

CREDIT: Cost-guided Reduction-reuse with Efficient DSMEM Inter-CTA Tiling

Zhengxiong Li
University of Wisconsin–Madison
Madison, WI, USA
zhengxiong.li@wisc.edu

Tsung-Wei Huang
University of Wisconsin–Madison
Madison, WI, USA
tsung-wei.huang@wisc.edu

Umit Ogras
University of Wisconsin–Madison
Madison, WI, USA
uogras@wisc.edu

Abstract—NVIDIA distributed shared memory (DSMEM) enables direct shared-memory access within a thread block cluster. However, cluster synchronization, remote access, and resource costs make it difficult to determine when DSMEM improves performance. To fill this gap, we propose CREDIT, a cost-guided framework that identifies DSMEM-profitable workload patterns, predicts their profitability range, and delivers consistent speedups across diverse workloads. CREDIT combines three innovations: (1) a profiling-driven characterization that identifies workload patterns likely to benefit from DSMEM; (2) a transformation that applies DSMEM to reduction-reuse workloads; (3) a cost model based on profiling data, to determine its profitability range. Evaluations on diverse workloads show CREDIT achieves 91.7% prediction accuracy on profitability. CREDIT beats `torch.compile`, Triton, and optimized non-DSMEM CUDA baselines on all six workloads, with geometric-mean speedups of $1.466\times$ on RTX 5090 and $1.318\times$ on H100. CREDIT’s source code is publicly available at <https://github.com/zhengxiongli08/CREDIT>.

I. INTRODUCTION

Modern GPUs increasingly expose the memory wall as peak performance has grown faster than off-chip memory bandwidth [1]. Table I summarizes this trend under the Roofline model [2] for representative NVIDIA GPUs, including Volta [3], Ampere [4], Hopper [5], [6], and Blackwell [7], [8]. The dense FP16 compute-to-bandwidth ratio rises from 139 FLOPs/byte on V100 to about 295 FLOPs/byte on H100, and Blackwell GPUs remain in the same regime. As a result, many operators fail to achieve peak performance unless they can reduce memory footprint to increase arithmetic intensity.

Starting from Hopper, NVIDIA introduced thread block clusters and DSMEM [9], [10]. A cluster is a group of cooperative thread arrays (CTAs) that are co-scheduled on streaming multiprocessors (SMs) and can synchronize as a unit. DSMEM allows these blocks to directly load, store, and perform atomic operations on one another’s shared memory (SMEM). However, DSMEM operations are not free. Remote shared-memory accesses are slower than local shared memory accesses, cluster synchronization adds latency, and larger shared memory allocations or fixed cluster sizes can reduce occupancy and performance.

We propose CREDIT, which identifies which workload patterns benefit from DSMEM, predicts the profitability range for a given shape, and applies the transformation only when it pays

TABLE I: Compute-to-memory scaling in representative NVIDIA GPUs

GPU	Arch.	Peak Perf. (TFLOP/s)	Mem. BW (TB/s)	Roofline Ridge (FLOPs/byte)
V100 SXM	Volta	125	0.900	139
A100 SXM	Ampere	312	2.039	153
H100 SXM	Hopper	990	3.350	295
B200 SXM	Blackwell	2250	8.000	281
RTX 5090	Blackwell	419	1.792	234

off. Supported by detailed profiling data on DSMEM performance, we identify that DSMEM pays off for wide reduction-reuse kernels, which reduce a wide vector and then reuse its per-element data to compute a full-size output. DSMEM can reduce off-chip memory traffic by exchanging data across a cluster, avoiding rereading from global memory between two reduction passes. CREDIT applies a transformation to split the input data across multiple blocks, where each block computes its own partial results and exchanges with others within the cluster. CREDIT also involves a cost model to determine whether it’s beneficial to apply the transformation to a given workload and problem size.

We evaluate CREDIT across diverse workloads. The results show size- and architecture-dependent crossovers: no-DSMEM baselines are usually preferable for small problem sizes, while CREDIT becomes competitive when the avoided global-memory rereads amortize extra overheads introduced by DSMEM, including synchronization, DSMEM communication, and occupancy costs. Evaluations show CREDIT outperforms the fastest baseline on all six workloads, with a geometric-mean speedup of $1.466\times$ on RTX 5090 and $1.318\times$ on H100. Nsight Compute measurements demonstrate CREDIT reduces off-chip memory traffic by 33-60% on different workloads.

Our contributions in this paper are outlined as follows:

- **Workload and DSMEM performance analysis.** We quantitatively characterize the performance of DSMEM in terms of read/write latency, throughput, and synchronization overhead, which indicates promising workload patterns.
- **Profitability cost model.** We develop a profile-guided cost model that compares the time we can save against

extra overheads introduced by DSMEM, including cluster synchronization, DSMEM communication, and occupancy costs, to predict profitability.

- **Cross-platform evaluation.** We evaluate diverse workloads on RTX 5090 and H100 against `torch.compile`, Triton, and optimized no-DSMEM baselines, and validate the off-chip traffic reduction using Nsight Compute.

For the rest, Section II provides background and discusses prior works. Section III presents the proposed CREDIT framework. Finally, Section IV presents the evaluations on diverse workloads and Section V concludes the paper.

II. BACKGROUND AND RELATED WORK

A. DSMEM Programming Model

In the conventional CUDA execution model, shared memory is scoped to one thread block, and blocks in a grid have no guaranteed relative scheduling order. As shown in Fig. 1(a), communicating intermediate state across thread blocks generally requires device memory, atomics, or a kernel boundary. Starting from Hopper architecture, thread block clusters add an intermediate cooperation scope: all blocks in a cluster are scheduled concurrently on SMs within one graphics processing cluster (GPC) and can synchronize through cluster-wide barriers [9], [10]. Each block still owns a distinct shared-memory allocation, but DSMEM allows a block to map an address into a peer block’s allocation and issue remote loads, stores, and atomic operations. The resulting address space is logically distributed across the cluster rather than a physically unified cache, as illustrated in Fig. 1(b).

B. Prior Approaches to Reducing Data Movement

Off-chip traffic can first be reduced without cross-CTA communication. IO-aware algorithms restructure an operator around the memory hierarchy; FlashAttention [11] and FlashAttention-2 [12] use tiled online reductions and refined work partitioning to avoid materializing the attention matrix in HBM. At the graph and compiler levels, TVM [13], PyTorch [14], DNNFusion [15], Ansor [16], and Roller [17] optimize fusion or schedules using generated code and performance models. AStitch [18] exploits hierarchical reuse when fusing memory-intensive graphs, Welder [19] uses a tile-traffic cost model to coordinate intra- and inter-operator reuse, and MonoNN [20] forms monolithic GPU kernels to reduce intermediate materialization and launch overhead. Triton [21] provides a tiled programming model for custom kernels, while ThunderKittens [22] exposes tile-level abstractions for registers, shared memory, asynchronous copies, and matrix units. These approaches make data movement a first-class optimization objective, but registers and per-CTA shared memory remain their principal scopes for on-chip communication and reuse.

Prior microarchitectural studies provide an empirical basis for reasoning about this additional scope. Luo et al. [23] and Luhn et al. [24] benchmark Hopper’s memory hierarchy and characterize DSMEM latency and bandwidth across cluster

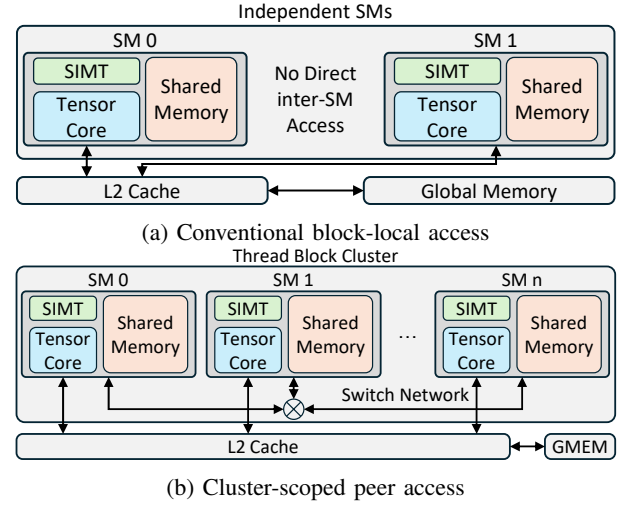


Fig. 1: Shared-memory access with and without a thread block cluster. DSMEM lets a CTA address allocations owned by peer CTAs through the inter-SM interconnect; each allocation remains physically local to its owner

configurations, while Jin et al. [25] expose topology-dependent latency and bandwidth behavior in real GPU networks-on-chip. These studies establish that peak specifications are insufficient and inter-SM communication must be calibrated on physical hardware. CREDIT complements them by isolating the read, store, visibility, and barrier primitives exercised by its protocol on two architectures and feeding those measurements into workload selection.

Recent systems directly turn the cluster scope into workload optimizations. FlashFuser [26] introduces DSMEM communication abstractions, data movement analysis, and an analytical cost model to search fusion plans for compute-intensive operator chains. ClusterFusion [27] profiles sensitivity to cluster configuration, models collective traffic, and uses optimized operators to fuse LLM decoding stages. Quack [28] proposes a series of optimizations for memory-bound kernels on Hopper and Blackwell platforms, including RMSNorm, softmax, and cross-entropy loss.

C. Positioning of CREDIT

Prior work therefore spans conventional kernel fusion and dataflow optimization, hardware characterization, and target-specific DSMEM designs. CREDIT aims to answer whether an operator at a given shape should use clustered staging at all, relative to a competitive non-clustered implementation. We formalize this workload class and use matched primitive and control measurements together with one no-DSMEM timing to predict the profitability of each workload-shape pair and crossover. The contribution is a workload-level characterization and selection framework, validated on Hopper H100 and Blackwell RTX 5090 across both profitable and unprofitable cases, rather than the first use of DSMEM or a new cluster-reduction collective.

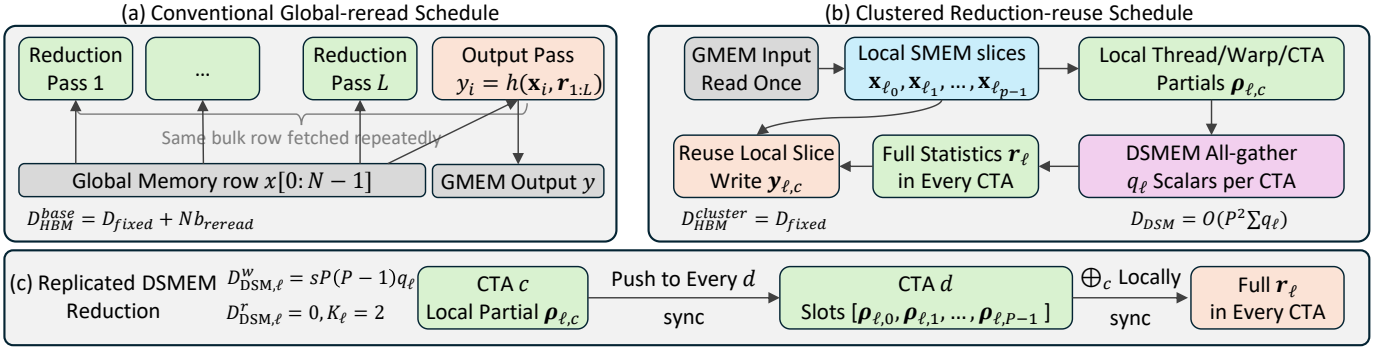


Fig. 2: CREDIT transformation for reduction-reuse kernels. (a) Conventional passes reread bulk rows. (b) A cluster retains owner-local slices and exchanges scalar partials. (c) Each CTA pushes partials to every peer, which combines its local slots after synchronization

III. CREDIT FRAMEWORK

This section presents CREDIT using three key parts: (1) detailed profiling on the hardware costs introduced by DSMEM; (2) a transformation from multi-pass reduction into distributed local staging and compact reduction; (3) a cost model to determine the profitability for a given workload and problem size.

A. DSMEM Primitive Characterization

We measure DSMEM performance with three CUDA microbenchmarks based on established GPU memory-hierarchy profiling methods [29], [30]. The first makes every load wait for the preceding load to expose access latency. The second issues many independent reads or stores in parallel to measure their achieved rate. The third repeatedly synchronizes the cluster. Each microbenchmark forces its two CTAs onto different SMs. We report the median of 15 runs.

TABLE II: Profiling on RTX 5090 and H100

GPU	Latency (cycles)			Rate (byte/cycle)			
	Local load	Remote load	Sync	Local read	Remote read	Local store	Remote store
RTX 5090	34	216	404	20.89	3.84	33.01	2.14
H100	30	191	851	20.89	4.77	27.67	21.37

Table II exposes two key facts. First, a remote load has about $6.4\times$ the latency and $4.4\text{--}5.4\times$ lower throughput than a local read. Therefore, only compact partial results should be moved between CTAs and read locally after synchronization; bulk data remains in its owner's SMEM. Second, synchronization and remote-store costs differ substantially between the two GPUs, so one fixed DSMEM penalty cannot represent both devices. The cost model in Section III-D uses the measured local-read, local-store, and remote-store rates, while a matched control kernel captures synchronization and scheduling overhead.

B. Reduction-Reuse Transformation

As shown in Fig. 2(a), consider an input row of width N , with per-element inputs $\mathbf{x}_i$. At stage $\ell \in \{1, \dots, L\}$, index $j \in \{1, \dots, q_\ell\}$ selects one of the q_ℓ scalar statistics, $\mathbf{r}_{<\ell}$ denotes all statistics from earlier stages, $g_{\ell,j}$ is the per-element contribution, and $\oplus$ is an associative reduction such as sum or maximum. After all L stages, the elementwise function h produces the output:

$$r_{\ell,j} = \bigoplus_{i=0}^{N-1} g_{\ell,j}(\mathbf{x}_i, \mathbf{r}_{<\ell}) \quad j = 1, \dots, q_\ell \quad (1)$$

$$y_i = h(\mathbf{x}_i, \mathbf{r}_{1:L}) \quad i = 0, \dots, N-1$$

As shown in Fig. 2(b), we partition each row into P non-overlapping slices $\{\mathcal{I}_c\}_{c=0}^{P-1}$, one per CTA. CTA c retains only its slice, and $\rho_{\ell,c,j}$ denotes its local partial for statistic j at stage ℓ :

$$\rho_{\ell,c,j} = \bigoplus_{i \in \mathcal{I}_c} g_{\ell,j}(\mathbf{x}_i, \mathbf{r}_{<\ell}), \quad r_{\ell,j} = \bigoplus_{c=0}^{P-1} \rho_{\ell,c,j} \quad (2)$$

For retained bytes per element b_{keep} , fixed per-CTA scratch S_{scratch} , and the configured per-CTA dynamic-SMEM limit S_{max} , staging is legal only if:

$$S_{\text{CTA}}(N, P) = \left\lceil \frac{N}{P} \right\rceil b_{\text{keep}} + S_{\text{scratch}} \leq S_{\text{max}} \quad (3)$$

Fig. 2(a) and (b) compare this global-reread schedule with our clustered reduction-reuse schedule. Let D_{fixed} denote per-row HBM traffic that is identical in both schedules (e.g.,

TABLE III: Notation used throughout Section III

Symbol	Definition
M, N	Number of rows, and elements per row
P, L, q_ℓ, s	Cluster CTAs; stages; stage- ℓ statistics; scalar bytes
$b_{\text{keep}}, b_{\text{reread}}, b_B$	Staged; eliminated-reread; baseline B/element
$S_{\text{CTA}}, S_{\text{scratch}}, S_{\text{max}}$	Memory footprint; scratch; SMEM limit
S, f, W	SM count; device frequency; sequential cluster waves
$\bar{B}_{\text{eff}}, B_{\text{SMEM}}^r, B_{\text{DSM}}^w$	Effective source; local-read; remote-store rates
$D_{\text{HBM}}, D_{\text{DSM}}^{w/r}$	HBM and remote DSMEM write/read byte/row
$T_B, T_{\text{save}}, T_{\text{ctrl}}$	Baseline; saved-reread; cluster-control times
$T_{\text{replay}}, T_{\text{deposit}}, T_{\text{DSM}}$	Local-replay; deposit; remote-store times

output writes, which may themselves scale with N , but are not affected by clustering). If we denote the eliminated reread bytes per element by b_{reread} , then:

$$D_{\text{HBM}}^{\text{base}} = D_{\text{fixed}} + N b_{\text{reread}}, \quad D_{\text{HBM}}^{\text{cluster}} = D_{\text{fixed}} \quad (4)$$

For a memory-bound kernel, the traffic-only speedup ceiling is $D_{\text{HBM}}^{\text{base}}/D_{\text{HBM}}^{\text{cluster}}$.

C. Cluster Reduction and Reuse Protocol

At stage ℓ , each CTA compresses values through thread accumulation, warp shuffles, and an SMEM reduction to obtain $\rho_{\ell,c} = (\rho_{\ell,c,1}, \dots, \rho_{\ell,c,q_\ell})$ before performing any cluster-wide communication. Then, it pushes compact partials into peer-owned buffers so that the final combining reads are local.

Fig. 2(c) shows the replicated push/all-gather used by CREDIT. For source CTA c , destination CTA d , and statistic j , let $\text{slot}_d[c, j]$ be the local-SMEM location in d reserved for c , and let $r_{\ell,j}^{(d)}$ be the full statistic reconstructed by d . One of the first P threads writes each partial into the corresponding remote slot:

$$\text{slot}_d[c, j] \leftarrow \rho_{\ell,c,j}, \quad r_{\ell,j}^{(d)} = \bigoplus_{c=0}^{P-1} \text{slot}_d[c, j] \quad (5)$$

The first barrier exposes pushed partials. Every CTA then combines only local slots; a second barrier protects scratch reuse. Unlike a design where a single root CTA gathers all partials and every peer then rereads the result, this replicated all-gather lets every CTA reconstruct the full statistic independently, avoiding both a serial bottleneck at the root and a second round of remote reads at the cost of replicated scalar stores. Let $D_{\text{DSM}}^{w/r}$ denote total remote write and read bytes per row, let s be the bytes per scalar statistic, and let K count cluster-wide barriers per row. The protocol costs are

$$D_{\text{DSM}}^w = sP(P-1) \sum_{\ell=1}^L q_\ell, \quad D_{\text{DSM}}^r = 0, \quad K = 2L \quad (6)$$

The factor $P(P-1)$ counts P source CTAs writing to their $P-1$ remote peers; subsequent reads are local because each destination owns its slots. CTA c finally applies h to the elements in its retained slice and writes its output segment.

D. Profitability and Candidate Selection

The model is evaluated for M rows and each legal cluster size P . Using peak HBM or L2 bandwidth to predict how long the baseline takes to stream a row is unreliable since cache residency changes with row shape, and a one-CTA kernel achieves very different fractions of peak bandwidth on RTX 5090 versus H100. Therefore, we use a single timing of the selected non-DSMEM CUDA path, $T_B(M, N)$. Since this timing requires only the baseline kernel, it can be measured before any clustered variant is generated or tuned, making it a cheap first screen rather than part of the search itself.

$$\begin{aligned} \hat{B}_{\text{eff}}(M, N) &= \frac{MNb_B}{T_B(M, N)} \\ T_{\text{save}}(M, N) &= \frac{MNb_{\text{reread}}}{\hat{B}_{\text{eff}}} = T_B(M, N) \frac{b_{\text{reread}}}{b_B} \end{aligned} \quad (7)$$

Eq. (7) replaces a device peak with the rate actually achieved by the baseline at the same shape; it requires no DSMEM workload timing and has no fitted coefficient. We independently launch a work-free control kernel with the same grid, P , dynamic-SMEM footprint, and $K = 2L$ barriers. The excess time this control takes relative to a one-CTA control, $T_{\text{ctrl}}(M, N, P)$, captures cluster launch, scheduling, occupancy, and synchronization costs.

When staging limits residency to one CTA per SM, a row cluster occupies P SMs, so at most $\lfloor S/P \rfloor$ row clusters execute concurrently on a device with S SMs. Let $W(M, P)$ be the resulting number of sequential cluster waves, f the SM frequency, B_{SMEM}^r the achieved local-SMEM read rate, and $B_{\text{DSM}}^w(P)$ the per-CTA remote-store issue rate for cluster size P . The incremental local replay and remote-store times are

$$\begin{aligned} W(M, P) &= \left\lceil \frac{M}{\lfloor S/P \rfloor} \right\rceil \\ T_{\text{replay}} &= \frac{W(M, P) \lfloor N/P \rfloor b_{\text{keep}}}{f B_{\text{SMEM}}^r} \\ T_{\text{DSM}} &= \frac{s W(M, P) (P-1) \sum_{\ell} q_\ell}{f B_{\text{DSM}}^w(P)} \end{aligned} \quad (8)$$

Because the P source CTAs issue their stores concurrently on different SMs, their combined wall-clock cost is not P times a single CTA's cost. So, although Eq. (6) counts P source CTAs for byte accounting, Eq. (8) does not multiply elapsed time by P . T_{replay} charges the incremental local-SMEM read needed to apply h . The initial SMEM deposit is issued alongside the compulsory global load that streams each element from HBM. We call this load's duration T_{input} , the source-stream time. Let T_{SMEM}^w be the deposit's standalone local-store issue time and define $[z]^+ = \max(z, 0)$. We charge only the non-overlapped residual $T_{\text{deposit}} = [T_{\text{SMEM}}^w - T_{\text{input}}]^+$, which is zero for both measured devices because local-SMEM store issue is faster than the per-SM input stream. The predicted benefit is

$$\Delta T(M, N, P) \approx T_{\text{save}} - T_{\text{ctrl}} - T_{\text{replay}} - T_{\text{deposit}} - T_{\text{DSM}} \quad (9)$$

We define N^* as the smallest tested width for which Eq. (3) holds and $\Delta T > 0$ for at least one legal P ; final kernel tuning may choose a different legal P because this is a pair and crossover predictor rather than a cycle-exact model. The non-DSMEM timing makes the model profile-guided, but evaluating three legal P values is inexpensive relative to compiling and benchmarking three clustered kernels. The same reasoning rejects three cases by construction: elementwise and scalar-output reductions when $b_{\text{reread}} \approx 0$, scans when ordered dependencies grow K , and selection or stencil kernels when remote state grows with N rather than $\sum_{\ell} q_\ell$. In each case, increasing N does not help. A reduction with no reread has nothing to amortize; growing K adds synchronization faster than width can offset it, and remote state that scales with N defeats the compact-partial assumption.

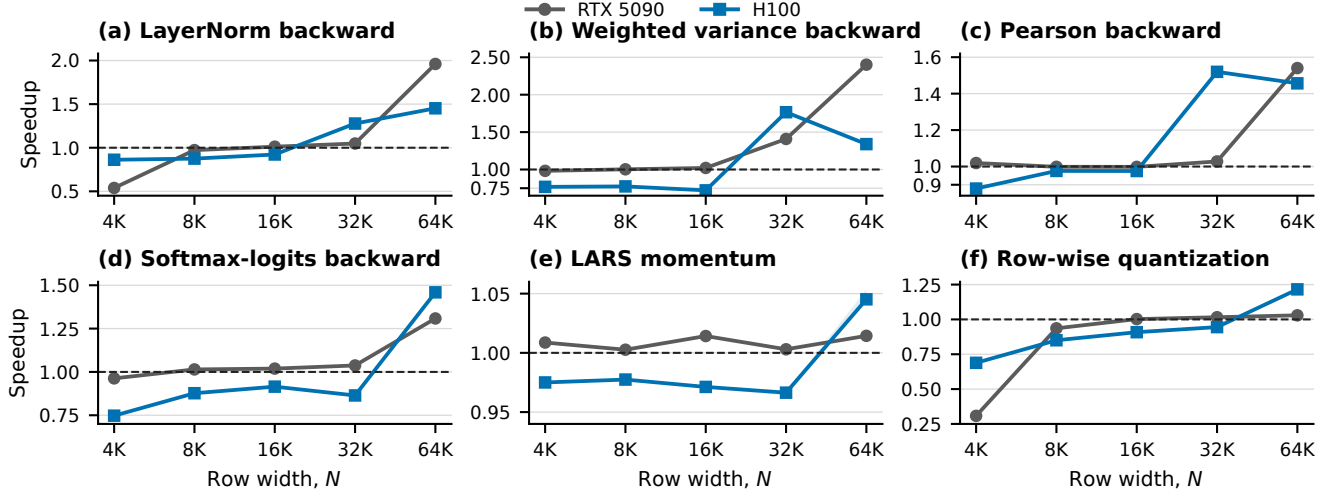


Fig. 3: CREDIT’s speedups over the fastest baseline across workloads and shapes

IV. EVALUATION AND DISCUSSION

A. Experimental Setup and Baselines

We evaluate CREDIT on an NVIDIA RTX 5090 and an H100 SXM. Both use CUDA 13.0, PyTorch 2.11.0 and Triton 3.6.0. Our benchmark suite contains six reduction-reuse workloads with FP32 input format from five categories: LayerNorm [31] and weighted-variance backward, Pearson-correlation backward, softmax-logits backward, LARS momentum [32], and row-wise int8 quantization. LayerNorm uses $M = 2048$ rows; the others use $M = 4096$. We sweep $N \in \{4096, 8192, 16384, 32768, 65536\}$ and legal cluster sizes $P \in \{2, 4, 8\}$, selecting the fastest measured P for performance results.

We compare CREDIT against three non-DSMEM baselines: `torch.compile` [14], Triton [21], and a CUDA baseline that selects a local-SMEM-staged kernel when the row fits in one CTA, or a global-reread kernel otherwise. The first two represent optimized framework and expert-DSL implementations, while the CUDA baseline provides an ablation mechanism. Reported times exclude `torch.compile` compilation time and Triton’s autotuning search time; both are measured once and amortized outside the timed region. We report the speedup as $S_B = T_B / T_{\text{CREDIT}}$, where T_{CREDIT} is the CREDIT kernel runtime. Thus, $S_B > 1$ means CREDIT is faster than baseline B , and “best” denotes the fastest of the three baselines at that shape.

B. Workload-by-Workload Scaling

Fig. 3 shows width- and architecture-dependent crossovers. At 4K, CREDIT is slower than the one-CTA baseline (geometric-mean speedups are $0.738\times$ and $0.815\times$ for RTX 5090 and H100, respectively), because the one-CTA baseline exploits cache or local staging without cluster overhead. At 64K, CREDIT beats the fastest baseline on all six workloads. Table IV decomposes this result by baseline: the

TABLE IV: Geometric-mean CREDIT speedup at $N = 64K$ over the baselines across all workloads

GPU	<code>torch.compile</code>	Triton	CUDA	Best
RTX 5090	$1.617\times$	$1.937\times$	$1.980\times$	$1.466\times$
H100	$1.469\times$	$1.525\times$	$1.533\times$	$1.318\times$

geometric-mean speedups over the fastest baseline are $1.466\times$ on RTX 5090 and $1.318\times$ on H100 across all workloads.

As expected, the crossover point changes as a function of the hardware and workload. RTX 5090 first exceeds parity at 16K and wins every workload at 32K, whereas H100 wins every workload only at 64K, demonstrating that device costs shift the amortization point. This result is consistent with Table II, where H100’s higher barrier latency (851 vs. 404 cycles) raises T_{ctrl} and shifts the amortization point to wider rows. Overall, CREDIT beats the fastest baseline at 22 points in Fig. 3 (i.e., workload-shape pairs) when running on RTX 5090, and 9 points when running on H100.

The workload spread reinforces this interpretation. Weighted variance provides the largest RTX 5090 gain, rising from $0.982\times$ at 4K to $2.401\times$ at 64K, while Pearson and softmax-logits exhibit later, moderate crossovers. LARS is the limiting positive case, reaching only $1.014\times$ and $1.045\times$ at 64K, and quantization crosses the best baseline at 16K on RTX 5090 but only at 64K on H100. Thus, width creates an opportunity, but reusable traffic, baseline efficiency, and cluster overhead determine whether CREDIT realizes it.

C. Cost-Model Validation

Fig. 3 establishes practical performance against the fastest baseline, whereas the model asks the narrower mechanism question of whether CREDIT beats matched non-DSMEM CUDA without ever timing a DSMEM workload. We obtain local-SMEM and DSMEM rates from Table II in Section III-A; the work-free control supplies T_{ctrl} for each

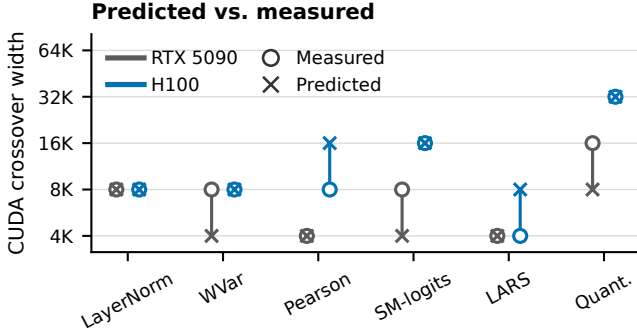


Fig. 4: Cost-model validation. Crosses and open circles show the predicted and measured smallest width at which DSMEM beats matched non-DSMEM CUDA; vertical segments denote prediction error. Each vertical step doubles the row width

TABLE V: Profitability-pair accuracy across six workloads and five shapes per GPU

Model	RTX 5090	H100	Overall
Peak bandwidth + serialized staging	27/30	16/30	43/60
CREDIT: achieved bandwidth + overlap	27/30	28/30	55/60

(M, N, P); the non-DSMEM kernel supplies T_B ; and static analysis supplies $b_B, b_{\text{keep}}, b_{\text{reread}}$, and $\{q_\ell\}$.

Table V evaluates 60 profitability pairs: five widths, six workloads, and two GPUs. CREDIT reaches 90.0% accuracy on RTX 5090, 93.3% on H100, and 91.7% overall. Replacing device peaks with the baseline’s achieved bandwidth and overlapping the compulsory input load with the SMEM deposit improves H100 accuracy from 16/30 to 28/30 and overall accuracy from 43/60 to 55/60, without fitting a coefficient to DSMEM workload timings. The five remaining errors occur only among the 32 points where the baseline instead uses the local-SMEM-staged kernel, at shapes near the staging boundary, with measured DSMEM/CUDA ratios of 0.996, 0.986, and 0.994 on RTX 5090 and 1.015 and 1.022 on H100, all within 2.2% of parity. All 28 points using a pure global-reread CUDA baseline are classified correctly.

Fig. 4 provides a stricter crossover test. CREDIT’s prediction is exact for seven of the 12 device–workload pairs and differs by one tested factor-of-two width for the other five; no crossover is missed by more than one interval. The result supports the intended use of Eq. 9: it is a mechanism screen that rejects clearly unprofitable shapes and narrows the tuning region, not a cycle-exact runtime predictor or a replacement for final cluster-size search. Identifying a mechanism opportunity does not by itself guarantee a win over a separately optimized framework schedule; practical deployment requires both the model screen and the fastest-baseline comparison in Fig. 3.

Negative controls further validate the screening model’s structural rules. One-pass reductions expose no recoverable reread traffic, scans introduce ordered dependencies, and selection or stencil kernels communicate noncompact state; none

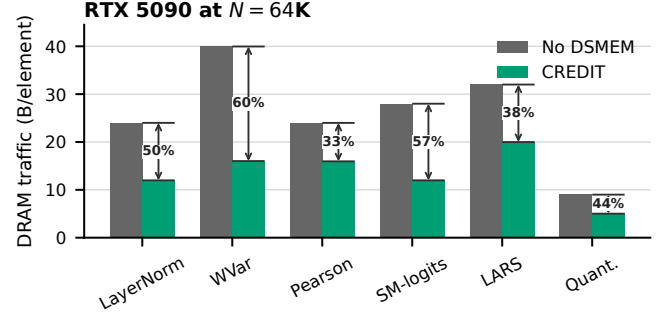


Fig. 5: Nsight Compute DRAM traffic on RTX 5090 at $N = 65536$. Bars compare matched non-DSMEM CUDA with CREDIT; labels give the traffic reduction from owner-local staging

creates the owner-local reduction–reuse pattern required by CREDIT. Rejecting these cases before clustered code generation shows that CREDIT identifies opportunity from dataflow rather than vector width alone.

D. Traffic Validation

Fig. 5 independently validates the root cause of speedups enabled by CREDIT. CREDIT reduces measured RTX 5090 DRAM traffic by 33–60%: LayerNorm drops from 24 to 12 B/element, weighted variance from 40 to 16, Pearson from 24 to 16, softmax-logits from 28 to 12, LARS from 32 to 20, and quantization from 9 to 5. Output traffic is unchanged; the reduction comes from retaining owner-local slices and replacing repeated global reads with compact statistics. Traffic savings alone do not predict the speedup: LARS removes 38% of traffic but gains only 1–5% over the best baseline at 64K, because the framework baseline is already substantially faster than matched CUDA. This is exactly why CREDIT values avoided source time at the baseline’s achieved rate rather than at raw traffic reduction. It explicitly subtracts cluster control, replay, and DSMEM transport, while a purely traffic-based metric would overstate LARS’s practical benefit.

V. CONCLUSION

DSMEM is a selective reduction-reuse mechanism, not a general fusion fabric. CREDIT correctly classifies 55/60 workload-shape pairs across RTX 5090 and H100. As problem size increases, CREDIT beats the best baseline on all six workloads, with geometric-mean speedups of $1.466\times$ and $1.318\times$, respectively. Clustered variants should be generated only when recovered source time amortizes distributed staging and compact inter-CTA reduction. Our future work will explore integrating CREDIT with more complex GPU workloads and graph models [33], [34].

Disclosure: Dr. Ogras is affiliated with Samsung Austin Research & Development Center and Advanced Computing Lab (SARC/ACL). This relationship has been approved under applicable outside activities policies.

REFERENCES

- [1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, “Ai and memory wall,” *IEEE Micro*, vol. 44, no. 3, pp. 33–39, May 2024.
- [2] S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.
- [3] NVIDIA, *NVIDIA V100 Tensor Core GPU Datasheet*, NVIDIA Corporation, 2020. [Online]. Available: <https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf>
- [4] NVIDIA, *NVIDIA A100 Tensor Core GPU*, NVIDIA Corporation, 2020. [Online]. Available: <https://www.nvidia.com/en-us/data-center/a100/>
- [5] NVIDIA, “NVIDIA H100 Tensor Core GPU,” <https://www.nvidia.com/en-us/data-center/h100/>, 2022.
- [6] NVIDIA, “NVIDIA HGX Platform Specifications,” <https://www.nvidia.com/en-us/data-center/hgx/>, 2026.
- [7] NVIDIA, “NVIDIA DGX B200 Specifications,” <https://www.nvidia.com/en-us/data-center/dgx-b200/>, 2026.
- [8] NVIDIA, “NVIDIA RTX Blackwell GPU Architecture,” <https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf>, 2025.
- [9] NVIDIA, “NVIDIA Hopper Architecture In-Depth,” in *NVIDIA Technical Blog*, 2022. [Online]. Available: <https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/>
- [10] NVIDIA, “CUDA C++ Programming Guide: Distributed Shared Memory,” <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#distributed-shared-memory>, 2026.
- [11] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Re, “Flashattention: Fast and memory-efficient exact attention with IO-awareness,” in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: <https://openreview.net/forum?id=H4DqfPSibmx>
- [12] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=mZn2Xyh9Ec>
- [13] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, “TVM: An automated End-to-End optimizing compiler for deep learning,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594. [Online]. Available: <https://www.usenix.org/conference/osdi18/presentation/chen>
- [14] J. Ansel, E. Yang, H. He, N. Gimselshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhersch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala, “Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS ’24. ACM, Apr. 2024, pp. 929–947.
- [15] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, “Dnnfusion: accelerating deep neural networks execution with advanced operator fusion,” in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI ’21. ACM, 2021, pp. 883–898.
- [16] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica, “Ansor: Generating High-Performance tensor programs for deep learning,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 863–879. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/zheng>
- [17] H. Zhu, R. Wu, Y. Diao, S. Ke, H. Li, C. Zhang, J. Xue, L. Ma, Y. Xia, W. Cui, F. Yang, M. Yang, L. Zhou, A. Cidon, and G. Pekhimenko, “ROLLER: Fast and Efficient Tensor Compilation for Deep Learning,” in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 233–248. [Online]. Available: <https://www.usenix.org/conference/osdi22/presentation/zhu>
- [18] Z. Zheng, X. Yang, P. Zhao, G. Long, K. Zhu, F. Zhu, W. Zhao, X. Liu, J. Yang, J. Zhai, S. L. Song, and W. Lin, “Astitch: enabling a new multi-dimensional optimization space for memory-intensive ml training and inference on modern simt architectures,” in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’22. ACM, Feb. 2022, pp. 359–373.
- [19] Y. Shi, Z. Yang, J. Xue, L. Ma, Y. Xia, Z. Miao, Y. Guo, F. Yang, and L. Zhou, “Welder: Scheduling deep learning memory access via tile-graph,” in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 701–718. [Online]. Available: <https://www.usenix.org/conference/osdi23/presentation/shi>
- [20] D. Zhuang, Z. Zheng, H. Xia, X. Qiu, J. Bai, W. Lin, and S. L. Song, “MonoNN: Enabling a new monolithic optimization space for neural network inference tasks on modern GPU-Centric architectures,” in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 989–1005. [Online]. Available: <https://www.usenix.org/conference/osdi24/presentation/zhuang>
- [21] P. Tillet, H. T. Kung, and D. Cox, “Triton: an intermediate language and compiler for tiled neural network computations,” in *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. PLDI ’19. ACM, 2019, pp. 10–19.
- [22] B. F. Spector, S. Arora, A. Singhal, A. Parthasarathy, D. Y. Fu, and C. Re, “Thunderkittens: Simple, fast, and Adorable kernels,” in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: <https://openreview.net/forum?id=0fJfVOSUra>
- [23] W. Luo, R. Fan, Z. Li, D. Du, Q. Wang, and X. Chu, “Benchmarking and dissecting the nvidia hopper gpu architecture,” in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2024, pp. 656–667.
- [24] T. Lühnen, T. Marschner, and S. Lal, “Benchmarking thread block cluster,” in *2024 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2024, pp. 1–7.
- [25] Z. Jin, C. Rocca, J. Kim, H. Kanan, M. Rhu, A. Bakhoda, T. M. Aamodt, and J. Kim, “Uncovering real gpu noc characteristics: Implications on interconnect architecture,” in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Nov. 2024, pp. 885–898.
- [26] Z. Huang, Y. Zhou, Z. Liu, X. Luo, Y. Diao, M. Guo, J. Zhai, Y. Feng, C. Zhang, A. Wu, and J. Leng, “Flashfuser: Expanding the scale of kernel fusion for compute-intensive operators via inter-core connection,” in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Jan. 2026, pp. 1–14.
- [27] X. Luo, Z. Liu, Y. Zhou, S. Fang, Z. Huang, Y. Feng, C. Zhang, S. Sun, Z. Zheng, J. Leng, and M. Guo, “Clusterfusion: Expanding operator fusion scope for LLM inference via cluster-level collective primitive,” in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. [Online]. Available: <https://openreview.net/forum?id=eX0m4qMYVN>
- [28] W. Guo, T. Zadouri, and T. Dao, “Getting Memory-Bound Kernels to Speed-of-Light,” Dao-AILab Quack technical article, 2025. [Online]. Available: <https://github.com/Dao-AILab/quack/blob/main/media/2025-07-10-membound-sol.md>
- [29] X. Mei and X. Chu, “Dissecting gpu memory hierarchy through microbenchmarking,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 72–86, Jan. 2017.
- [30] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, “Dissecting the nvidia volta gpu architecture via microbenchmarking,” Apr. 2018.
- [31] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” Jul. 2016.
- [32] Y. You, I. Gitman, and B. Ginsburg, “Large batch training of convolutional networks,” Aug. 2017.
- [33] Z. Li, T.-W. Huang, and U. Ogras, “SET: Stream-Event-Triggered Scheduling for Efficient CUDA Graph Pipelines,” in *International European Conference on Parallel and Distributed Computing (Euro-Par)*, 2026.
- [34] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, “Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System,” in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2022, pp. 1303–1320.