TECHNICAL PAPER

Scalable Generation of Distributed-Memory Mappings for Large Tensor Networks

Botao Wu (presenter) and Martin Kong

The Ohio State University

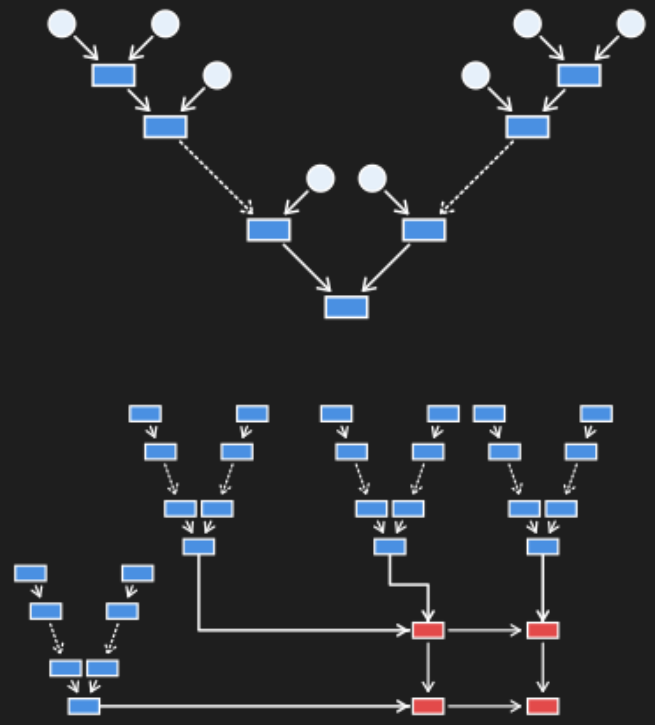
THE OHIO STATE UNIVERSITY
COLLEGE OF ENGINEERING
(CSE department)

SC26 hpc unites | Chicago

Motivation

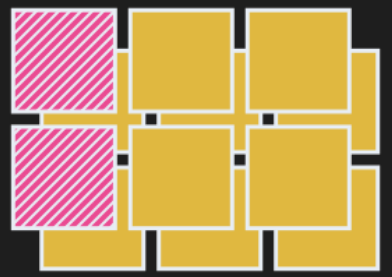
Tensor computation DAG at scale

- Tensor-computation DAGs support a broad class of applications:
  - Computational chemistry
  - Quantum physics simulation
  - Artificial intelligence
  - …
- Large in scale:
  - Tens to hundreds of operators
  - Tensorial data with large memory footprints
  - Cluster-scale execution



Limitation of previous systems

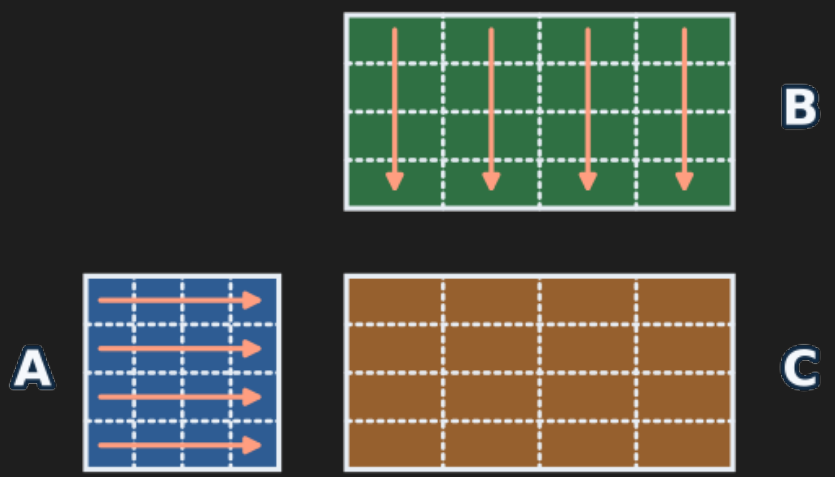
Previous systems lack flexibility in choosing the processing element (PE) grid.



→ We address this limitation with **DiMage-HG**

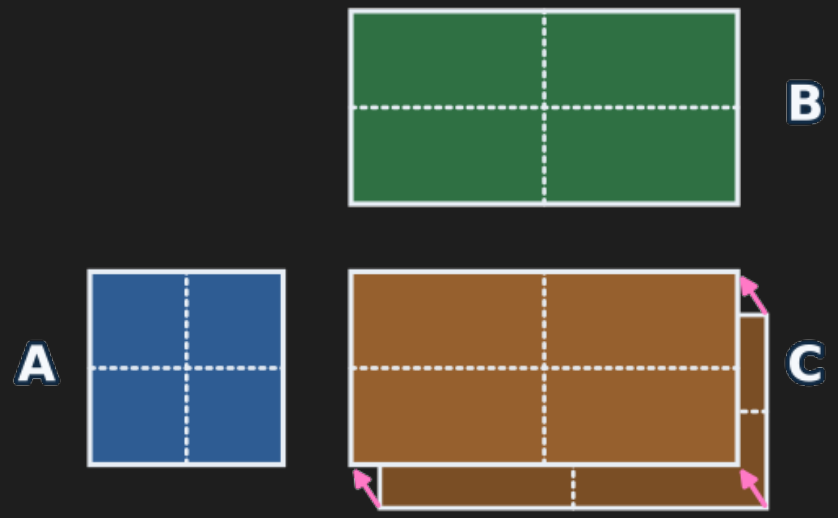
The PE grid in previous systems

SUMMA



User-specified 2D grid
e.g., $\langle p^{1/2}, p^{1/2} \rangle$

Johnson's 3D algorithm

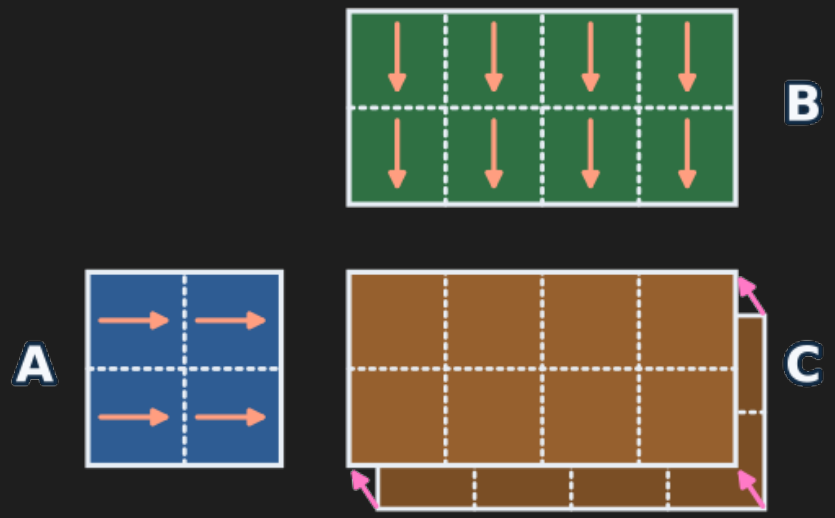


User-specified 3D grid
e.g., $\langle p^{1/3}, p^{1/3}, p^{1/3} \rangle$

The PE grid in previous systems (cont'd)

COSMA

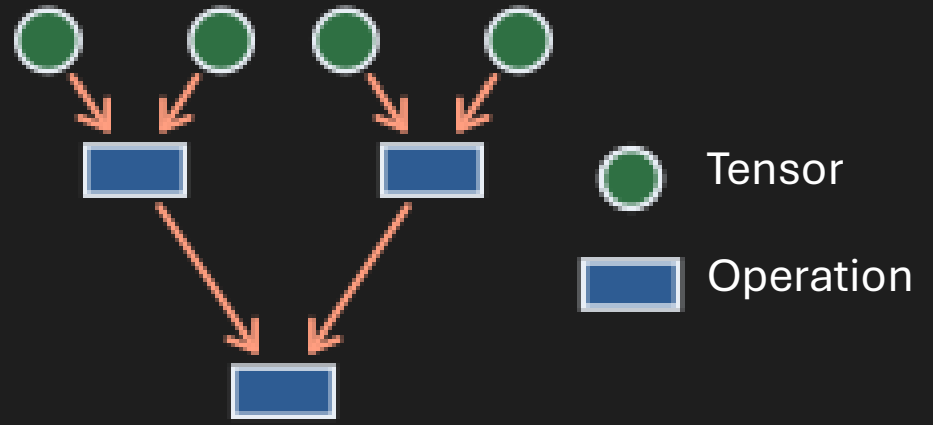
[Kwasniewski et al. 2019]



Optimal grid for the given GEMM size

DiMage

[Kong et al. 2023]



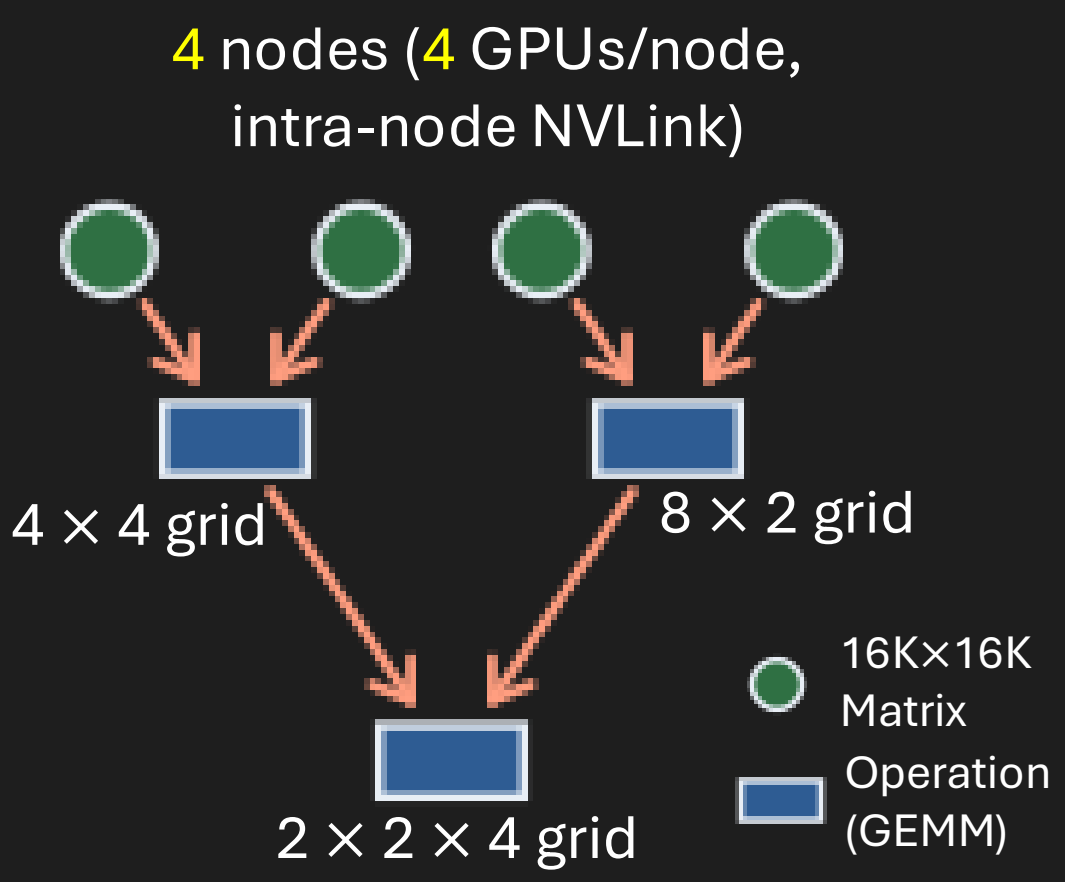
A single DAG-wide optimal grid (shared by all ops)

Grid flexibility

Tailored per-operator grids unlock better parallel strategies in DiMage-HG.

An example from the paper (right):

- 18.6% lower communication cost (vs. the best 2D DAG-wide)
- 42.1% lower communication cost (vs. the best 3D DAG-wide)



Contributions

- **DiMage-HG modeling abstraction**
  → Grid flexibility and fast solving
- **A rewrite of the non-linear model using Google CP-SAT**
  → Fast solving
- **A novel index-assembling scheme**
  → Coherent tensor distribution
- **A robust experimental evaluation:** $> 50\times$ solver-time speedup, $4.01\times$ average over CTF (CPU), and $1.73\times$ average over cuTensorNet (GPU).

Expanded search space



1D partition



1D partition
with replicated



2D partition (L)
and
Fully replicated (R)

Unified search space in
DiMage-HG
with per-op grid flexibility

Models

The pipeline of DiMage-HG

- Preprocessing
  - Factorizing #Nodes, #GPUs to build hypergrid
- Model code generation
  - Converting formulas to call CP-SAT
- Google CP-SAT
  - Find the optimal mappings $(\{\pi_t\}, \{\mu_c\})$
- Code Generator
  - Generate the final C++ code

Hypergrid

A constant vector of the prime factors (with multiplicity) of a fixed number of PEs:

- Hypergrid: $G^{all} = \langle p_0, \ldots, p_{L-1} \rangle$

where $nPE = \prod_{x=0}^{L-1} p_x$

Note: second-layer (inter/intra-node) hypercube omitted for simplicity.

Index the subspaces of an L-D hypercube with length-L bit-vectors.

Hypergrid Example

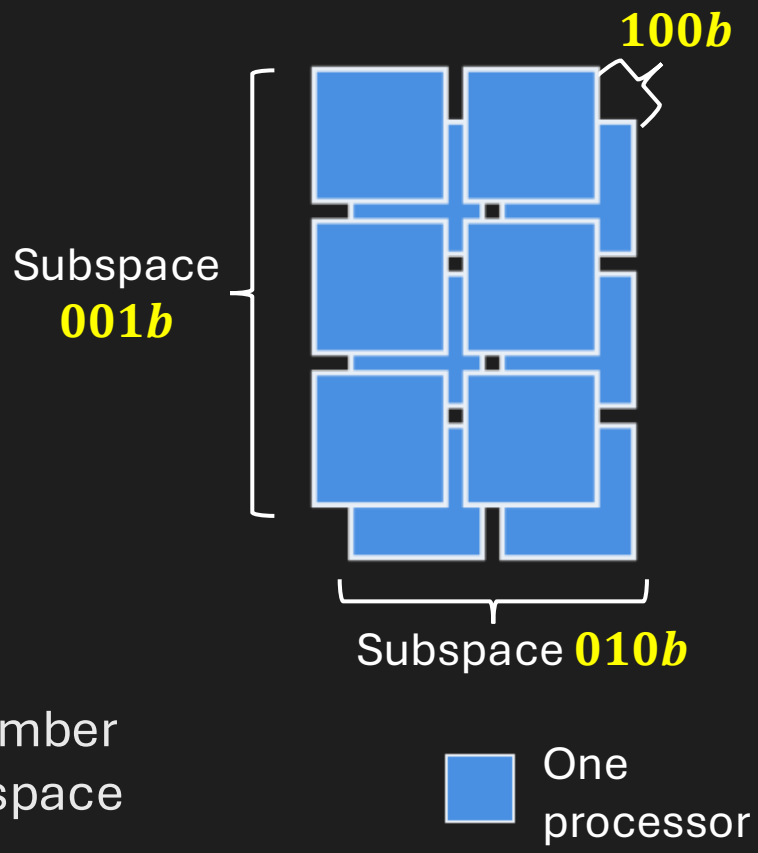
Suppose we have 12 processors:

- Hypergrid: $G^{all} = \langle 3, 2, 2 \rangle$

This hypergrid would have 2^3 subspaces, such as:

- $S^{all}[001b] = 3$
- $S^{all}[011b] = 2{\times}3 = 6$
- $S^{all}[110b] = 2{\times}2 = 4$

Note: S^{all} is the number of PEs in each subspace



Mapping

DiMage-HG maps tensor dimensions (π_t) and loops (μ_c) to hypercube subspaces to minimize C^{DAG}.

$$\{\pi_t^*\}, \{\mu_c^*\} = \underset{\{\pi_t\},\{\mu_c\}}{\operatorname{argmin}} C^{DAG}\left(\{\pi_t\}, \{\mu_c\}\right)$$

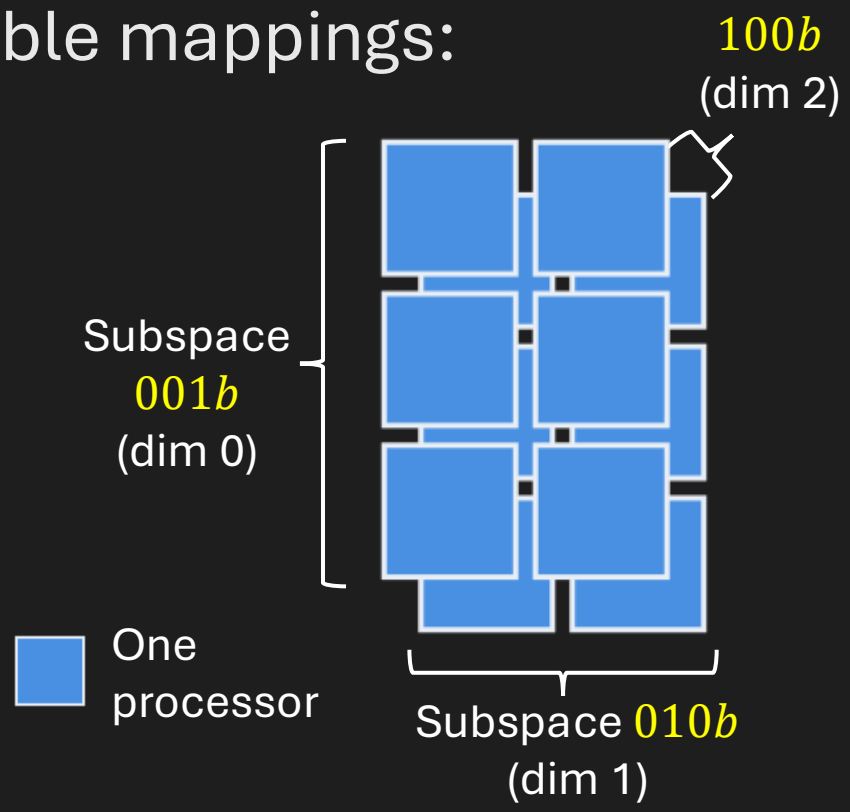
where $C^{DAG} = compute\ cost + communication\ cost$

- Once mapped, the data (or workload) is partitioned across $S^{all}[\pi_t]$ (or $S^{all}[\mu_c]$) PEs.

Mapping example 1

Hypercube $G^{all} = \langle 3, 2, 2 \rangle$ allows 2^3 possible mappings:

- Unmapped $(000b)$,
  - each PE executes all iterations
- Mapped to dimension 0 $(001b)$,
  - each PE executes 1/3 of the iterations
- Mapped to dimensions 1 and 2 $(110b)$,
  - each PE executes 1/4 of the iterations
- …



Mapping for tensors/operations

A GEMM on hypergrid $G^{all} = \langle 3, 2, 2 \rangle$

3 loops → 3 bit-vectors

- 1D Grid
  - $\langle 111b, 000b, 000b \rangle$, $\langle 000b, \underbrace{111b}_{\text{Dim-0,1, and 2}}, 000b \rangle$
- 2D Grid
  - $\langle \underbrace{110b, 001b,}_{\text{4×3 PE grid}} 000b \rangle$, $\langle \underbrace{101b, 000b, 010b}_{\text{6×2 PE grid}} \rangle$, $\langle \underbrace{100b, 000b, 010b}_{\text{2×2 PE grid (Replicated along Dim 0)}} \rangle$, ...
- ...

Note:

- Mappings within one op/tensor are pairwise disjoint.

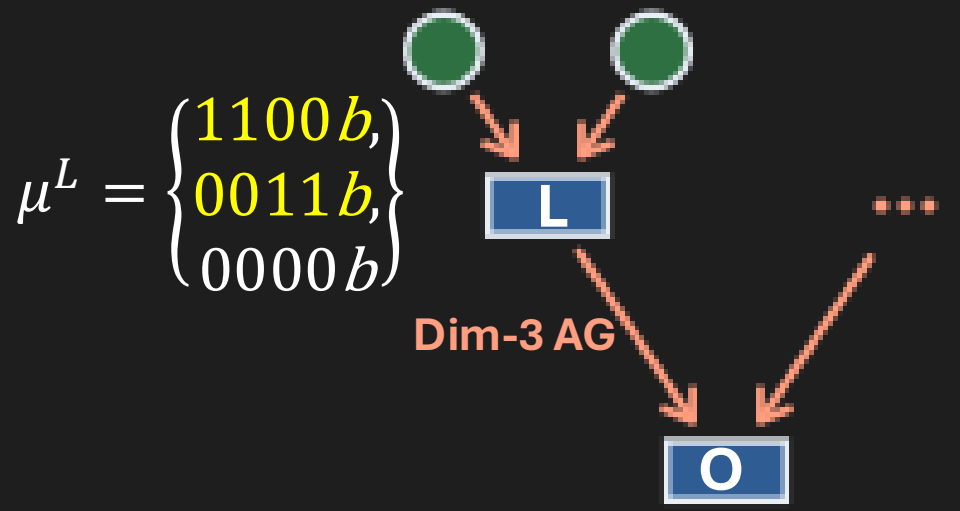
Communication Example

L produces matrix $tmp1$ as
$\pi^{tmp1} = \{1100b, 0011b\}$

Communication
along Dim-3 ($1000b$)

O reads matrix $tmp1$ as
$\pi^{tmp1*} = \{0100b, 0011b\}$

4 nodes (4 GPUs/node, intra-node NVLink)
$G^{all} = \langle 2,2,2,2 \rangle$



Code Generation: p-set/b-set

Our solution (p-set/b-set):

Pre-partition dimensions by PE count and align them with our hypergrid

Pre-partitioning:

1200-element dimension inside a ⟨3,2,2⟩ hypergrid → 12 segments



Local Data Example:

PE$[0,0,0]$'s local data with mapping $101b$ →



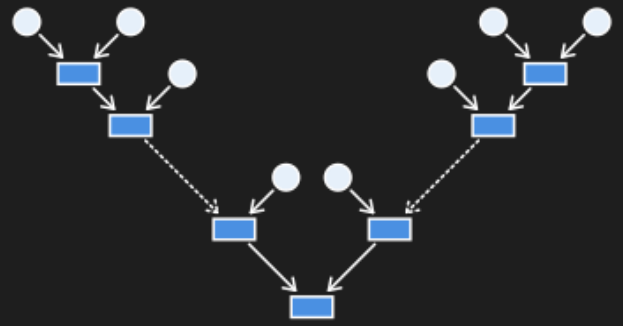
Experiments

Testbed

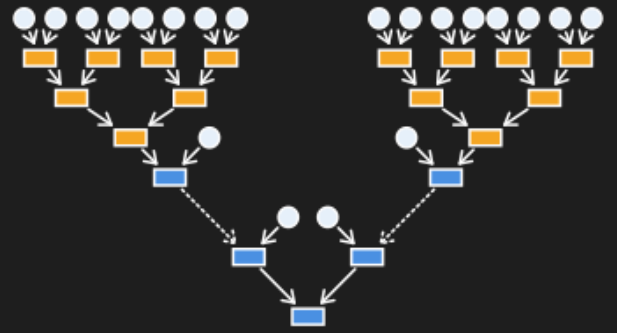
	Cluster-C (CPU Experiments)	Cluster-G (GPU Experiments)
Processor	Dual Intel Xeon 8268s, 48 cores @2.9 GHz	Dual AMD EPYC 7H12, 60 usable cores @2.6 GHz
Memory	192 GB	472 GB
GPU	-	NVIDIA A100 (2 per node)
VRAM	-	40 GB VRAM per GPU
Internode Links	100 Gbps InfiniBand	100 Gbps InfiniBand
Intranode Links	UPI	PCIe Gen 4 (31.5 GB/s)

Test cases

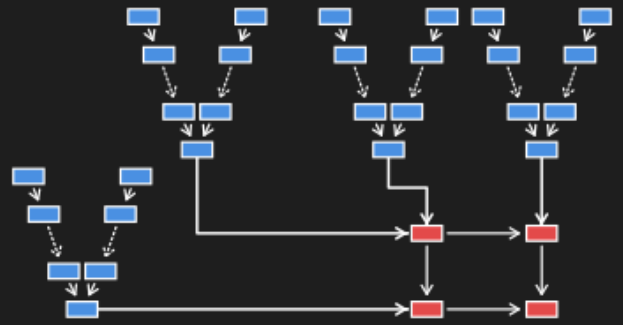
11 different DAGs, each with double-precision 16K × 16K square matrices, up to 128 GB input footprint.



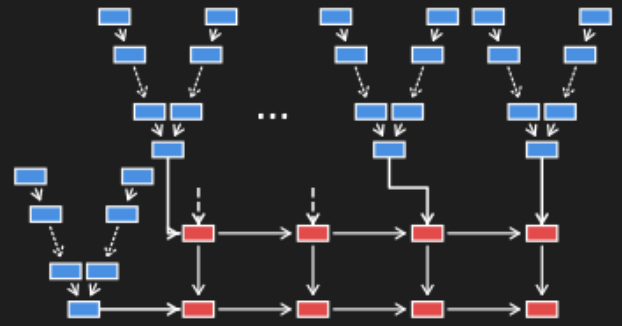
V-shape
3–63 ops



W-shape
63 ops



H2x2V15
64 ops



H2x4V9
62 ops

Experiment: Overall speedup

	CPU			GPU			
#PE	CTF	D2D*	D3D*	CTF	cuTN	D2D*	D3D*
4	1.54	1.09	–	15.08	1.24	1.27	–
8	2.65	1.12	1.18	25.58	1.43	1.52	1.70
16	10.59	1.40	1.48	24.67	1.72	1.76	2.05
32	6.50	1.56	2.34	39.49	2.33	2.42	2.60
64	3.73	1.24	2.52	27.36	2.19	2.77	2.68

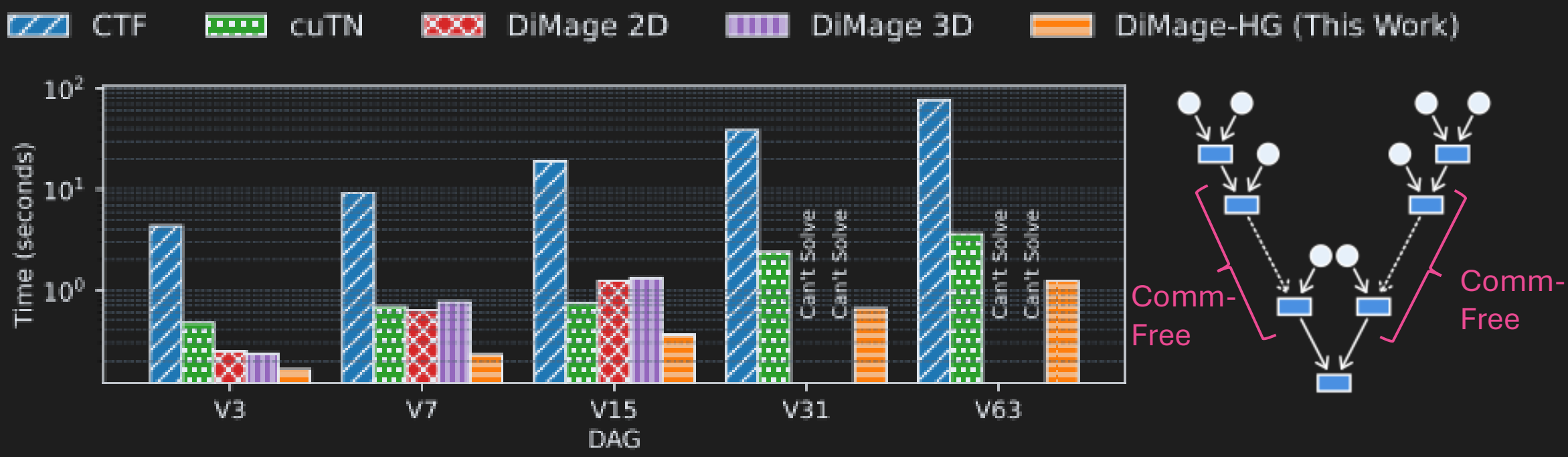
Average Speedup vs. Baseline Systems (higher is better)

- CTF [Solomonik et al. 2013]
- cuTN (cuTensorNet) [Bayraktar et al. 2023]
- D2D (DiMage-GPU 2D grid)
- D3D (DiMage-GPU 3D grid)

] [Wu et al. 2025]

*: only includes V3–V15

Experiment: V-Shape



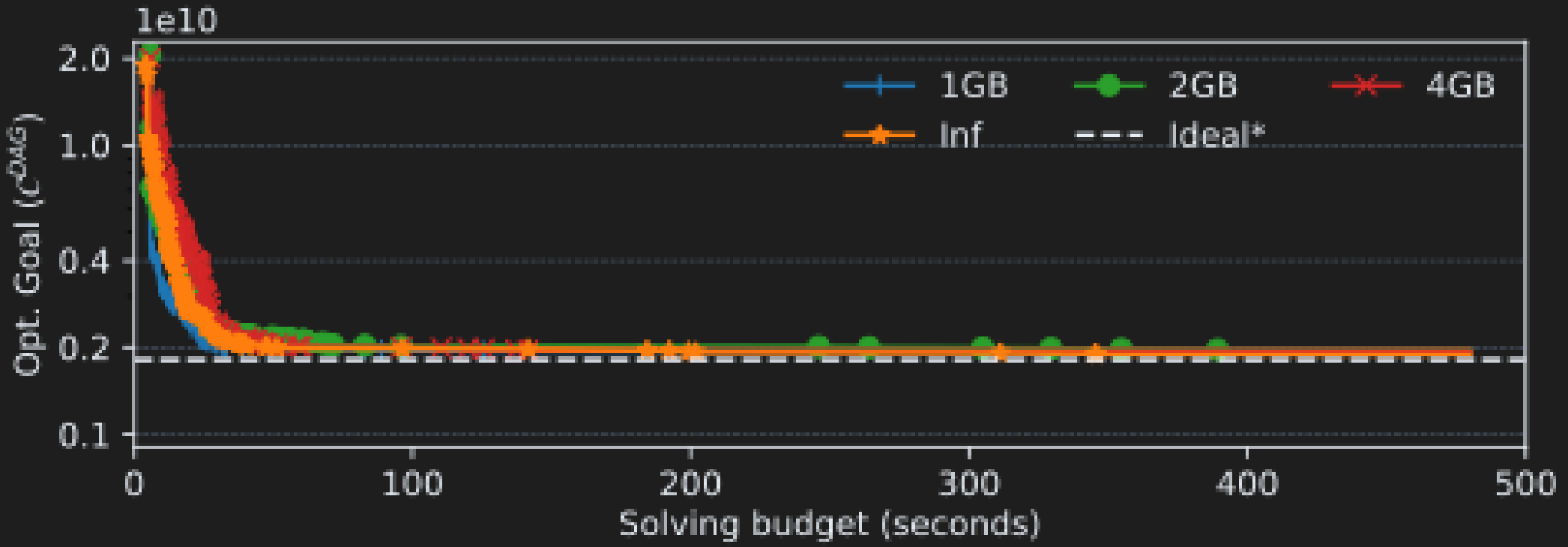
32 A100 GPUs $|DAG| = 3–63$ 8–128 GB input footprint

Experiment: Solving overhead

Mapper	Avg time (s)	Timeouts	Min timeout DAG (#ops)
DiMage-HG (CP-SAT)	33.87 ± 14.63	17/55	63
DiMage-HG (Z3)	308.4 ± 192.8	45/55	15
DiMage 2D (Z3)	3943.7 ± 1972.7	47/55	7
DiMage 3D (Z3)	412.4 ± 123.0	40/44	7

Note: CP-SAT solver used an 8-minute timeout with 8 threads.
Z3 solver used a 10-hour timeout with one thread.

Solving overhead part 2



W31 (perfect binary tree) targets 64 GPUs with different memory limits.
Loss $\leq$ 6.95% relative to the "ideal*"

Limitations

- Requires a factorizable number of PEs.
- Still sensitive to the evaluation order of tensor operators in the input DAG.
- No communication/compute overlap yet.

Conclusion

- **Key contribution:** DiMage-HG featuring an exponentially expanded search space and accelerated solving efficiency.

- **Performance gains:** 4.01× avg over CTF (CPU), 1.73× avg over cuTensorNet (GPU), and over 50× faster solving.

Thank you!