

TIDES: Tiered Block-Sparse Tensor Contraction with Streaming Transpose on GPUs

Zecheng Li[1], Sri Harshavardhan Reddy Deverapalli[1], Yanbo Zhao[1]
Frank Mueller[1], Karl Pierce[2], Jiajia Li[1]

[1] North Carolina State University
[2] Flatiron Institute

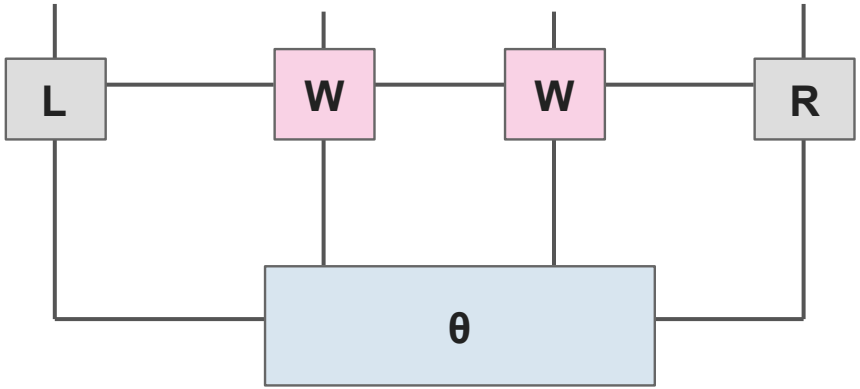
Code: github.com/tensorworld/TIDES



Two-site DMRG tensor network

An iterative algorithm for quantum many-body problems.

DMRG network



θ: Two-site state tensor; W: Operator tensors
L, R: Left/right environment tensors

1. Optimize the local state

S1–S3: repeated contractions inside an iterative solver.

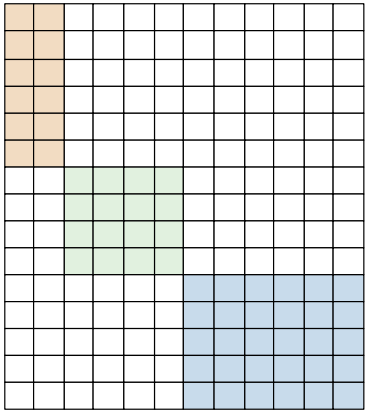
2. Move to the next site pair

Split the updated state. E1–E2 update the environments.

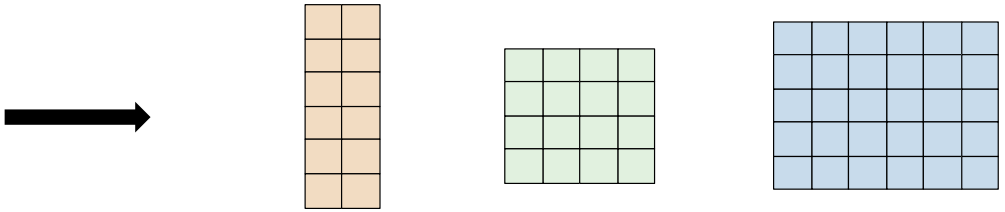
Tensor contractions: 76–81% of sweep runtime in our A4500 profile.

Symmetry and block sparsity

Only combinations of sector labels allowed by a conservation rule are stored.



Symmetry sectors



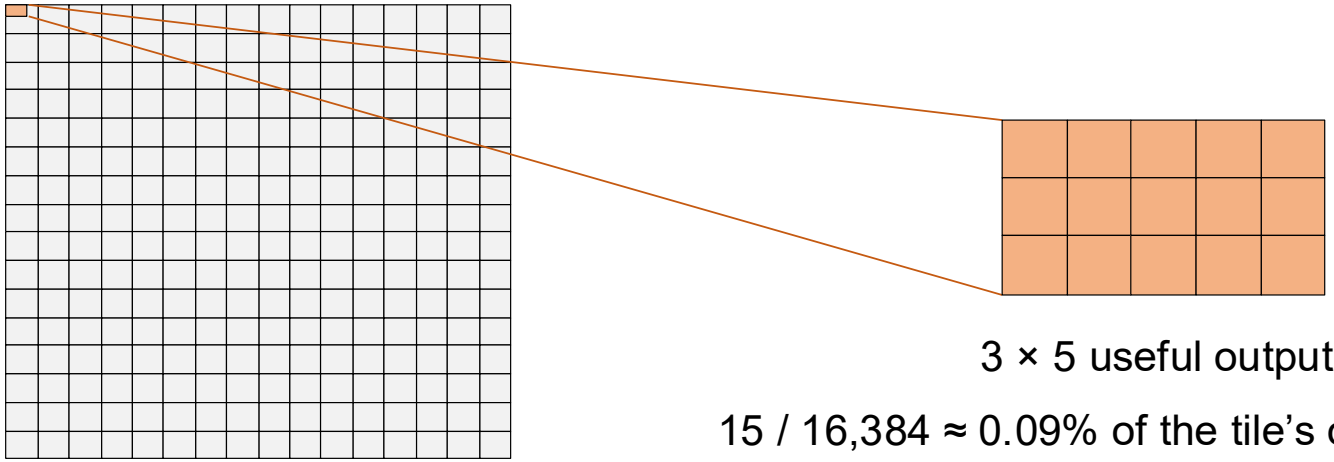
Dense blocks with different shapes

Matched pair: $(M \times K) \cdot (K \times N) \rightarrow (M \times N)$

Five orders of magnitude in block size. Up to 100K matched block pairs.

GEMM tile utilization

Logical output shape: 3 × 5. Candidate tile: 128 × 128.



128 × 128 tile

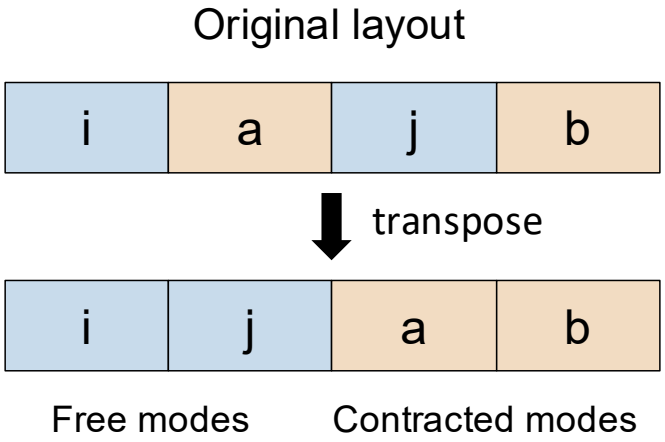
3 × 5 useful output

15 / 16,384 ≈ 0.09% of the tile's output positions

Narrow and compact pairs need smaller execution units.

Tensor layout and contraction

Free modes and contracted modes may be interleaved in physical storage.



Direct contraction

Convert coordinates while reading the original layout.

TTGT

Transpose into matrix layout, then call grouped GEMM.

Direct kernels avoid a transpose. TTGT needs bounded workspace.

Three challenges of block sparsity

Three properties of the workload drive the design.

1 Size heterogeneity → **Tiered dispatch**

Scalar outputs to blocks with 10^5+ elements

2 Layout and workspace → **Streaming TTGT**

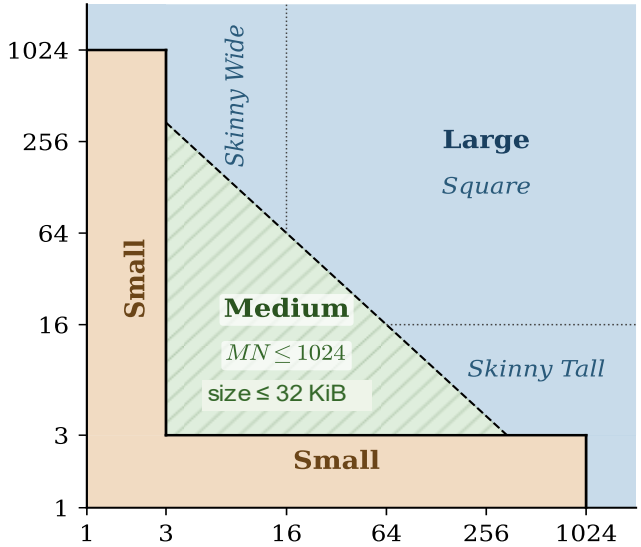
Interleaved modes make full transposes costly

3 Shape heterogeneity → **Hypertile**

Thousands of irregular blocks amplify launch cost

Tiered dispatch by block shape

Challenge 1: match the execution unit to each block pair.



Small → lanes

Narrow outputs; register accumulation

Medium → CTA

Compact pairs; shared-memory reuse

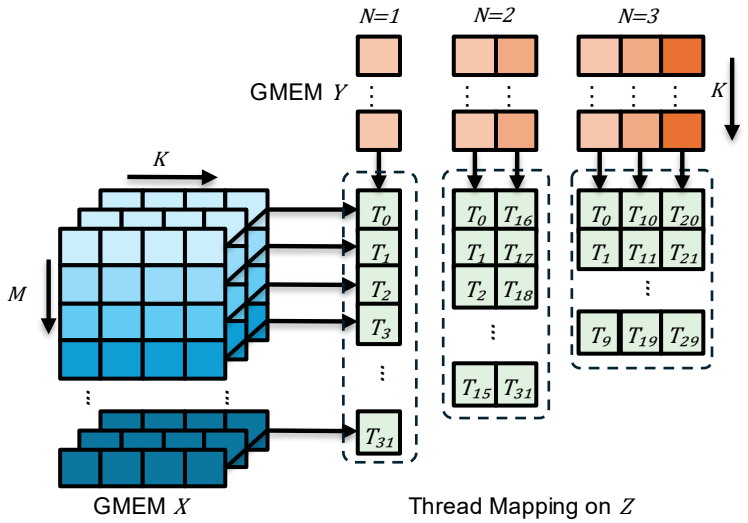
Large → GEMM

Aspect-ratio bins; streaming TTGT

Reusable host plan → concurrent tier execution → shared output Z

Small tier: warp-level contraction

Narrow outputs need little tiling overhead.

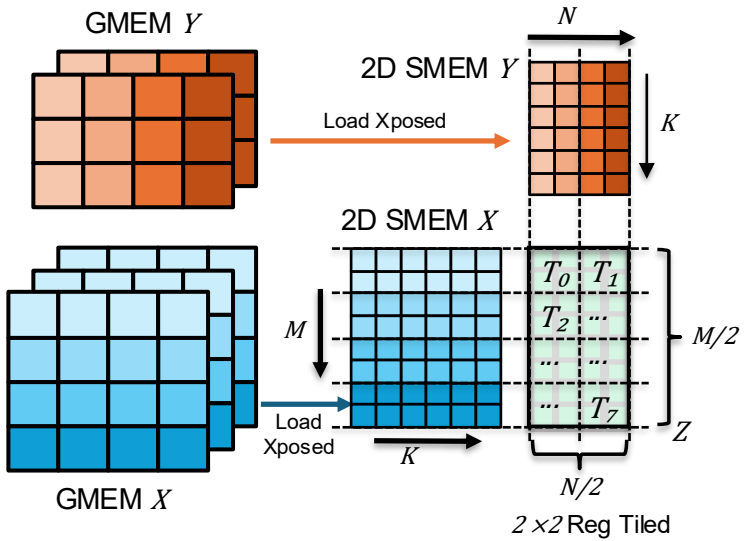


- One lane owns one output
- Lookup-table lane mapping
- Register accumulation over K

Scalar output, K ≥ 128: one 256-thread CTA reduces over K.

Medium tier: shared-memory kernel

Fuse coordinate conversion into loads; reuse data on chip.

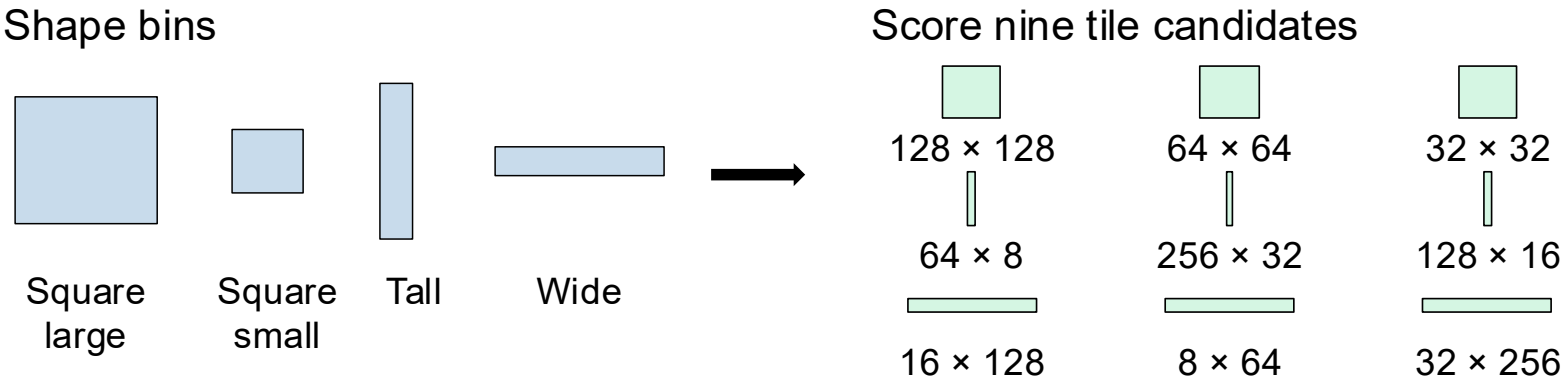


- Load and rearrange once
- Reuse the operands in shared memory
- Each thread owns a 2 × 2 register tile

The output partition adapts to the block shape.

Large tier: GEMM tile selection

Group by aspect ratio, then select a CUTLASS tile.

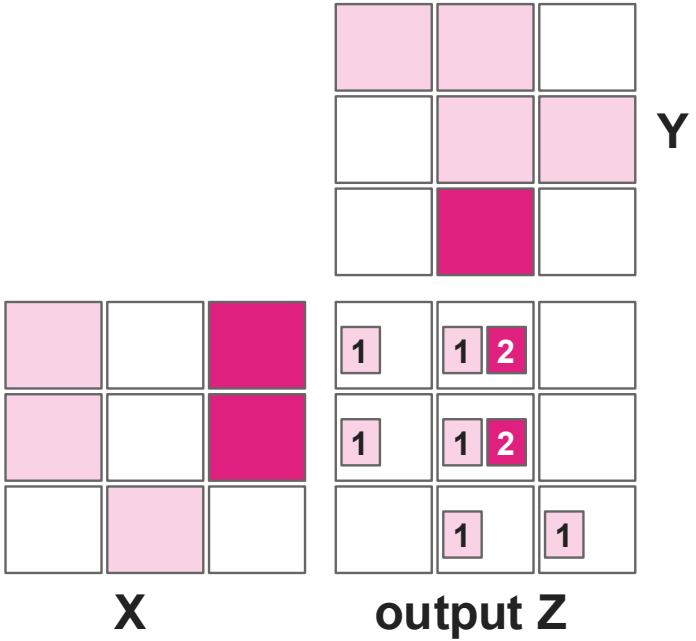


Offline tile efficiency η × useful FLOPs / padded FLOPs

Balance padding cost against the throughput of the tile itself.

Group partitioning

Large tier: each group fits one transpose-buffer slot.



1 Sort by output write number

Separate contributions to the same output across groups that execute in order.

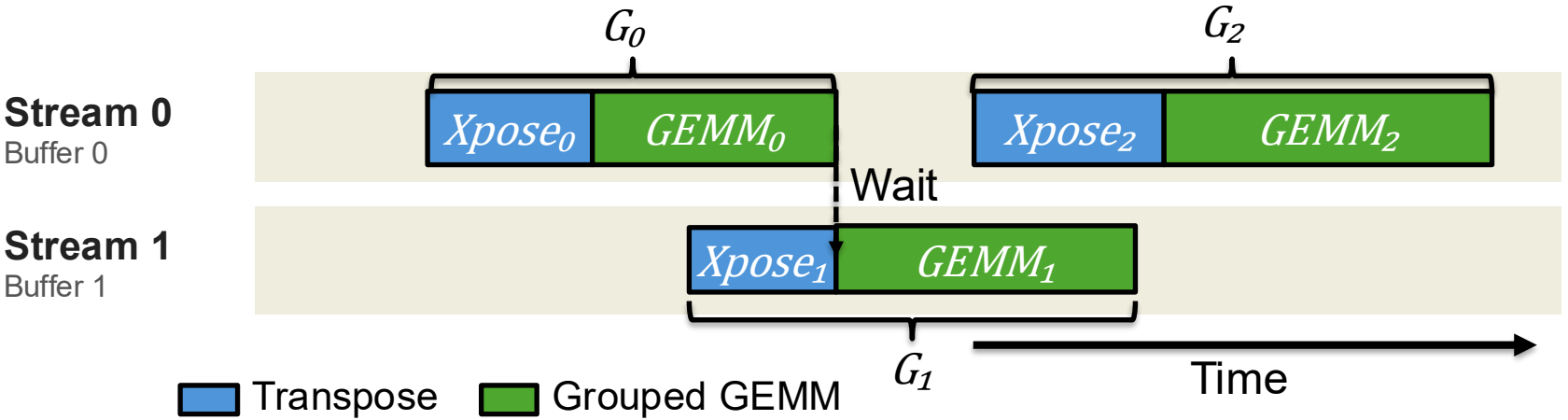
2 Sort by shared input block

Keep reuse together; transpose an input once per group.

Group 0 **Group 1**

Streaming transpose and GEMM

Challenge 2: group i computes while group i + 1 transposes.

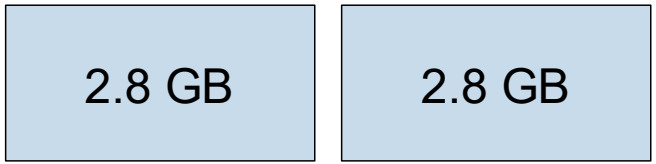


The buffers alternate and consecutive GEMMs remain serialized to allow non-atomic accumulation.

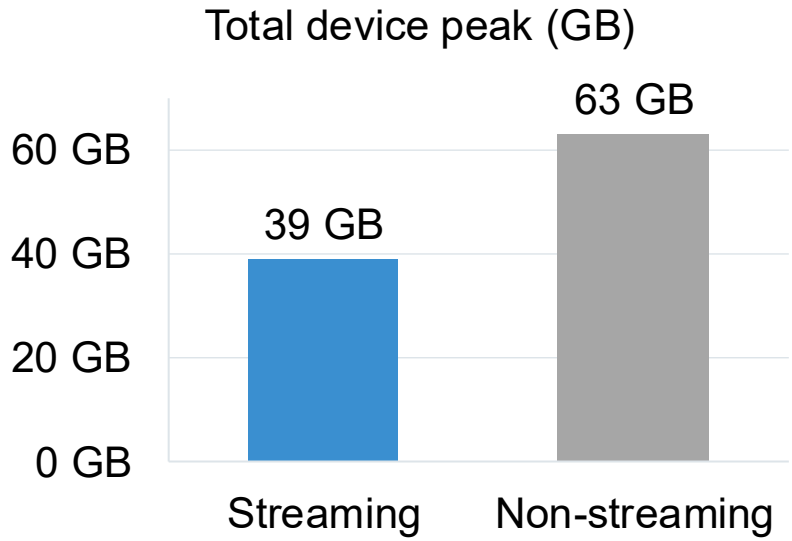
Streaming workspace

FP64 on H100, canonical nfparity S3, 82 streaming groups

Two reusable slots
Each fits the largest pair



Transpose workspace: 5.6 GB



Hypertile: adaptive transpose tiles

Challenge 3: adapt transpose tiles under a fixed shared-memory budget.

X[a, b, c] → Y[b, a, c] **Keep C = 2 fully tiled**

INPUT: A × (B · C)

A: 16 → 32

Contiguous read: 32 → 32

OUTPUT: B × (A · C)

B: 16 (unchanged)

Contiguous write: 32 → 64

Initial tile (16, 16, 2)

Grow A: (32, 16, 2)

Group by effective order → sort shapes / permutations → reuse offsets

Experimental setup

H100 NVL (94 GB), cuTENSOR 2.6.0, CUTLASS backend

4 symmetries × 5 contractions × 2 layouts = 40 cases

FP64 / FP32 • median of 5 runs • GPU kernel time

Symmetry regime	Conserved labels	Block-pair mix
nfparity	Particle-number parity	100% Large
nf	Particle number	About 90% Large
sznf	Spin + particle number	62% Large
kysznf	Momentum + spin + number	Up to 82% Medium + Small

Canonical kysznf S3 contains 22,879 matched pairs.

FP64 contraction performance

H100, speedup = cuTENSOR time / TIDES time

2.14× geometric mean, 38/40 wins **up to 5.97×**

Canonical

	E1	E2	S1	S2	S3
nfparity	2.55	0.78	2.21	2.65	2.76
nf	2.05	1.27	1.86	2.19	1.68
sznf	1.74	1.77	2.02	1.96	0.94
kysznf	2.14	3.30	2.07	2.43	2.30

Internally permuted

	E1	E2	S1	S2	S3
nfparity	2.53	1.31	2.17	2.55	2.75
nf	2.14	2.52	2.12	2.19	2.15
sznf	5.97	4.01	2.04	2.08	1.80
kysznf	2.53	5.34	2.10	1.03	2.22

Speedup = cuTENSOR / TIDES time. Green > 1×; red < 1×.

Regressions: nfparity E2 0.78× (GEMV-like) and sznf S3 0.94×.

FP32 contraction performance

H100, speedup = cuTENSOR time / TIDES time

1.69× geometric mean, 32/40 wins **up to 9.86×**

Canonical

	E1	E2	S1	S2	S3
nfparity	1.03	1.14	0.99	0.94	1.03
nf	1.07	1.80	1.10	1.03	0.80
sznf	1.38	3.40	1.39	1.43	0.73
kysznf	2.88	6.23	2.10	3.15	2.59

Internally permuted

	E1	E2	S1	S2	S3
nfparity	0.95	2.93	1.11	0.95	1.05
nf	1.11	4.83	1.33	1.08	0.84
sznf	7.11	6.88	1.74	1.52	0.97
kysznf	2.88	9.86	2.13	1.30	1.62

Speedup = cuTENSOR / TIDES time. Green > 1×; red < 1×.

Peak: internally permuted kysznf E2, 9.86×. Large-tier FP32 trails GETT.

Canonical layout is faster in absolute time

Kernel time summed over E1–S3. Mode fusion can inflate the work.

TIDES FP64	Canonical	Internally permuted	Canonical speedup
kysznf	1.4 ms	2.6 ms	**1.86×**
sznf	4.9 ms	6.7 ms	**1.37×**
nf	26.0 ms	52.6 ms	**2.02×**
nfparity	211.7 ms	379.1 ms	**1.79×**

S3 FLOPs rise 2.2×–10.3× as fusion coarsens symmetry sectors.

TIDES makes native layouts fast while preserving the work symmetry removes.

Per-tier scaling and hardware portability

Synthetic contractions isolate the behavior of each tier.

Tier	Peak speedup	Scope of the measurement
Small	33× FP32	Per-configuration peak, canonical K = 4
Medium	7.9× FP32	MN = 16
Large	3.25× FP64	M = N = 4624

RTX A4500: 2.42× FP64 without retuning

37/37 fitting cases win. Refreshing tile efficiency η gives 2.76×.

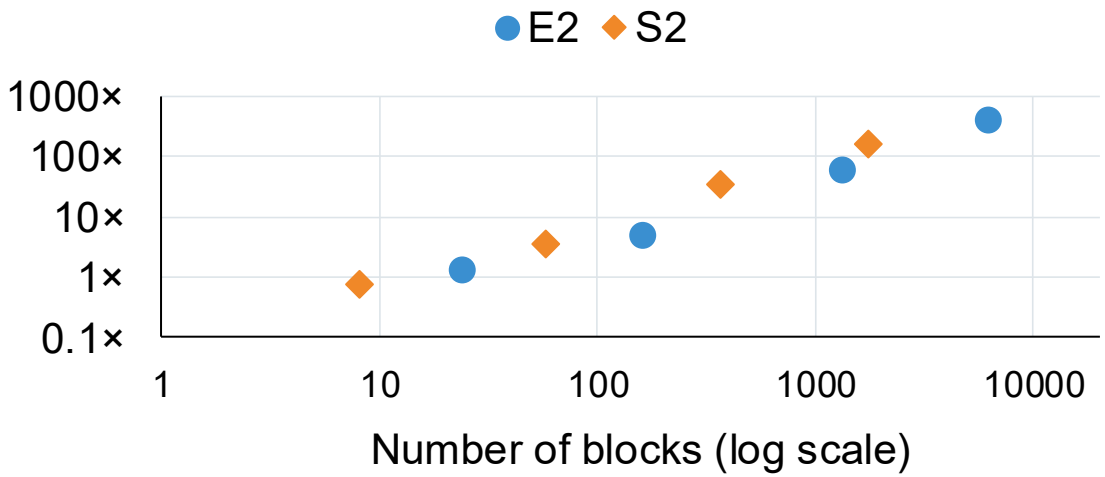
Newer-library check: H100, expanded 60-case set

vs. cuTENSOR 2.8.1: 2.00× FP32 / 2.56× FP64 geometric mean

Hypertile benefits grow with block count

FP64 on H100, first input tensor of E2 and S2 in each symmetry regime

Speedup over cuTENSOR



6,159 blocks: 377×

kysznf E2

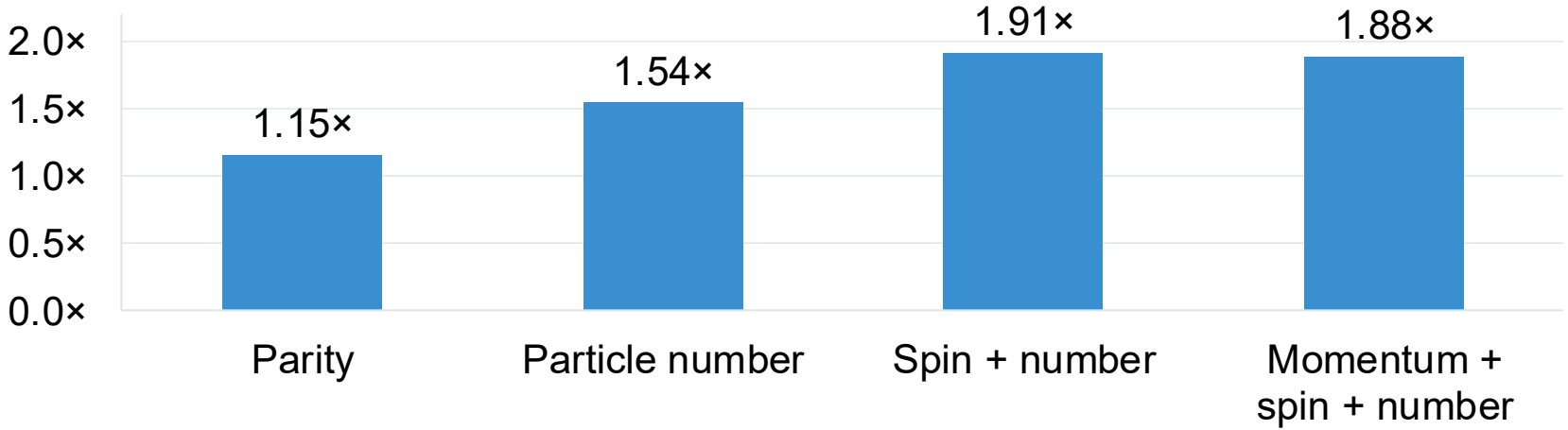
8 blocks: 0.74×

nfparity S2

Grouped dispatch amortizes the cost of launching one transpose per block.

DMRG sweep performance

FP32 on H100, ITensors.jl, cuTENSOR block-sparse baseline



Reuse plans while metadata matches. Sweep speedups include host planning.

TIDES: match execution to native blocks

1.15×–1.91× faster DMRG sweeps in the evaluated H100 FP32 runs.

Tiered dispatch	Direct kernels for narrow or compact pairs
Streaming TTGT	Reuse buffers and overlap transpose with GEMM
Hypertile	Grouped launches, reused indexing, adaptive tiles

github.com/tensorworld/TIDES

TensorSuite: benchmark tensors

A public collection of real tensors and synthetic tensor generators.

Tensors used in this work are available on TensorSuite.

Two forms of sparsity

Element-wise: individual nonzero entries

Block-wise: dense nonzero blocks

Real data + synthetic generators

Explore datasets or generate controlled workloads.



https://tensorworld.github.io/TensorSuite/