESIGN CONSIDERATIONS

Conventional GPU memory controllers assume that aggressive row-buffer locality mining is worth its complexity. In this section, we revisit that assumption, to understand the limits of the benefits of row hit harvesting.

A. When Does Row-Buffer Search Pay Off?

A conventional GPU memory controller, such as SMC, spends energy to mine row-buffer locality that is not explicit in the incoming request order. That tradeoff is worthwhile only if the additional bytes-per-activate (BpA) recovered by reordering is large enough to offset the controller energy spent to find it. We quantify that break-even point to motivate the PARE design family (Sections V and VI).

a) Energy Model: We decompose total memory access energy per bit into three additive components, following [9]:

$$E_{total}(BpA) = \underbrace{\frac{E_{act,row}}{8 BpA}}_{\text{row activation}} + \underbrace{E_{dm}}_{\text{data movement}} + \underbrace{E_{mc}}_{\text{controller}} \quad (1)$$

where BpA is bytes transferred per row activation, $E_{act,row}$ is the energy of a row activation (pJ), E_{dm} is the per-bit

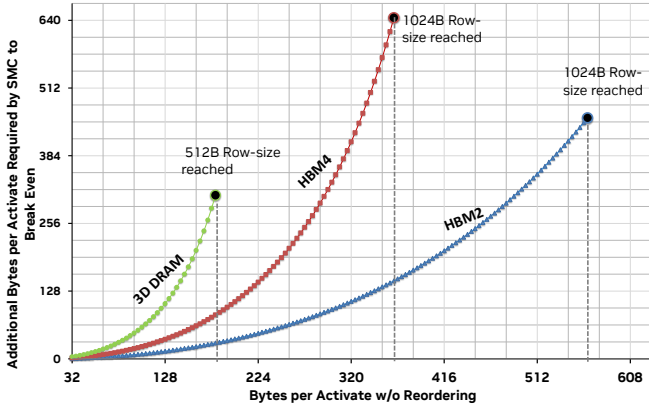


Fig. 4. Additional bytes-per-activate that SMC must find to match PARE’s energy.

data movement cost (column select lines through TSVs and interposer to the GPU, independent of row-buffer reuse), and E_{mc} is the per-bit controller energy. We set the memory controller target from the projected memory-system power budget rather than from the PARE implementation result. F-3DMem consumes approximately 0.30 pJ/bit for the DRAM access. Limiting controller energy to 20% of that cost gives a target of $E_{mc}^{PARE} = 0.06$ pJ/bit, or 2.4 W at a 5 TB/s stack-equivalent interface. We compare this target with a conservative 0.15 pJ/bit sorting-controller point, corresponding to the upper end of the SMC traces in Section VIII-B. The resulting scheduler complexity premium is $\Delta E_{mc} = 0.09$ pJ/bit. The DRAM energy terms across generations are summarized in Figure 2. At equal row-buffer reuse, PARE is always lower energy by ΔE_{mc} . The question is therefore whether SMC can find *enough additional* reuse to pay back that overhead.

b) Break-Even Reuse: Let BpA_{nat} be the natural bytes per activation seen by the controller, and let BpA_{smc} be the higher reuse, if any, recovered after SMC reordering. Equating $E_{total}^{SMC}(BpA_{smc}) = E_{total}^{PARE}(BpA_{nat})$ and solving for BpA_{smc} yields the break-even reuse.

At 32 B, one additional atom from the same row is enough to justify SMC. At 128 B, roughly two more atoms are required. At 256 B, the SMC must recover about 465 B per activation, or roughly seven additional atoms from the same row. The return from row-hit mining therefore shrinks quickly as the natural access granularity rises. The denominator $E_{act,row} - 8 BpA_{nat} \Delta E_{mc}$ goes to zero at about 569 B of natural BpA. Beyond that point, the SMC cannot recover enough additional reuse to break even.

c) Implication of Improved Row Energy: Figure 4 shows the same break-even relation across three DRAM generations. The endpoints in the figure mark the point at which the SMC would need to read the full row to break even; beyond those points, no amount of reordering can recover enough reuse to offset the SMC controller’s energy overhead.

As row activation energy falls, the benefit of mining extra row hits falls with it. In HBM2 ($E_{act,row} = 909$ pJ), approxi-

TABLE I
BREAK-EVEN REUSE FOR HBM4.

Natural BpA	Break-even SMC BpA	Extra 32 B atoms
32 B (1 atom)	34 B	1 (trivial)
128 B (GPU cache line)	165 B	≈ 2
256 B	465 B	≈ 7

mately two additional 32B atoms fetched from a stream with $BpA_{nat} = 256$ B is enough to justify SMC, with break-even at 321 BpA. In HBM4, the same 256 B stream now breaks even if SMC finds an additional 209 B. In projected 3D DRAM (205 pJ, shown dashed), the 256 B break-even point is not possible, as it would require reading more than the entire 512 B row. SMC can still pay for itself for some $BpA_{nat} = 128$ B streams, but the profitable region is narrow.

Emerging 3D-stacked DRAM architectures [7], [8], [18] also reduce E_{dm} , but that term is common to both controllers and cancels in the break-even derivation. The dominant trend in Figure 4 is therefore the steady reduction in $E_{act,row}$: as each generation makes row activation cheaper, the energy returned by additional row hits shrinks, and the case for paying extra controller energy becomes more limited.

B. Performance through Bytes per Activate

The case for aggressive row-buffer locality mining is not just about energy. It is also commonly justified as necessary for performance. If the controller can increase BpA, it can keep the DRAM data bus busy while one bank waits for its next activation to become legal. The relevant question is how much data each activation must supply to keep the bus busy while that bank is unavailable. If b banks have useful pending work, the required BpA per bank falls with that bank-level parallelism. Once the natural BpA of a stream exceeds this threshold, additional row hits do not contribute to throughput.

a) Minimum Useful BpA for F-3DMem: Consider a channel with 32 B atoms, row-cycle time tRC , read-to-read spacing $tCCD$, and b banks with useful pending work. Over one tRC interval, the bus must supply $tRC/tCCD$ bursts. If those bursts are distributed across b banks that can make progress in parallel, the per-bank threshold is

$$BpA_{min}(b) = \frac{tRC}{tCCD} \cdot \frac{32 B}{b}. \quad (2)$$

For the F-3DMem timing parameters from §VII-A ($tRC = 45$ ns and $tCCD = 1.6$ ns), this yields $BpA_{min}(2) = 450$ B, $BpA_{min}(4) = 225$ B, $BpA_{min}(6) = 150$ B, and $BpA_{min}(8) \approx 113$ B.

b) Implication for Workloads: With only two or four banks carrying useful work, the required BpA is still high, so recovering extra row hits can matter. With six or eight banks, the threshold drops to about 150 B and 113 B. In that regime, a natural 128 B access is already close to, or above, the useful range. Additional row-buffer hits mainly add slack beyond what is needed to keep the bus busy. Our workload characterization suggests that many HPC and

ML kernels operate in this region. For those workloads, the throughput return from aggressive reordering is small even before considering its energy and area cost. This factor is the other motivation for our approach. Once BpA is high enough for the available bank-level parallelism, more row-hit extraction has no performance upside.

C. Factors Affecting Bytes per Activate

The locality seen by the memory controller is set by two factors. The first is the natural access order generated by the workload. Fine-grained accesses to semi-random locations produce low row-buffer locality (e.g. GUPS), whereas strided accesses to contiguous address ranges produce high row-buffer locality (e.g. STREAM).

The second factor is the physical-address mapping used by the GPU memory system. That mapping must balance two competing goals: spreading traffic across all available DRAM channels and banks, while also keeping nearby addresses within the same row long enough to preserve row-buffer locality. The key knob is channel-interleave granularity. It determines how large a contiguous block of addresses remains within one DRAM row before the mapping moves to a different channel and bank. If the interleave granularity is too small, contiguous accesses are broken up across channels too early and row-buffer locality drops.

For our work, we follow the address mapping approach used in previous work and map contiguous 512 B regions to a single row before interleaving to another channel [10]. This preserves row-buffer locality for streaming accesses while still distributing traffic across the many channels exposed by F-3DMem.

D. Opportunities in Emerging Workloads

In practice, many AI and HPC kernels already present naturally large, unsorted accesses to the memory controller, so a simplified controller can inherit enough BpA directly from the request stream instead of expending area and energy to reconstruct it through aggressive reordering.

As illustrated in Figure 5, the naturally occurring BpA of many kernels is already at or above the 128 B cache-line granularity, and in many cases substantially higher. That observation is exactly what enables PARE. If the incoming request stream already provides enough bytes per activate to cover the row-cycle delay across the available banks, then the controller does not need deep queues and complex associative logic to mine further locality. Instead, it can rely on the native access granularity, combined with bank-level parallelism, to deliver most of the available performance at far lower controller area and energy.

IV. BASELINE SORTING MEMORY CONTROLLER (SMC)

We use a conventional sorting memory controller (SMC) as the baseline for PARE. SMC represents the design choice made when both row-buffer locality and bank-level parallelism must be recovered inside the controller: its transaction scheduler performs deep row-hit search, while its command

scheduler arbitrates across banks subject to DRAM timing constraints.

a) Input Queue: The SMC incorporates many of the architectural advancements utilized by conventional memory controllers. Focusing on the digital scheduling logic of the controller, memory transactions are enqueued from the L2 interface into the scheduler. Each request enters a 32-depth unified input queue to handle traffic congestion, as it awaits the transaction scheduler. By buffering memory requests, the transaction scheduler can focus on selecting from a smaller subset of transactions that are actively ready to enter the command scheduler and issue commands to DRAM. The input queue follows a simple First-Come-First-Serve (FCFS) policy, which dequeues the first-in transaction once a valid slot becomes available in the transaction scheduler.

b) Transaction Scheduler: The SMC follows the First-Ready-FCFS (FR-FCFS) policy common in GPU controllers [15], [16]. In the framing of Section II-B, this scheduler is responsible for the first of the two bandwidth mechanisms: harvesting row hits. The transaction scheduler tries to form *transactions* consisting of several commands that can issue back-to-back to the same DRAM row [19]. The transaction scheduler dominates complexity: 8 read + 8 write, 16-deep bank-aligned queues with per-bank CAMs, linked-list age tracking, and a row-hit streak starvation counter. To prevent workloads that have significant row-level contention from locking out older transactions, a maximum of 8 sequential memory requests that share row-buffer locality can be issued to a bank queue prior to forcing the oldest transaction to be selected for a given CAM. The associative CAMs independently traverse the age tracker to locate row-hits, achieving parallel row-hit detection, maximizing the probability of locating pending transactions that can amortize the latencies caused by the DRAM timing constraints. The controller enables dynamic request reordering, which is essential for maximizing our effective bandwidth.

c) Command Scheduler: The deep bank-aligned queues from the transaction scheduler are sufficient to handle the row-hit search, allowing for a total of 8, 2-depth FCFS bank queues. The queues contain metadata that updates when the next command becomes issuable based on the timing parameters specified by the system and state of the scheduler. These timing parameters are initialized into an array of registers and used to calculate the minimum latency required for future row and column commands [9]. The command scheduler arbitrates across the bank queues on each cycle and selects a command to issue to DRAM, using an FR-FCFS selection process. This stage is what preserves bank-level parallelism by interleaving progress across banks once transactions have already been steered into their bank queues. Both a row and column command can be selected and issued to DRAM in the same cycle through separate issue ports. Transactions do not dequeue from the bank queues until all four 32 B atoms have been serviced after issuing their column command.

The state of each bank queue is tracked by the finite state machine (FSM) in Figure 7. After activating a row, a 128 B transaction issues at least four 32 B column commands before

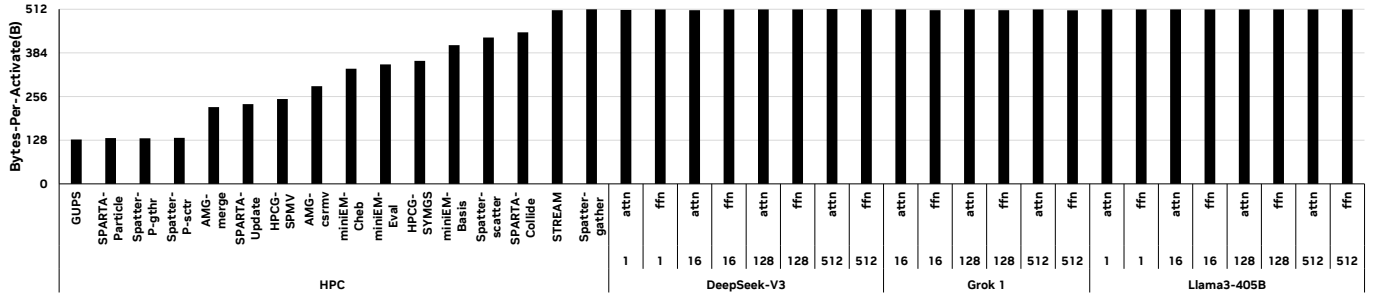


Fig. 5. Observed unsorted bytes-per-activate in representative HPC kernels, and the attention (ATTN) and feed forward network (FFN) layers for 3 different LLM models at various batch-sizes. Many workloads already expose naturally high BpA without controller-side reordering.

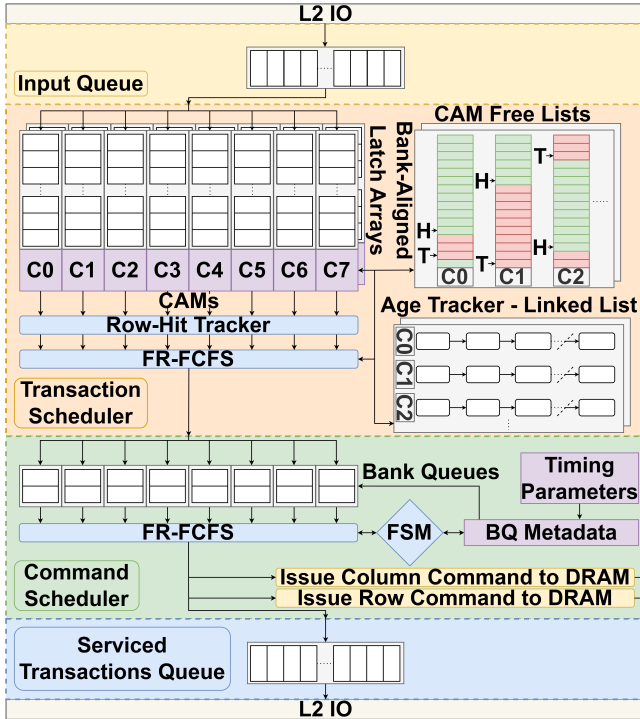


Fig. 6. Sorting Memory Controller (SMC) architecture.

the bank can return to the ACT state. If the next transaction in the bank queue maps to the active row, its column commands can begin after satisfying $tCCD$, without another activation. Otherwise, the FSM advances through PRE and ACT subject to the corresponding DRAM timing constraints.

Since DRAM is volatile, the refresh scheduler handles issuing refresh commands to restore the electrical charge in the storage capacitors. The command scheduler utilizes per-bank refresh, REFpb, with round-robin arbitration across the 8 banks every $tREFpb$ to maintain effective memory bandwidth without stalling the memory bus. The remaining 7 banks continue to find issuable commands to send to DRAM, ensuring that memory bandwidth does not degrade during these periodic refresh periods.

d) Serviced Transactions Queue: Upon issuing the final column command, the transaction leaves the bank queue and

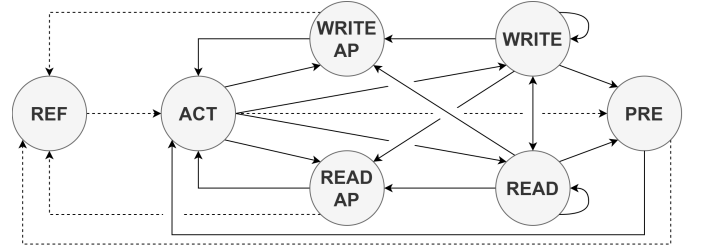


Fig. 7. Command-scheduler finite state machine. Each bank queue transitions through ACT, column (RD/WR), and PRE states under DRAM timing constraints; READ AP and WRITE AP denote read and write commands with auto-precharge, and dotted transitions show the per-bank refresh path.

is enqueued into the serviced transaction queue. This queue serves a similar purpose to the input queue, and acts as a buffer for the transaction while the CAS latency between the column command being issued and the data appearing on the bus is satisfied. Operating in FCFS order, once the completion cycle has been reached, the transaction returns to the L2 interface, completing the life cycle of a memory request in the scheduler.

e) Shared Reliability Logic: All evaluated controllers integrate the same Hsiao SEC-DED codec [20] at the 32 B DRAM interface to protect data traversing the interposer, complementing the on-die ECC that guards against intra-stack faults [21]. Hsiao's construction restricts the parity-check matrix to odd-weight columns, allowing double-bit errors to be detected from an even syndrome without a separate global parity bit. Single-bit errors produce an odd syndrome that locates and corrects the fault, while the balanced XOR fan-in closes timing at 625 MHz with limited combinational depth. Because this block is identical in SMC, PARE, and MS-PARE, it does not contribute to the area or energy differences reported in Section VIII-B.

f) Sorting Memory Controller Complexity: SMC contains the machinery to pursue both sources of bandwidth: aggressive row-hit harvesting in the transaction scheduler and bank-level parallelism in the command scheduler [16]. The transaction-level machinery is costly because it requires deep bank-aligned queues, associative row-address comparisons, linked-list age tracking, and starvation control. The opportunity established in Section III-D is that many evaluated workloads already provide enough BpA in their incoming

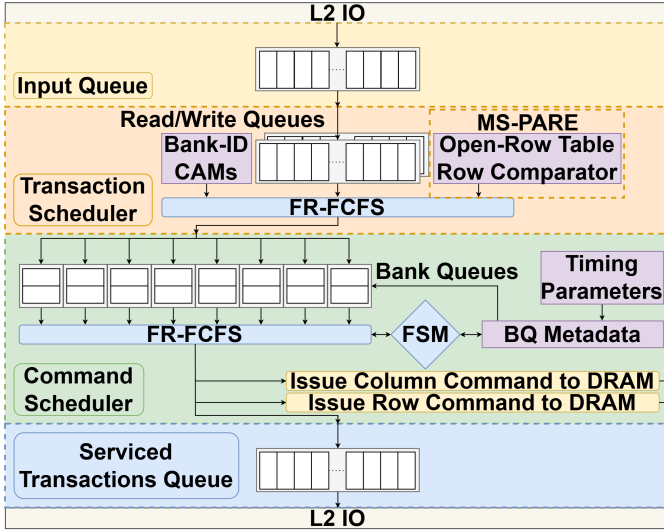


Fig. 8. PARE & MS-PARE controller organization. PARE replaces deep bank-aligned transaction queues with shallow unified queues while retaining per-bank command arbitration. MS-PARE incorporates a lightweight row comparator to avoid premature row-miss commitment.

request streams that little additional locality remains for this global search to recover. This motivates separating the two mechanisms: PARE removes transaction-level row-hit sorting while retaining the per-bank command arbitration needed to preserve bank-level parallelism.

V. PARE: ELIMINATING TRANSACTION-LEVEL SORTING

PARE exploits the native access granularity of the evaluated HPC and LLM workloads to remove global row-hit sorting from the transaction scheduler. It retains per-bank command arbitration but makes transaction admission independent of the open row. Figure 8 shows the resulting organization.

A. Simplifying the Transaction Scheduler

PARE replaces SMC’s deep, row-hit-oriented transaction scheduler with unified read and write queues. As established in Section II-B, a conventional controller derives bandwidth from row-buffer hit mining in the transaction scheduler and bank-level parallelism in the command scheduler. The first mechanism requires associative row-address comparisons, age tracking, and deep queues; the second requires only per-bank command arbitration. PARE removes the deep sorting machinery while retaining command arbitration across banks.

Each unified transaction queue is reduced from 128 to 32 entries. Its CAM stores only the bank ID of each transaction, allowing the scheduler to determine whether an entry can advance to the corresponding bank command queue without comparing every queued row address. To avoid head-of-line blocking when a bank is refreshing or its command queue is full, PARE scans a validity bitmap from a round-robin pointer until it finds an issuable entry. The pointer advances only on a grant, limiting unnecessary switching. The unified structure also eliminates the free lists needed to track holes in the SMC’s bank-aligned queues. Any issuable transaction can enter a bank

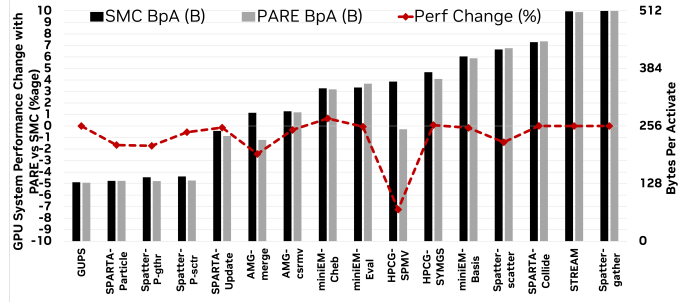


Fig. 9. GPU performance of HPC workloads with PARE. Workloads are ordered by SMC BpA. Bars show BpA; the line shows PARE performance relative to SMC. The evaluation methodology is described in Section VII.

command queue whenever that queue has space, giving PARE a FR-FCFS policy with no row-aware transaction selection.

a) **Retaining Bank-Level Parallelism:** PARE retains SMC’s eight per-bank command queues and command-scheduler FSM (Figure 7). Each 128 B transaction supplies at least four 32 B column commands, and the command scheduler interleaves command sequences across banks to mask *IRC*. PARE therefore removes the expensive mechanism for finding additional row hits while retaining the lower-cost mechanism that makes independent banks progress.

This division also defines PARE’s remaining vulnerability. The command scheduler can arbitrate only among transactions that have already entered the bank queues; it cannot recover a row hit left upstream by the row-oblivious transaction scheduler. We next quantify when this occurs.

B. Where PARE Loses Row Locality

Using the architecture-level methodology described in Section VII, Figure 9 compares PARE with SMC. PARE remains close to SMC for most workloads: its geometric-mean performance change is -0.95% , with 11 of 16 kernels within 1% and 14 of 16 within 2% . At one end of the BpA spectrum, SMC recovers little more than one cache line per activation, leaving little locality for any scheduler to extract. At the other end, naturally streaming kernels consume most or all of a row before moving on, so PARE observes nearly the same locality without transaction sorting.

The two meaningful exceptions are AMG merge-sort and HPCG SPMV. AMG merge-sort loses 2.43% as BpA falls from 285 B to 225 B; HPCG SPMV loses 7.25% as BpA falls from 355 B to 249 B. These kernels occupy the middle of the BpA spectrum, where SMC can still increase row locality and the recovered bandwidth affects execution time.

The loss is not caused by spatial bank imbalance, temporal concentration in too few banks, or write interference. SMC and PARE distribute requests similarly across banks, expose comparable bank-level parallelism, and retain the HPCG slow-down when writes are bypassed. Instead, PARE can place a same-bank row miss in the second command-queue entry while a transaction matching the open row remains in the transaction queue. These cases constitute 4.8% of scheduler decisions in HPCG but account for essentially all of the

serviced-request locality lost relative to SMC. This identifies a narrower requirement than global transaction sorting: the controller should avoid committing a row miss when the current activation can still serve a pending row hit.

VI. MINIMALLY-SORTING PARE (MS-PARE): RETAINING MINIMAL ROW AWARENESS

PARE shows that deep transaction sorting is unnecessary for most evaluated workloads, but the AMG and HPCG results expose one remaining scheduling hazard. MS-PARE retains PARE’s shallow unified queues and per-bank command arbitration while adding only the local decision needed to avoid premature row-miss commitment.

A. Avoiding Premature Row-Miss Commitment

Each bank command queue is two entries deep. If its head is accessing the open row, placing an older row miss in the second entry provides no immediate scheduling benefit: the miss cannot issue until the current access satisfies t_{RTP} or t_{WL} and the bank can be precharged. At the evaluated 625 MHz controller clock, this leaves at least three cycles to select and issue the bank’s next transaction. Occupying the second entry with the miss can instead prevent a later transaction matching the open row from reaching the command scheduler, prematurely ending the activation.

MS-PARE therefore admits a row-miss transaction to a bank command queue only when that queue is empty. A row hit may occupy the second entry, allowing the next column command to issue as soon as DRAM timing permits. This policy preserves useful locality without reconstructing SMC’s global row-hit search: MS-PARE traverses the depth of the transaction queue to identify possible row-hits, utilizing a round-robin selection pointer to provide fairness to the transaction selection. Section VIII evaluates how this local guard recovers the PARE performance losses while retaining most of its area and energy savings.

VII. EXPERIMENTAL METHODOLOGY

We evaluate PARE and MS-PARE at two levels. First, we compare both designs with a conventional sorting memory controller (SMC) in a future-GPU performance model to measure throughput and BpA. Second, we implement all three as synthesizable per-channel designs and carry them through HLS, RTL verification, and physical analysis to quantify area and energy.

A. System Configuration

Our baseline is a forward-looking throughput processor derived by scaling from the memory system up. Recent HBM4 prototypes already reach 3.3 TB/s per stack [4]. We project one more generation of stack-level scaling and assume 5 TB/s per stack as a plausible future design point, which we refer to as F-HBM. We then consider a two-reticle GPU package with 12 such stacks in total, i.e., six stacks per reticle, for an aggregate DRAM bandwidth of 60 TB/s. This 12-stack organization is an extrapolated package-level assumption, not a product claim.

TABLE II
FUTURE-GPU BASELINE AND F-3DMEM CHANNEL PARAMETERS USED
FOR ARCHITECTURAL EVALUATION.

Parameter	Value
Num. SMs / SM freq. / FP64 vec. FLOPS	576 / 1.22 GHz / ~180 TFLOP/s
L2 Bandwidth	~120 TB/s
DRAM bw.	~60 TB/s
DRAM organization	12 stacks (6/reticle), 256 ch./stack, 5 TB/s/stack
Ctrl. clk / chan. bw.	625 MHz ($t_{CK} = 1.6$ ns) / 32 B atom, 1 atom/cycle
Banks per ch.	8 (no bank groups/pseudo-channels)
Row size	512 B
DQ/ch. / BL / t_{BURST}	32 / 8 / 1.6 ns
<i>Key DRAM timings (ns unless noted)</i>	
t_{RC} 45	t_{RAS} 29 t_{RP} 16
t_{RRD} 2	t_{FAW} 16 $t_{RCD,RD}$ 16
$t_{RCD,WR}$ 6	t_{CCD} 1.6 RL 16
WL 10	t_{WTR} 3 t_{RTP} 3.5
t_{WR} 20	t_{RTRS} 1 t_{RREFD} 8
t_{REFI} 3.9 μ s	$t_{REFI,pb}$ 487.5 $t_{RFC,pb}$ 280

We size compute to match that bandwidth target. NVIDIA reports up to 22 TB/s of HBM bandwidth for a Rubin GPU package [22]; scaling package bandwidth from 22 TB/s to 60 TB/s implies roughly $2.7\times$ more memory throughput. We therefore scale Rubin-class compute resources approximately in proportion to the bandwidth increase, rounding the configuration to 576 SMs (as shown in Table II).

Finally, we use the F-3DMem organization summarized in Table II for the architecture-level evaluation. It maps the 60 TB/s bandwidth target onto a 3D-stacked design similar in spirit to BBCube [8]. Each controller in the evaluated F-3DMem organization operates at 625 MHz ($t_{CK} = 1.6$ ns) and transfers 32 B per cycle, providing 20 GB/s per channel. These assumptions intentionally favor channel proliferation so that we can study the cost of serving many fine-grained channels in the bandwidth regime targeted by F-3DMem-class throughput processors.

B. DRAM Energy Model

Figure 2 lists the DRAM energy terms we use. For HBM2, we use the component-wise values from O’Connor et al. [10]. For HBM4, no public energy model is available, so we derive E_{act} and E_{dm} from the projected HBM4 baseline in Adhinarayanan et al. [9]. For projected 3D DRAM, we keep the same per-bit activation energy as HBM4 and scale it to a 512 B row. We take E_{dm} from the FB-G-HBM point in [9] and set $E_{I/O} = 0$ under direct die-to-die bonding [8], [18].

C. Verification & Design Flow

The infrastructure used to design, verify, and evaluate the physical feasibility of PARE and MS-PARE spanned several complementary environments (Table III, Figure 10). System-level GPU simulation through NVArchSim [23] served a dual purpose: it provided the performance evaluations reported in Section VIII, and it generated the time-ordered, per-channel

TABLE III
CONTROLLER DESIGN AND EVALUATION INFRASTRUCTURE.

Design phase	Environment
Perf. sim. / trace gen.	NVArchSim
LLM request generation	LLMSim
Cycle-accurate model	SystemC + MatchLib
Functional verification	g++
High-level synthesis	Catapult HLS
RTL verification / FSDB	Synopsys VCS
Physical synthesis (3 nm)	Synopsys Fusion Compiler
Power estimation	Synopsys PrimePower

memory traces that drove all downstream verification and power analysis. These traces were converted to the bit-field encoding of the controller’s L2 interface and replayed through both the high-level model and the synthesized RTL, ensuring that the same request streams validated functional correctness at every abstraction level. All three controllers were carried through an identical flow, including cycle-accurate SystemC modeling, high-level synthesis, RTL verification, physical synthesis on a 3 nm process, and gate-level power estimation, so that all area and energy comparisons reflect equivalent design effort and optimization.

a) Cycle-Accurate Modeling & Functional Verification:

All three controllers were modeled in synthesizable SystemC using the MatchLib Connections library [24] for cycle-accurate ready/valid handshaking across module boundaries [25]. Each controller was partitioned into four hierarchical modules: input queue, transaction scheduler, command scheduler, and serviced transactions, which allowed independent unit-level verification before integration. The NVArchSim memory traces were read by a Verilog testbench acting as the L2 interface and driven into the SystemC model, which was compiled with g++ to produce a cycle-accurate executable. The SystemC models provide a cycle-accurate golden reference for each controller, enabling the architectural and RTL implementations to be checked against identical request streams. Total execution cycles per trace were compared against the architectural model to confirm functional equivalence, and the same cycle counts were later cross-checked against the RTL simulation to close the verification loop. This approach facilitates an agile VLSI design methodology that can be flexibly iterated, streamlining early detection and mitigation of design bottlenecks prior to committing to any physical implementation.

b) High-Level Synthesis & RTL Verification: Catapult HLS [26] converted the verified SystemC models into RTL under a 1.6 ns clock constraint (625 MHz). Each module was synthesized independently, enabling per-block architectural exploration of scheduling, pipelining, and resource allocation trade-offs. Queue storage for the input and serviced transactions modules was mapped to dual-ported read/write SRAM macros during HLS to match the target memory architecture, while the transaction scheduler queues were mapped to latch arrays. RTL verification was performed using Synopsys VCS [27], replaying the identical NVArchSim traces used during model verification. Execution cycle counts were compared

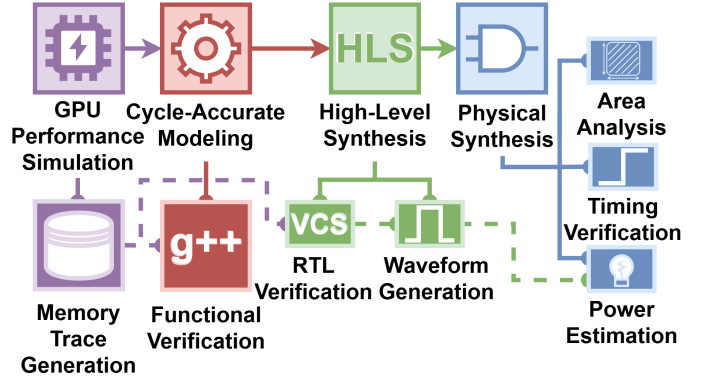


Fig. 10. SMC, PARE, and MS-PARE performance environment and physical design flow.

against the SystemC reference to confirm functional equivalence, and the resulting gate-level simulations were captured as FSDB waveforms to provide signal-level switching activity for power estimation.

c) Physical Synthesis: The verified RTL was synthesized into gate-level netlists using Synopsys Fusion Compiler on an advanced 3 nm process. Physically-aware synthesis optimized PARE and MS-PARE by accounting for parasitics and gate latency, along with performing a virtual placement to visualize the architecture (Figure 14). All SRAM macros were pre-generated to allow configurable queue depths during design space exploration. A key optimization was mapping the SMC’s bank-aligned transaction scheduler queues to latch arrays rather than SRAM macros, which provided Fusion Compiler with greater flexibility to apply aggressive clock gating and reduce dynamic power. PARE and MS-PARE benefited from the same optimization strategy. Fusion Compiler performed concurrent logic and physical optimization with virtual placement, and all designs met the 625 MHz timing constraint.

d) Power Estimation: The FSDB waveforms from gate-level simulation were replayed onto the synthesized netlists using Synopsys PrimePower. All three controllers were synthesized and analyzed at the typical-typical (TT)/0.67 V/105°C corner with standard-Vt cells under an identical flow, ensuring that all comparisons reflect equivalent optimization effort. PrimePower computed switching power from per-net toggle counts and load capacitances, internal power from liberty-table lookups on each cell-pin transition, and glitch power from timing-accurate spurious transitions on the gate-level netlist. Leakage power was determined from the state-dependent static power characterization of each cell. Latch-array memories were synthesized as standard cells and therefore received full gate-level power analysis, while SRAM macros were treated as black boxes with power derived from their RAM liberty models. We report energy efficiency as:

$$\text{Energy Efficiency (pJ/bit)} = \frac{P_{\text{total}} \cdot t}{N_{\text{bits}}} \quad (3)$$

where P is the total power consumption, t is the simulation runtime observed during RTL verification, and N is the total

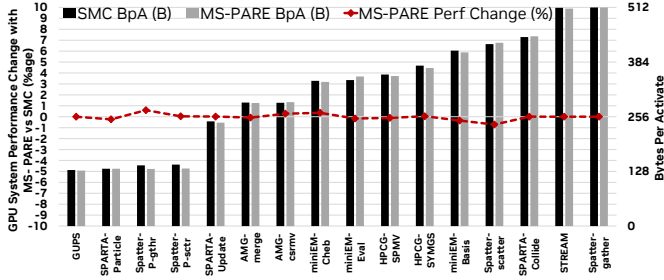


Fig. 11. **GPU performance of HPC workloads with MS-PARE.** Workloads are ordered by SMC BpA. Bars show BpA; the line shows MS-PARE performance relative to SMC.

bits sent to and from DRAM. This approach measures controller energy per delivered bit and accounts for both power and workload runtime.

D. Architecture-Level Performance and Workloads

Our primary architectural evaluation platform is NVArchSim [23], a full GPU simulator validated against Ampere-class GPU silicon and independently evaluated by Baddouh *et al.* [28]. In NVArchSim, we instantiate the future GPU from Table II and compare otherwise identical memory systems using SMC, PARE, or MS-PARE. For the HPC kernels, NVArchSim directly models GPU execution, the cache hierarchy, and DRAM timing. From these runs, we collect normalized performance and controller-visible BpA statistics.

For LLM inference, we couple LLMsim, a publicly available LLM simulator [29], [30], to the NVArchSim memory model. LLMsim generates the time-ordered memory traffic for a given batch-size $\times$ network configuration, and that traffic drives the same DRAM timing model used by NVArchSim. This lets us evaluate the controllers under a consistent future-memory model while using an LLM front-end that is specialized for transformer inference.

NVArchSim is also the source of the controller-validation traces used in our implementation flow. From an architectural run, we record the time-ordered requests arriving at a single memory channel, including the controller-visible address fields and request type. Because each controller instance serves one channel, replaying a single-channel trace is sufficient to validate the cycle-accurate controller model and to generate realistic switching activity for power analysis. The same trace format is used for SMC, PARE, and MS-PARE, so all downstream comparisons operate on identical request streams.

We evaluate two workload families. For traditional HPC, we use the performance-critical kernels from the ATS-5 [31] suite and HPCG [32], along with STREAM [33] and GUPS [34]. These applications were run on H100 GPUs through NVTracer [23] to collect input traces for NVArchSim. The kernels span regular streaming behavior, sparse accesses, and irregular update patterns. These kernels form the basis of the performance results in Section VIII-A and the controller area and energy results in Section VIII-B. Their DRAM

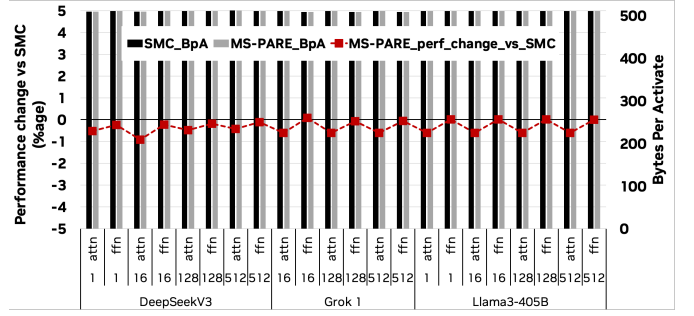


Fig. 12. **GPU performance of LLM workloads with MS-PARE.** Workloads are organized by model, batch size, and layer type. Bars show BpA; the line shows MS-PARE performance relative to SMC.

write fractions span 44.1% for SPARTA particle, 37.8% for Spatter scatter, 36.8% for AMG merge-sort, and 32.8% for STREAM add, down to less than 2% for the most read-dominated kernels. Finally, to study LLM inference, we evaluate representative batch-size $\times$ network combinations from the decode phase of 3 models (DeepSeek-V3 [35], Grok 1 [36], Llama-3-405B [37]), following the LLM-side methodology described by Duplex [30] and RoMe [12]. LLMsim captures the inference execution and generates the memory request stream seen by the GPU memory system.

VIII. EVALUATION

A. Architecture-Level Performance

Figure 11 shows that MS-PARE remains close to SMC across the HPC suite, with a geometric-mean performance change of -0.03% and a worst-case loss of 0.71% . This substantially narrows the losses observed for PARE in Figure 9: the geometric-mean loss falls from 0.95% to 0.03% , and the worst-case loss falls from 7.25% to 0.71% . In particular, the 7.25% loss for HPCG SPMV under PARE falls to 0.11% under MS-PARE. This workload exposed the premature row-miss commitment that motivated MS-PARE’s local row-hit guard. The guard recovers the lost performance without restoring SMC’s global transaction sorting, and the remaining deviations around parity do not consistently favor either controller.

Figure 12 extends the MS-PARE comparison to LLMsim-derived LLM inference workloads. These runs use the same future-memory model in NVArchSim, driven by batch-size $\times$ network traffic from LLMsim. LLM decode traffic generally consumes the full 512 B row on each activation, leaving little residual locality for SMC’s transaction scheduler to exploit. MS-PARE therefore tracks SMC across all evaluated configurations while avoiding global transaction sorting.

a) **Comparison with RoMe:** RoMe [12] removes controller complexity by changing the DRAM interface to transfer a full row. This is effective when the workload consumes the entire row but can overfetch substantially for fine-grained HPC accesses. Figure 13 applies a RoMe-style row-granularity interface to our 512 B-row future memory, a more conservative configuration than RoMe’s published 4 KB rows. This configuration loses as much as 64% when most transferred bytes

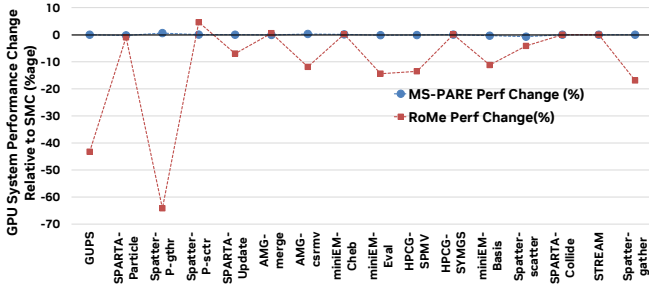


Fig. 13. GPU performance relative to SMC. A RoMe-style row-granularity interface, conservatively evaluated with 512B rows rather than RoMe’s published 4 KB rows, can overfetch on fine-grained HPC kernels; MS-PARE retains the fine-grained interface and tracks SMC.

are unused, whereas MS-PARE retains the conventional fine-grained interface and remains close to SMC. Thus, MS-PARE reduces controller area and energy without requiring a new external interface or software-visible memory abstraction.

B. Controller Area and Energy

a) Controller Area: Removing row-hit detection and starvation-prevention logic reduces PARE’s total controller area by 76.6% relative to SMC, from $15,450 \mu\text{m}^2$ to $3,608 \mu\text{m}^2$ (Table IV). The transaction scheduler provides the dominant savings: its area falls by 93%, shifting from 81.5% of the SMC footprint to 24.1% of PARE. This $4.28\times$ per-channel reduction is important for F-HBM and F-3DMem, where increasing the number of independent channels also replicates the controller logic.

TABLE IV
POST-SYNTHESIS CONTROLLER AREA AND AVERAGE ENERGY.

Design	Area (μm^2)	Area red.	Energy (pJ/bit)
SMC	15,449.75	—	0.120
PARE	3,607.61	76.6%	0.056
MS-PARE	4,119.00	73.3%	0.054

Since concurrent cross-boundary optimizations limit strict hierarchical tracking, we also analyze the synthesized cell distribution. Eliminating the associative row-address CAMs, age trackers, and starvation counters used for row-hit mining reduces combinational logic area from $9,928 \mu\text{m}^2$ to $1,600 \mu\text{m}^2$, an 83.9% reduction. Within that logic, buffer and inverter area falls from $3,171 \mu\text{m}^2$ to $262 \mu\text{m}^2$, a 91.7% reduction. Replacing SMC’s sixteen 16-entry bank-aligned queues (eight read and eight write) with two unified 32-entry queues (one read and one write) reduces noncombinational cell area from $4,912 \mu\text{m}^2$ to $1,397 \mu\text{m}^2$, a 71.6% reduction.

The $609 \mu\text{m}^2$ SRAM macros used for the input and serviced transactions queues are identical in SMC and PARE and do not contribute to these savings. Their share of the controller instead rises from 3.9% in SMC to 16.9% in PARE, confirming that the reduction comes from the transaction scheduling logic rather than the common storage structures.

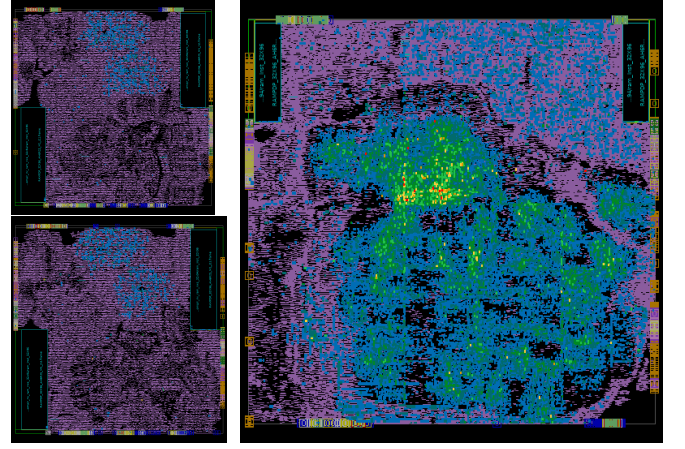


Fig. 14. Fusion Compiler virtual placement of PARE (top left, $3,608 \mu\text{m}^2$), MS-PARE (bottom left, $4,119 \mu\text{m}^2$), and SMC (right, $15,450 \mu\text{m}^2$). Identical SRAM macros occupy a larger fraction of the simplified designs.

MS-PARE retains PARE’s shallow unified queues and adds the lightweight row-address comparator and bounded-starvation control described in Section VI. These additions increase controller area by 14.2% over PARE, to $4,119 \mu\text{m}^2$, while retaining a **73.3%** reduction, or **$3.75\times$** smaller area, relative to SMC. The incremental cost therefore bounds the hardware required to recover the HPCG and AMG performance losses without restoring global transaction sorting.

b) Controller Energy: The same structures that dominate SMC area also consume energy through continuous CAM traversals, age tracking, and switching in the deep transaction queues. Across the 14 workload traces in Figure 15, SMC consumes 0.120 pJ/bit on average and reaches approximately 0.150 pJ/bit at its peak. PARE reduces the average to 0.056 pJ/bit, a 53.3% reduction. PARE’s worst case, 0.066 pJ/bit, remains 40% below SMC’s best case, 0.1105 pJ/bit. MS-PARE achieves an average energy efficiency of **0.054 pJ/bit**, a **55%** reduction relative to the conventional SMC, and marginally lower than PARE despite the hardware overhead of the local row-hit guard. Although the guard increases both dynamic and static power, the resulting runtime reduction compensates for these losses since energy per bit reflects total execution time. As a result, MS-PARE successfully addresses PARE’s performance vulnerabilities while preserving its low-energy profile, without adopting the high structural complexity of a conventional SMC.

The time-averaged power measurements show the source of this reduction. Average dynamic power falls from 12.27 mW for SMC to 6.32 mW for PARE, while average static power falls from 5.17 mW to 1.003 mW. Expressed per transferred bit, dynamic energy falls from 0.084 to 0.048 pJ/bit, a 43% reduction, and static energy falls from 0.036 to 0.008 pJ/bit, a 77.8% reduction. The larger static reduction follows from SMC’s substantially larger cell count. Place-and-route optimizations such as multi- V_t swapping and gate sizing can reduce absolute leakage in both designs, but the structural difference remains when those optimizations are applied equally.

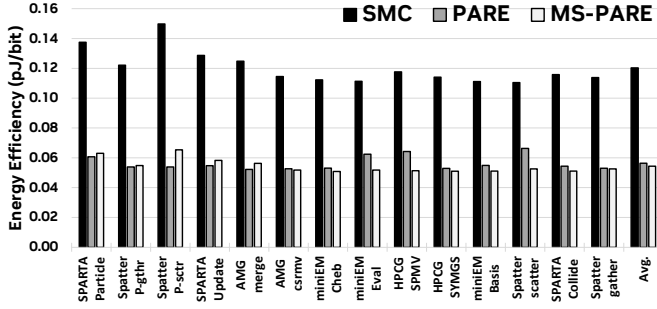


Fig. 15. Controller energy efficiency of SMC, PARE, and MS-PARE across 14 workloads. MS-PARE averages 0.054 pJ/bit versus 0.120 pJ/bit for SMC, a 55% reduction; MS-PARE’s worst case (0.065 pJ/bit) remains 41% below SMC’s best case (0.111 pJ/bit).

Across the 14 workloads, MS-PARE consumes slightly higher average power (6.45 mW dynamic, 1.186 mW static) than PARE. However, because energy per bit reflects execution time alongside power draw, MS-PARE achieves a lower average energy cost of 0.054 pJ/bit. By resolving PARE’s performance stalls on memory-bound workloads like HPCG SPMV and AMG merge-sort, MS-PARE’s runtime reductions compensate for its power overhead. Resolving PARE’s performance tail thus incurs minimal energy overhead, while avoiding the heavy energy overhead of SMC’s global search mechanism.

IX. SENSITIVITY STUDY

a) Queue Depth Analysis: Because a significant portion of the area saved by the PARE controller relative to the SMC is due to the reduced queue depth, we consider a version of the SMC with a 32-entry transaction-scheduler (TXS) queue like PARE. With 32 total entries in the queue, this corresponds to each per-bank TXS queue holding only 4 entries. Over a short window of time, if five requests to the same bank arrive, the fifth request will not be able to issue out of the input queue into a TXS queue. This head-of-line (HOL) blocking can occur with some frequency, leaving only a few bank-aligned TXS queues having any work, resulting in significant performance degradation in workloads with high DRAM utilization. Resolving this by adding out-of-order service from the input queue duplicates the TXS selection and starvation logic, increasing area and power consumption, going against our simplification goals.

Alternatively, switching to a unified TXS queue avoids HOL blocking but reduces the reorder window used to find row-buffer locality. Furthermore, it introduces timing pressure with the larger structure. Since the 1.6 ns clock constraint restricts our row detection each cycle, a unified queue with twice the depth of a per-bank queue would double our worst-case walk to find the oldest row-hit. The baseline SMC is able to mask the walk latency with parallel bank queues.

Overall, simply reducing the area of the SMC TXS queues is not an attractive path to effectively achieve goals similar to PARE.

b) Row Interleaving and Cache-Line Size: We first sweep two parameters for MS-PARE while holding the evaluated 512 B-row DRAM organization fixed: the L2 line size and channel-interleaving granularity, each varied from 128 B to 512 B. We report each point as (L2 line size, channel-interleaving granularity). Interleaving below the DRAM row size can artificially reduce natural BpA, while larger L2 lines and interleaving granularities expose more natural row locality. MS-PARE stays close to SMC for all workloads, including the most sensitive kernels. HPCG SPMV’s worst case is a 1.17% drop at (256 B, 256 B); its smallest drop is 0.13% at (128 B, 256 B), and it slightly outperforms baseline at (512 B, 512 B).

We separately evaluate a future-DRAM operating point with four banks per channel, 1 KB rows, and 1 KB channel interleaving. This point changes the DRAM organization and is not part of the two-parameter sweep above. MS-PARE and SMC are iso-performance at this operating point.

X. RELATED WORK

a) GPU Memory Scheduling and Controller Design:

Classical memory-controller work has long optimized for row-buffer locality and bank-level parallelism. Early policies such as FR-FCFS favored ready row hits over other requests [15], and later controllers such as SMS decomposed scheduling across stages to improve scalability and interference management in heterogeneous systems including CPU-GPU memory traffic [38].

Several works adapted these ideas to GPU-specific behavior. Complexity Effective Memory Access Scheduling for Many-Core Accelerator Architectures [39] observes that row locality can be disrupted before requests even reach the channel controller, and proposes preserving row-sorted order prior to the crossbar. That approach reduces the logic inside each channel controller, but it does not remove the sorting and arbitration cost from the GPU as a whole; it relocates that cost upstream. Other GPU schedulers make the controller more workload-aware. Lakshminarayana et al. [40] propose a potential-function-based scheduler for GPGPUs, while Chatterjee et al. [16] reduce latency divergence in irregular GPU workloads by coordinating service across requests from the same warp. More recently, Warped-MC [41] continues in this direction by using warp-aware prioritization to reduce long-tail memory latency in massively parallel processors.

These efforts all pursue better scheduling quality. PARE and MS-PARE explore a different design point. For future GPUs with many fine-grained channels, we ask when the right answer is not a better scheduler, but a much smaller one.

A large body of CPU memory-controller work targeted a different optimization regime: reducing latency and interference in shared DRAM systems for a small number of latency-sensitive threads. Representative examples include STFM [42], PAR-BS [43], ATLAS [44], and TCM [45], which balance throughput, fairness, and memory interference across CPU applications. Later work such as BLISS [46] and MISE [47]

continued this direction, focusing on simpler interference-aware scheduling and slowdown predictability. These designs are important background, but they address a different operating point from throughput processors, where abundant thread-level parallelism makes latency tolerance much higher and shifts the controller objective toward scalable bandwidth delivery.

XI. CONCLUSION

Memory bandwidth remains a primary constraint for LLM inference and HPC workloads. Successive HBM generations have increased per-stack bandwidth, and projected 2.5D and 3D organizations can extend that trend by exposing many more fine-grained channels. Replicating a conventional sorting controller for every channel, however, scales controller area and energy with bandwidth even as the energy of the DRAM access itself falls.

PARE exploits the natural bytes per activate of modern workloads to remove deep transaction-level row sorting while retaining bank-level command arbitration. Its isolated losses identify premature row-miss commitment as the remaining source of useful locality loss. MS-PARE restores only a local row-hit guard, avoiding this hazard without reconstructing SMC's aggressive associative search. MS-PARE remains within 0.71% of SMC performance across the evaluated HPC kernels, while reducing per-channel area by 73.3% and average controller energy across 14 workload traces by 55%. These results show that future high-bandwidth systems can treat controller simplicity as a first-class scaling constraint alongside DRAM organization, thermals, and array efficiency.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 2146752. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

- [1] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 5, 2023, pp. 606–624.
- [2] S. Yun, S. Park, H. Nam, Y. Lee, G. Lee, K. Kyung, S. Kim, N. S. Kim, J. Kim, H. Kim, J. Cho, S. Baek, and J. H. Ahn, "Rethinking LLM inference bottlenecks: Insights from latent attention and mixture-of-experts," 2026. [Online]. Available: <https://arxiv.org/abs/2507.15465>
- [3] NVIDIA, "NVIDIA NVLink Fusion brings NVHBM to next-generation AI infrastructure," <https://developer.nvidia.com/blog/nvidia-nvlink-fusion-brings-nvhbm-to-next-generation-ai-infrastructure/>, 2026.
- [4] S. Joo, J. Kim, Y. Lee, J.-Y. Kim, Y. Lee, Y.-M. Kim, C. Oh, K.-H. Shim, H. Lee, Y.-Y. Byun, C. Bae, J. Kim, J.-M. Ryu, S.-h. Kang, J. Lee, Y.-U. Chang, J. Lee, J. Hwang, D. Seo, K.-H. Na, Y. G. Song, D. Lim, K.-S. Ha, Y.-S. Sohn, and S.-J. Hwang, "15.6 A 36GB 3.3TB/s HBM4 DRAM with per-channel TSV RDQS auto calibration and fully-programmable MBIST," in *2026 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 69, 2026, pp. 264–266.
- [5] TrendForce, "TSMC unveils custom C-HBM4E details: N3P logic dies reportedly target 2× efficiency gain," <https://www.trendforce.com/news/2025/12/01/news-tsmc-unveils-custom-c-hbm4e-details-n3p-logic-dies-reportedly-target-2x-efficiency-gain/>, Dec. 2025.
- [6] S. Han, "HBM base die: How HBM will evolve using advanced logic processes," Samsung Electronics, Hot Chips 2026 Memory Technology Tutorial, Aug. 2026. [Online]. Available: <https://hc2026.hotchips.org/assets/program/tutorials/HC2026.HBMBaseDie.SangwookHan.v20.pdf>.
- [7] P. J. Nair, R. Hadidi, S. Ganesh, S. Kodge, S. Rathore, N. Thanawala, N. Reddy, G. Saharia, V. Patankar, A. Tiruvur, N. Kurella, and S. Bhoja, "Early silicon of Raptor: The first 3D-DRAM accelerator for generative inference," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. Raleigh, NC, USA: IEEE, 2026, pp. 2632–2647.
- [8] N. Chujo, K. Sakui, S. Sugatani, H. Ryoson, T. Nakamura, and T. Ohba, "Bumpless build cube (BBCube) 3D: Heterogeneous 3D integration using WoW and CoW to provide TB/s bandwidth with lowest bit access energy," in *2023 IEEE Symposium on VLSI Technology and Circuits*. IEEE, 2023, pp. 1–2.
- [9] V. Adhinarayanan, B. M. Beckmann, W. Li, M. Seyedzadeh, S. Blagodurov, D. Aguren, and H. H. Lee, "Folded banks: 3D-stacked HBM design for fine-grained random-access bandwidth," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 1819–1833.
- [10] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 41–54.
- [11] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Future scaling of processor-memory interfaces," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 42:1–42:12. [Online]. Available: <https://doi.org/10.1145/1654059.1654102>
- [12] H. Nam, S. Baek, J. Kim, M. J. Kim, and J. H. Ahn, "RoMe: Row granularity access memory system for large language models," in *2026 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Sydney, Australia: IEEE, 2026, pp. 1–15.
- [13] N. Chatterjee, M. O'Connor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally, "Architecting an energy-efficient DRAM system for GPUs," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 73–84.
- [14] H. Ha, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Improving energy efficiency of DRAM by exploiting half page row access," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.
- [15] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 128–138.
- [16] N. Chatterjee, M. O'Connor, G. Loh, N. Jayasena, and R. Balasubramanian, "Managing DRAM latency divergence in irregular GPGPU applications," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 128–139.
- [17] N. Chatterjee, N. Muralimanohar, R. Balasubramanian, A. Davis, and N. P. Jouppi, "Staged reads: Mitigating the impact of DRAM writes on DRAM reads," in *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA)*. New Orleans, LA, USA: IEEE Computer Society, 2012, pp. 41–52.
- [18] S. Bhoja, "Corsair – an in-memory computing chiplet architecture for inference-time compute acceleration," in *Hot Chips 37 (HC37): A Symposium on High Performance Chips*. Stanford, CA: IEEE, Aug. 2025, d-Matrix. Available: <https://www.d-matrix.ai/hot-chips-2025/>.
- [19] J. Shao and B. T. Davis, "A burst scheduling access reordering mechanism," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 285–294.
- [20] M.-Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SECDED Codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.
- [21] S. Gurumurthi, K. Lee, M. Jang, V. Sridharan, A. Nygren, Y. Ryu, K. Sohn, T. Kim, and H. Chung, "HBM3 RAS: Enhancing resilience at scale," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 158–161, 2021.
- [22] NVIDIA, "Inside the NVIDIA Vera Rubin platform: Six new chips, one AI supercomputer," <https://developer.nvidia.com/blog/inside-the-nvidia>

- a-rubin-platform-six-new-chips-one-ai-supercomputer/, 2026, NVIDIA Technical Blog, accessed April 2, 2026.
- [23] O. Villa, D. Lustig, Z. Yan, E. Bolotin, Y. Fu, N. Chatterjee, N. Jiang, and D. Nellans, "Need for speed: Experiences building a trustworthy system-level GPU simulator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 868–880.
- [24] B. Khailany, E. Khmer, R. Venkatesan, J. Clemons, J. S. Emer, M. Fojtik, A. Klinefelter, M. Pellauer, N. Pinckney, Y. S. Shao, S. Srinath, C. Torng, S. L. Xi, Y. Zhang, and B. Zimmer, "A modular digital VLSI flow for high-productivity SoC design," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 72:1–72:6. [Online]. Available: <https://doi.org/10.1145/3195970.3199846>
- [25] A. Mahmoudi, A. Nešković, C. Thermann, R. Sehm, C. Hübner, T. Plattenteich, R. Meyer, R. Buchty, M. Berekovic, and S. Mulhem, "A systematic mapping study on SystemC/TLM modeling capabilities in new research domains," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, no. 4, pp. 53:1–53:41, Jun. 2025. [Online]. Available: <https://doi.org/10.1145/3735641>
- [26] T. Bollaert, *Catapult Synthesis: A Practical Introduction to Interactive C Synthesis*. Dordrecht: Springer Netherlands, 2008, pp. 29–52. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_3
- [27] J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, *Writing Testbenches using SystemVerilog*. Springer Science & Business Media, 2006.
- [28] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal kernel analysis: A tractable methodology to simulate scaled GPU workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 724–737. [Online]. Available: <https://doi.org/10.1145/3466752.3480100>
- [29] SCALE Lab, Seoul National University, "LLMSimulator: Cycle-accurate simulator for large language models," GitHub repository, 2025, accessed September 9, 2026. [Online]. Available: <https://github.com/scale-snu/LLMSimulator>
- [30] S. Yun, K. Kyung, J. Cho, J. Choi, J. Kim, B. Kim, S. Lee, K. Sohn, and J. H. Ahn, "Duplex: A device for large language models with mixture of experts, grouped query attention, and continuous batching," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 1429–1443.
- [31] Los Alamos National Laboratory, "ATS-5 benchmarks," <https://lanl.github.io/benchmarks/>, 2022, benchmark suite for ATS-5 with representative LANL/NNSA workloads.
- [32] M. A. Heroux and J. J. Dongarra, "High-performance conjugate-gradient benchmark (HPCG)," *International Journal of High Performance Computing Applications*, vol. 30, no. 3, pp. 392–408, 2016.
- [33] J. D. McCalpin, "STREAM: Sustainable memory bandwidth in high performance computers," University of Virginia, Tech. Rep., 1995, original technical report and continually updated benchmark. [Online]. Available: <https://www.cs.virginia.edu/stream/>
- [34] HPC Challenge Benchmark Team, "HPC challenge benchmark: RandomAccess (GUPS)," [https://hpcchallenge.org/projectsfiles/hpcc/Rand omAccess.html](https://hpcchallenge.org/projectsfiles/hpcc/Rand%20Access.html), 2008, measures Giga Updates Per Second (GUPS) via random memory updates.
- [35] DeepSeek-AI, "DeepSeek-V3 technical report," 2025. [Online]. Available: <https://arxiv.org/abs/2412.19437>
- [36] xAI. (2024) Grok-1: Open release. [Online]. Available: <https://x.ai/news/grok-os>
- [37] Meta AI. (2024) Introducing Llama 3.1: Our most capable models to date. [Online]. Available: <https://ai.meta.com/blog/meta-llama-3-1/>
- [38] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 416–427.
- [39] G. L. Yuan, A. Bakhoda, and T. M. Aamodt, "Complexity effective memory access scheduling for many-core accelerator architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: Association for Computing Machinery, 2009, pp. 34–44. [Online]. Available: <https://doi.org/10.1145/1669112.1669119>
- [40] N. B. Lakshminarayana, J. Lee, H. Kim, and J. Shin, "DRAM scheduling policy for GPGPU architectures based on a potential function," *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 33–36, 2012.
- [41] J. H. Jeong, M. K. Yoon, Y. Oh, and G. Koo, "Warped-MC: An efficient memory controller scheme for massively parallel processors," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 546–555. [Online]. Available: <https://doi.org/10.1145/3605573.3605645>
- [42] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 146–160.
- [43] —, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Proceedings of the 35th International Symposium on Computer Architecture*, 2008, pp. 63–74.
- [44] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *Proceedings of the 16th IEEE International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–12.
- [45] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 65–76.
- [46] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "The blacklisting memory scheduler: Achieving high performance and fairness at low cost," in *Proceedings of the 32nd IEEE International Conference on Computer Design*, 2014, pp. 8–15.
- [47] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "MISE: Providing performance predictability and improving fairness in shared main memory systems," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '13. USA: IEEE Computer Society, 2013, pp. 639–650. [Online]. Available: <https://doi.org/10.1109/HPCA.2013.6522356>

A. Abstract

This artifact provides the synthesizable, cycle-accurate hardware implementations of the three per-channel GPU memory controllers evaluated in the paper: the conventional Sorting Memory Controller baseline (SMC), the Power and Area-Efficient controller (PARE), and its minimally sorting variant (MS-PARE). Each design is written in synthesizable SystemC using the MatchLib/Connections library and is partitioned into the four hierarchical modules described in Sections IV–VI (input queue, transaction scheduler, command scheduler, and serviced transactions queue) plus a Hsiao SEC-DED codec. We also include the per-channel, time-ordered L2 request traces (14 HPC/ML kernels) that drive functional verification and switching-activity-based power estimation. Together these enable a user to (i) compile each controller with g++ and confirm cycle-accurate functional equivalence against the reference cycle counts, and (ii) reproduce the downstream area/energy flow (HLS → RTL verification → physical synthesis → gate-level power) that produces the paper’s area and energy results (Table IV and Fig. 14), given access to the required commercial EDA tools and an advanced process design kit (PDK). The upstream architectural simulator, the EDA tools, and the PDK are proprietary and are not included; they are documented below along with how to obtain them. While the provided SystemC models and traces enable compilation and functional verification of the schedulers, reproducing the end-to-end physical design flow requires proprietary EDA tools, PDKs, and commercial licenses.

B. Artifact check-list (meta-information)

- **Algorithm:** FR-FCFS command scheduling; PARE and MS-PARE transaction scheduling; Hsiao SEC-DED ECC.
- **Program:** SystemC memory-controller models (PARE, MS-PARE, SMC); MatchLib/Connections.
- **Compilation:** g++ (C++11), SystemC, MatchLib/nvhls; Catapult HLS.
- **Transformations:** HLS (SystemC → RTL), 1.6 ns / 625 MHz.
- **Data set:** Per-channel L2 traces; 14 HPC/ML kernels (ATS-suite, HPCG, STREAM).
- **Run-time environment:** Linux; SystemC + MatchLib; Synopsys/Siemens EDA.
- **Hardware:** x86-64; no special hardware.
- **Run-time state:** Deterministic; trace-driven.
- **Execution:** make per design; scripted HLS/synthesis/power.
- **Metrics:** Execution cycles; area (μm^2); energy (pJ/bit); power (mW).
- **Output:** Cycle counts, logs; (EDA) netlists, FSDB, area/power reports.
- **Experiments:** Functional verification; area/energy comparison.
- **How much disk space required (approximately)?:** ~100 MB (~18 MB compressed); several GB for full EDA flow.
- **How much time is needed to prepare workflow (approximately)?:** Minutes (SystemC); hours–days (EDA licenses/PDK).
- **How much time is needed to complete experiments (approximately)?:** Seconds–minutes/trace (SystemC); hours/design (full flow).

- **Publicly available?:** Open-source memory controllers will reside on <https://github.com/NVlabs>.
- **Workflow automation framework used?:** GNU Make; nvhls harness; Catapult TCL.
- **Archived (provide DOI)?:** Refer to “Publicly available?”.

C. Description

a) **How to access:** The open-source memory controllers will reside on <https://github.com/NVlabs>. The archive contains three self-contained design directories—PARE, MS_PARE, and SMC—which correspond to PARE, MS-PARE, and SMC, respectively. Each contains its SystemC source and the per-channel HPC/ML kernel trace set. Note that the high-level synthesis and physical-design flow depends on commercial EDA tools and an advanced-node PDK that cannot be redistributed. Those dependencies are documented below so that users with the appropriate licenses can reproduce the flow on their side.

b) **Hardware dependencies:** A standard x86-64 Linux workstation or server. No GPU, accelerator, or special hardware is required. Running the full physical-synthesis and power stages benefits from a multi-core machine with several GB of RAM and scratch disk.

c) Software dependencies:

- **Included, open-source or standard:** a C++11 compiler (g++); the Accellera SystemC reference library; and the NVIDIA MatchLib/Connections and nvhls libraries (open source), used for cycle-accurate ready/valid modeling.
- **Not included, proprietary (document only):** *Trace generation:* NVArchSim (system-level GPU simulator) and LLMSim (LLM request generator), used to produce the traces, which are already provided, so these are not needed to exercise the artifact. *Synthesis/analysis flow:* Catapult HLS (SystemC → RTL); Synopsys VCS (RTL verification and FSDB generation); Synopsys Fusion Compiler (physical synthesis, 3 nm); Synopsys PrimePower (gate-level power). *Technology:* an advanced (3 nm-class) PDK and the corresponding standard-cell and SRAM macro libraries. These tools require commercial licenses and the PDK requires a foundry agreement; they must be obtained independently by the user.

d) **Data sets:** Per-channel, time-ordered L2 memory-request traces for 14 HPC/ML kernels spanning streaming, sparse, and irregular access patterns (AMG, HPCG (SYMGS, SPMV), miniEM, SPARTA, and Spatter gather/scatter variants). Each trace is encoded in the controller’s L2 interface bit-field format and is replayed identically through all three controllers, so all downstream comparisons operate on identical request streams. Because each controller serves a single DRAM channel, a single-channel trace is sufficient to validate the model and to generate realistic switching activity.

e) **Models:** Not applicable. LLM inference traffic is captured as pre-generated memory traces.

D. Installation

Clone the PARE repository, which contains three top-level directories (PARE, MS_PARE, SMC), from NVlabs

(<https://github.com/NVlabs>). Each directory contains the cycle-accurate SystemC sources, testbench, and traces. To build a design, install the Accellera SystemC library, Boost, and the MatchLib/nvhls headers as shipped with Catapult (the Connections marshaller is not self-contained in the public hlslibs release), set the corresponding include/library environment variables, and build with the design Makefile. High-level synthesis and physical synthesis (Catapult HLS, VCS, Fusion Compiler, PrimePower, and the target PDK/macro libraries) are not part of this artifact.

E. Experiment workflow

- 1) **Cycle-accurate functional verification.** For each design, compile the SystemC model with g++ and run it against the included traces. Using a standard g++ flow, a user can build and simulate each controller with:

```
make BUILD=PARE
make BUILD=MS_PARE
make BUILD=SMC
```

The testbench acts as the L2 interface and replays each trace into the controller. The run reports total execution cycles and a completion status per trace. With runtime trace selection, a single compilation produces a simulation binary capable of evaluating any workload. Rather than re-compiling the testbench for each kernel, the target trace is selected dynamically at runtime via the TRACE environment variable (accepting 1-based indices from 1 to 14). This enables all HPC/ML traces to be executed in parallel without recompilation. The testbench acts as the L2 interface, replaying the selected trace into the controller and reporting total execution cycles and completion status for each run.

Note: While functional verification (Step 1) is executable with the provided open-source SystemC environment, the remaining physical design and ASIC evaluation steps (Steps 2–5) require commercial EDA tools (Siemens Catapult HLS, Synopsys VCS, Fusion Compiler, PrimePower; or similar tools) and a proprietary 3 nm PDK.

- 2) **High-level synthesis.** Run Catapult HLS on the verified SystemC model under the 1.6 ns (625 MHz) clock constraint to generate RTL, mapping the input queue and serviced transactions queue storage to dual-ported SRAM macros. The transaction scheduler queues should be mapped to latch-array macros. This will generate the Verilog RTL that will be used for physical synthesis.
- 3) **RTL verification and FSDB capture.** Re-run the identical traces on the generated RTL in Synopsys VCS, using the same testbench that was used for the SystemC model, and dump the FSDB waveforms for switching activity.
- 4) **Physical synthesis.** Synthesize the verified RTL to a gate-level netlist with Synopsys Fusion Compiler on the 3 nm process. Confirm the 625 MHz timing constraint is met and collect the area report.
- 5) **Power estimation.** Replay the FSDB activity on the synthesized netlist with Synopsys PrimePower at the

TT/0.67 V/ 105 °C corner to obtain dynamic, static, and total power, and compute energy efficiency (pJ/bit) per equation 3.

F. Evaluation and expected results

The artifact supports the paper’s implementation-level claims:

- **Functional equivalence.** The SystemC and RTL both execute identical traces, confirming that PARE, MS-PARE, and SMC are cycle-accurate and functionally equivalent at the L2 interface. This is reproducible with the open-source (SystemC/MatchLib) toolchain and standard build utilities.
- **Area (Table IV, Fig. 14).** Physical synthesis yields $\sim 3,608 \mu\text{m}^2$ for PARE, $\sim 4,119 \mu\text{m}^2$ for MS-PARE, and $\sim 15,450 \mu\text{m}^2$ for SMC, assuming similar PDK and standard-cell characteristics, as described in the paper.
- **Energy efficiency (Table IV, Fig. 15).** The paper reports averages of $\sim 0.056 \text{ pJ/bit}$ for PARE, $\sim 0.054 \text{ pJ/bit}$ for MS-PARE, and $\sim 0.120 \text{ pJ/bit}$ for SMC across 14 HPC/ML kernels.

Absolute area and power numbers depend on the specific PDK, standard-cell and macro libraries, and tool versions, so exact values will differ across technologies. The *relative* area and energy reductions between the controllers are the reproducible, technology-robust results, since all three designs are carried through an identical flow.

G. Experiment customization

Controller parameters are exposed in each design’s configuration header (e.g. queue depths, number of banks, DRAM timing parameters, row/atom sizes, and ECC width), allowing exploration of the sensitivity studies in Section IX, such as TXS queue depth, L2 line size, and channel-interleave granularity. Additional single-channel traces in the same L2 bit-field format can be dropped into a design’s trace directory and replayed without code changes.

H. Notes

Since the architectural performance results (Figs. 9, 11, 12, and 13) depend in part on the proprietary NVArchSim simulator alongside the public LLMSim framework, full end-to-end trace generation is limited to the public toolchain, whereas trace replay and scheduler evaluation are reproducible. The artifact instead reproduces the implementation-level results and the cross-level cycle-accurate functional equivalence. Because the HLS/synthesis/power stages require commercial EDA tools and a foundry PDK, users without those licenses can still verify the functional-equivalence claims using the included SystemC models and traces, which require only open-source dependencies.

I. Methodology

Submission, reviewing and badging methodology:

- <https://www.acm.org/publications/policies/artifact-review-and-badging-current>
- <https://cTuning.org/ae>