PipeThreader: Software-Defined Pipelining for Efficient DNN Execution

Yu Cheng[†], Lei Wang[†], Yining Shi[†], Yuqing Xia [◊], Lingxiao Ma [◊], Jilong Xue [◊], Yang Wang [◊],
Zhiwen Mo [‡◊], Feiyang Chen [¶◊], Fan Yang [◊], Mao Yang [◊], Zhi Yang[†]

† PEKING UNIVERSITY

◊ Microsoft

‡ Imperial College London

¶ SHANGHAI JIAO TONG UNIVERSITY

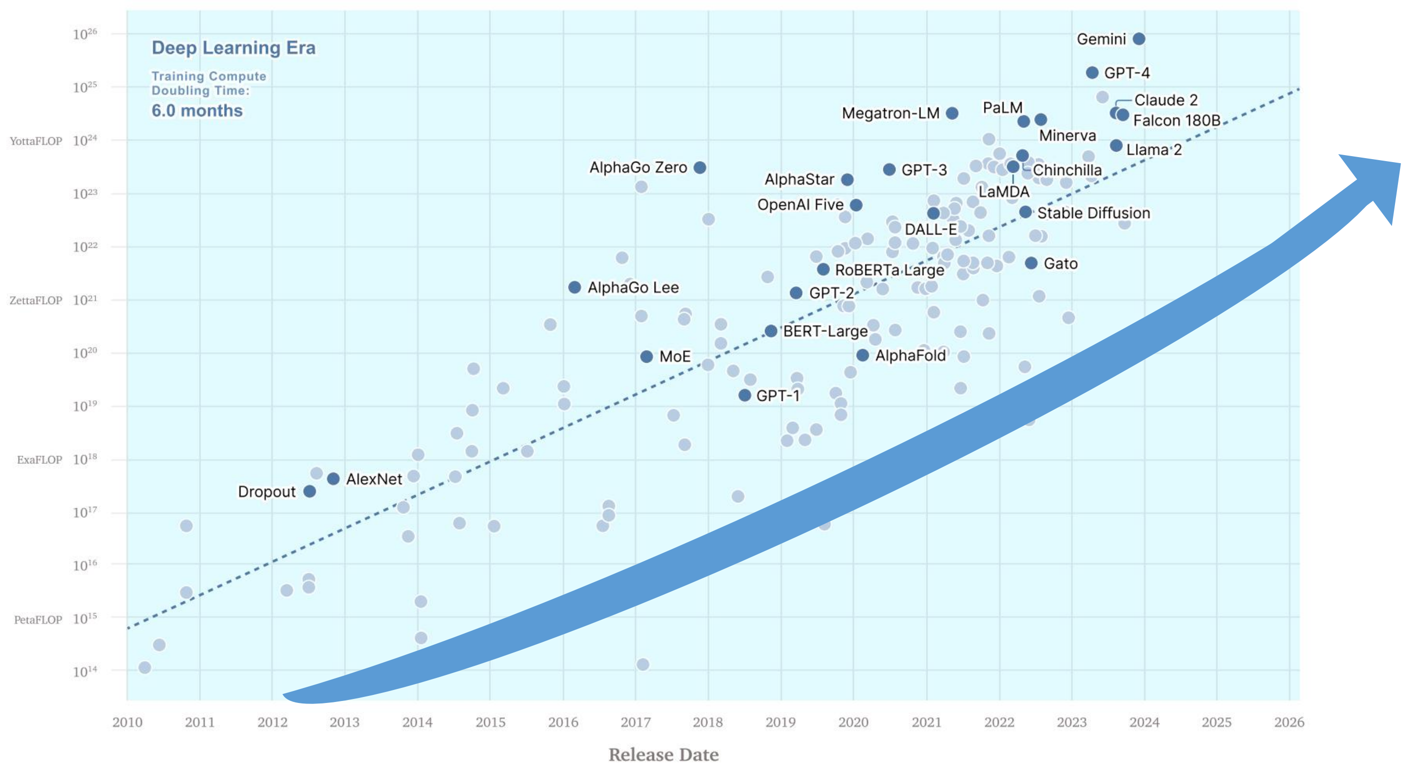
TileLang
TILE LANGUAGE

github.com/tile-ai/tilelang

Explosive compute demands in the LLM era

- Scaling Law drives ever-larger models

Compute Used for AI Training Runs (Deep Learning Era)

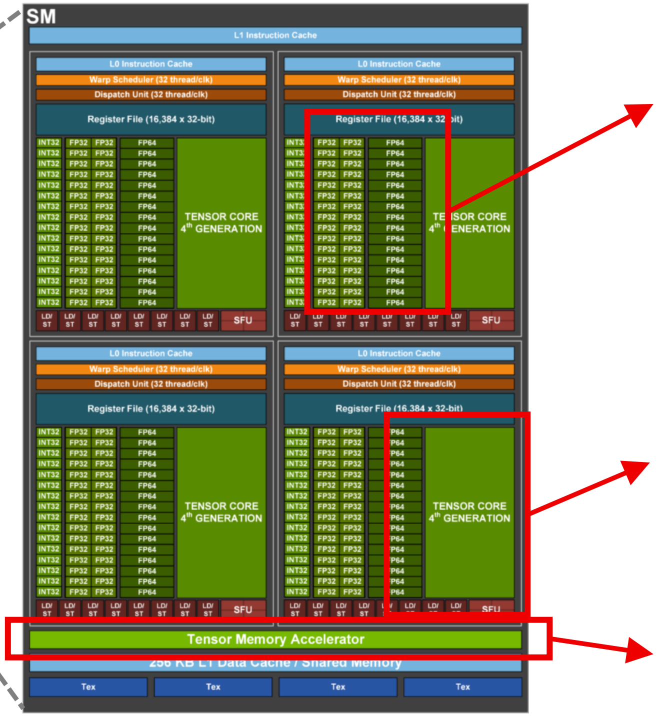


Sastry, Girish, et al. "Computing power and the governance of artificial intelligence." *arXiv preprint arXiv:2402.08797* (2024).

Hardware evolves to power the LLM era

- To meet the growing compute demands, hardware vendors have introduced specialized heterogeneous hardware units

NVIDIA H100 GPU


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE 4th GENERATION
LD/ST
SFU
Tensor Memory Accelerator
256 KB L1 Data Cache / Shared Memory
Tex

One SM in a NVIDIA H100 GPU

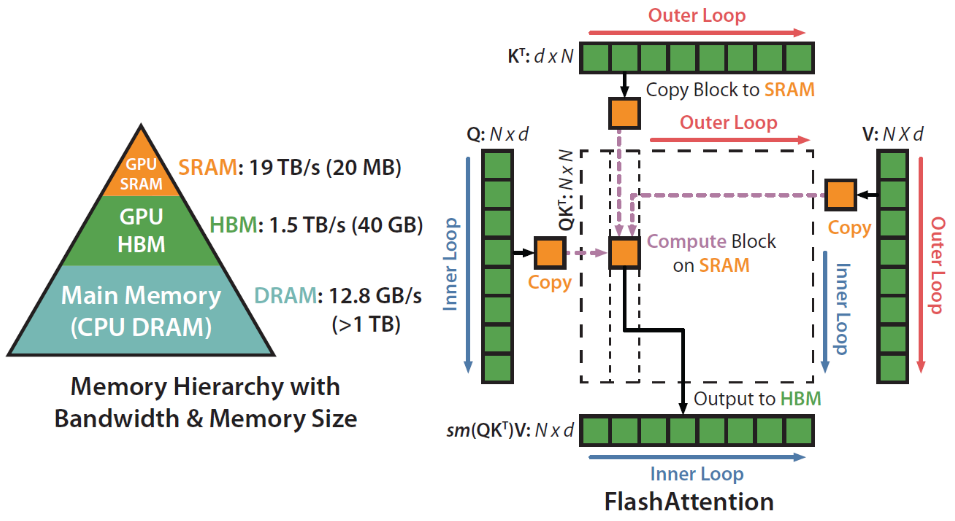
CUDA core:
general-purpose computation

TensorCore:
matrix multiplication

TMA:
memory load & store

Hand-crafted kernels for specialized hardware

- Specialized hardware requires sophisticated **hand-crafted** kernels for extreme performance
  - E.g., *FlashAttention*



Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness."

Challenges of manual kernel optimization

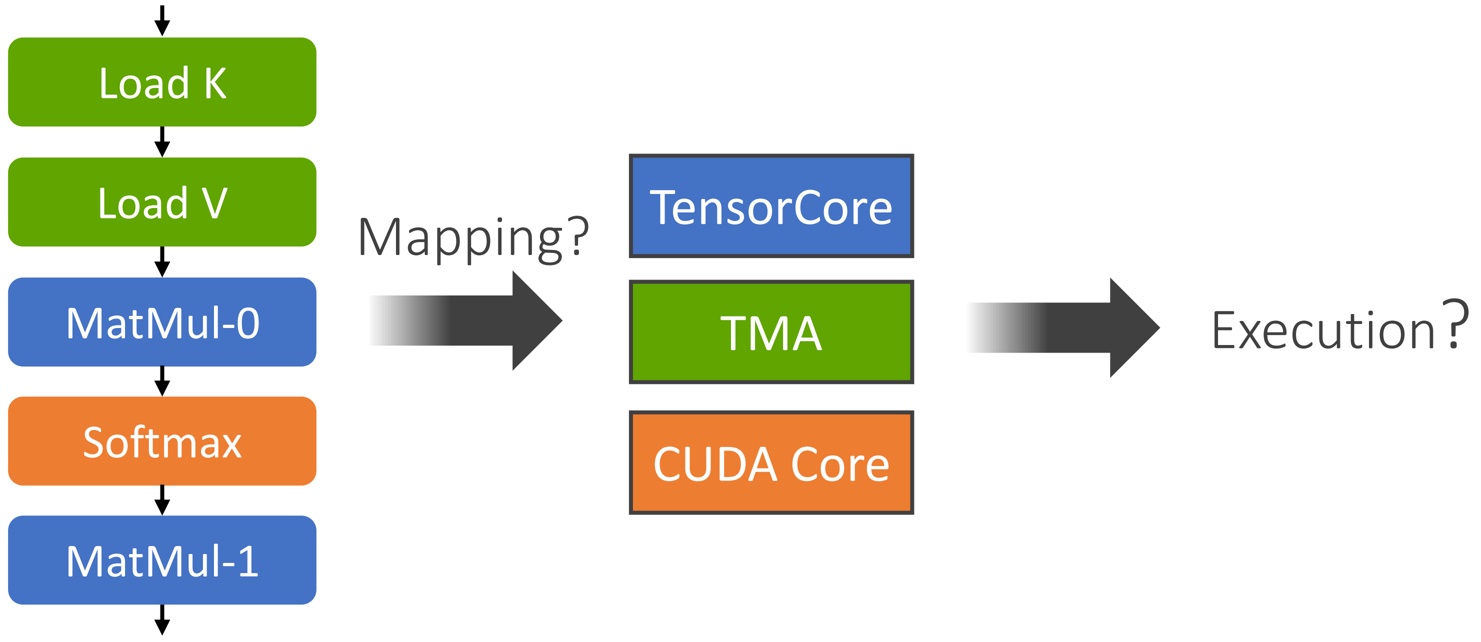
- Different model configurations require reimplementation
    - E.g., head dimension = 64, 128, 256, …

- Emerging models demand new compute patterns
    - E.g., Linear attention, mixed-precision GEMM

- Optimizations are often vendor-specific and don't transfer well, especially on less-studied hardware
    - E.g., AMD GPUs

Call for automated DNN compilation

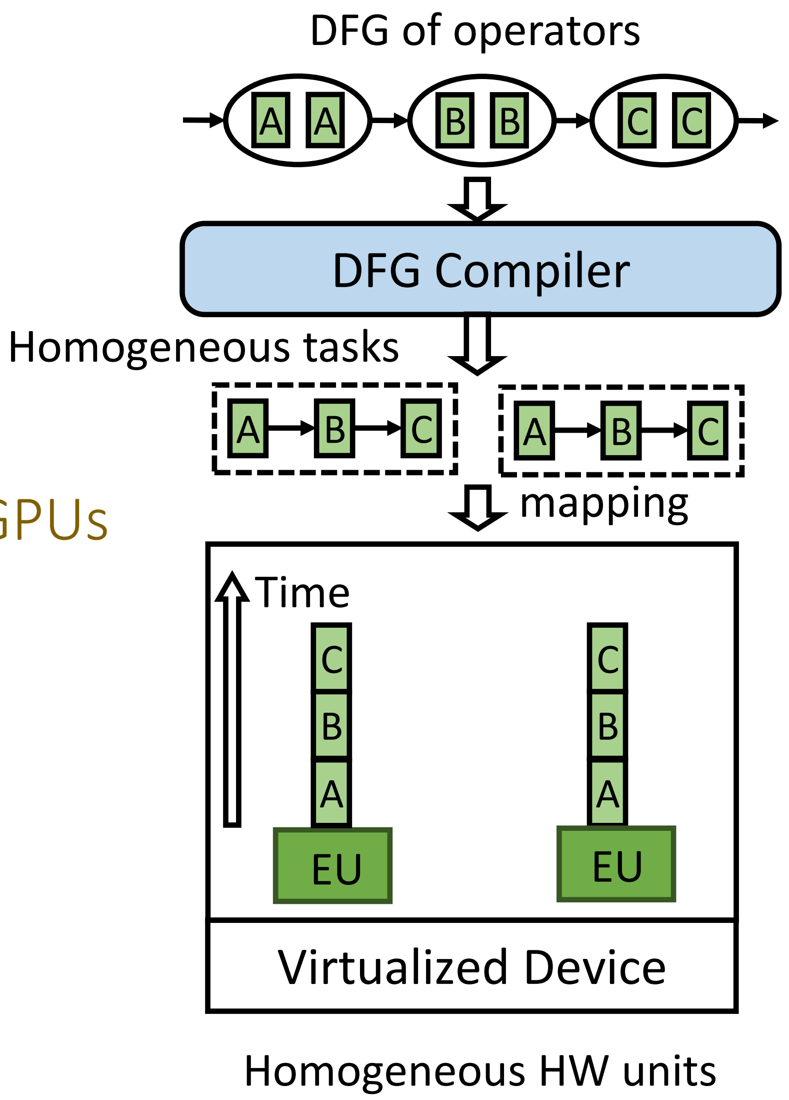
- Observation
  - The complexity primarily lies in pipeline scheduling
  - That is, mapping computation tasks to specialized hardware units and schedule these tasks



A pipelined view of *FlashAttention*

DNN compiler: no control of pipelining

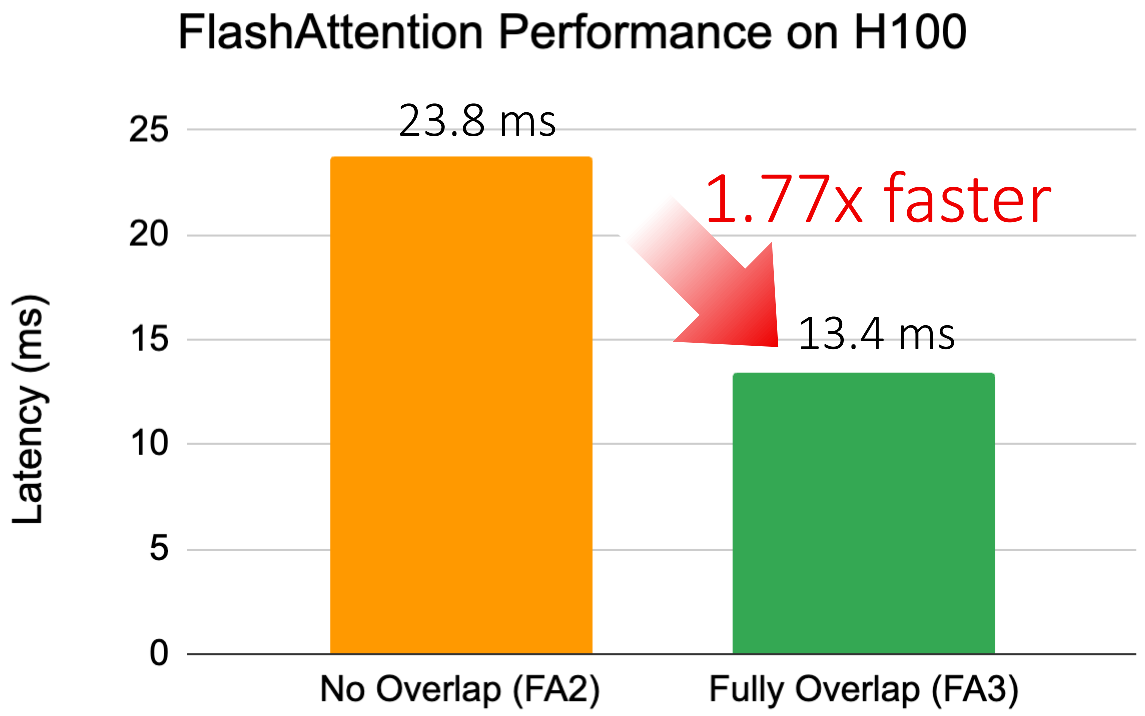
- Homogenous computation abstraction
  - Tasks are treated as the same, executed in a data-parallel manner
- Homogenous execution unit abstraction
  - E.g., Streaming Multiprocessors (SMs) on NVIDIA GPUs
- Leaves pipeline scheduling to hardware
  - i.e., warp switching by warp scheduler in SM
  - Often suboptimal, may introduce pipeline bubbles



Lack of pipeline control leads to hardware underutilization

- FlashAttention-2 only achieves 40% TensorCore utilization on H100

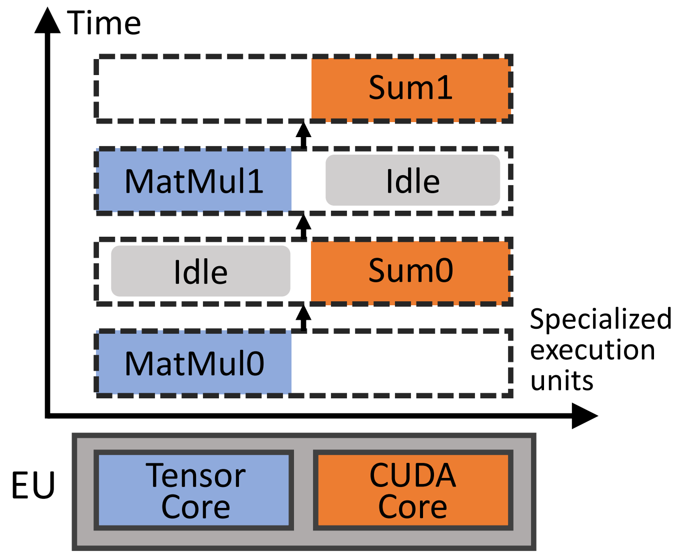
  → FlashAttention-3 emerges ~1 year after H100 enters the market



Example: MatMul + SUM

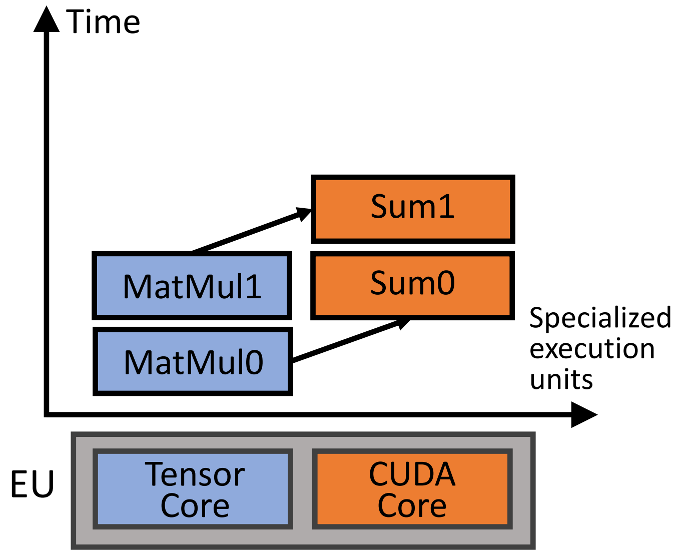
C[M, N] = A[M, K] x B[K, N]   // matrix multiplication
S[M] = sum(C[M, N], dim=-1)   // sum over the "N" dimension

Existing approach



No overlap between CUDA core and Tensor core

Desired approaches



Pipelined execution

Opportunities and Approach

- **Opportunities:**
  1. New hardware processes data at the **tile level**, i.e., subtensor
  2. DNN computation – analyzable at the tile level

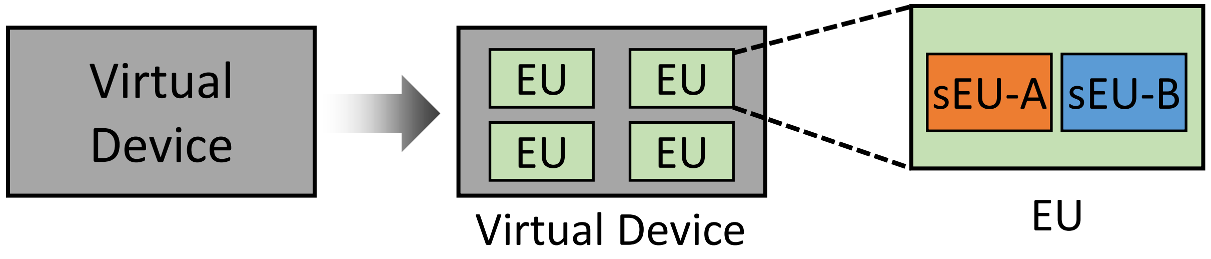
- **Approach**: software-controlled tile-level pipeline scheduling
  1. Hardware abstraction
  2. DNN workload abstraction
  3. Pipeline aware space construction and an efficient two-level schedule search policy

Specialization aware hardware abstraction

- Solution: two-layer abstraction
  - EU (*execution unit)*
    - Homogeneous hardware units
    - Execute in data-parallel manner
  - sEU (*specialized execution unit*)
    - Heterogeneous hardware units
    - Can execute in pipeline-parallel manner
    - E.g., TensorCores, TMA, CUDA cores, …

```
class vDevice {list<EU> EUs;};
class EU {list<sEU> sEUs;};
```

```
class sEU {
    bool is_async;
    void Execute(sTask s);
};
```



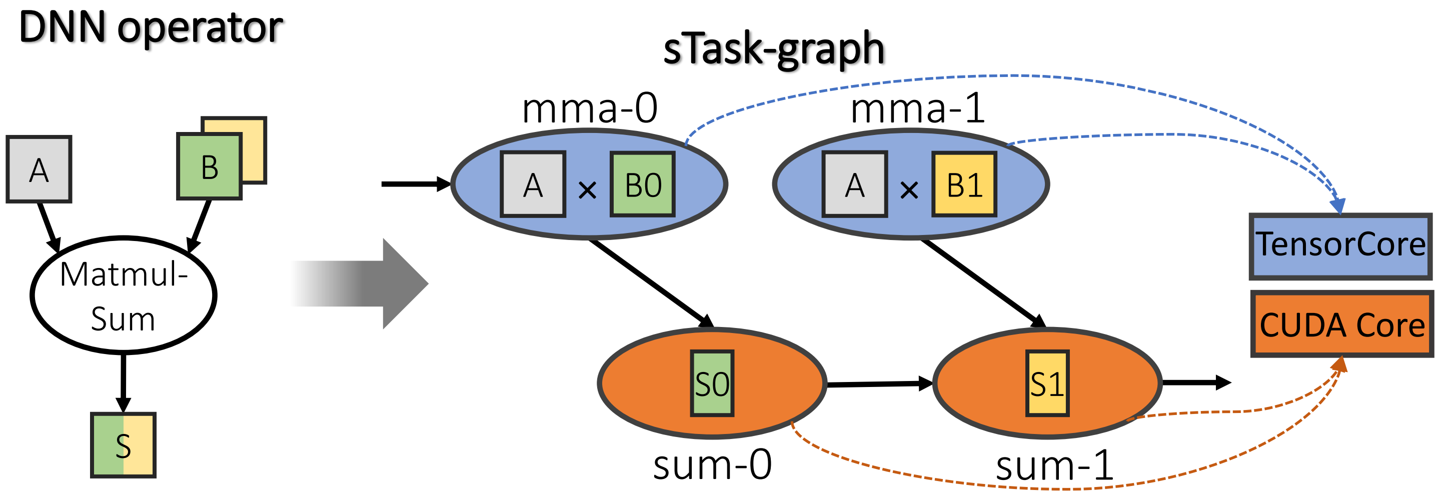
Specialized DNN computation abstraction

- sTask (*specialized tasks*)
  - E.g., load, mma, sum, …
- sTask-graph
  - nodes: sTasks in the computation task
  - edges: dependencies across sTasks

```
class sTask {
    size_t sTask_id;
    TensorExpr expr;
    TileShape shape;
    sEUType target_sEU;
};
```

```
class sTaskGraph {
    list<sTask> nodes;
    list<pair<sTask, sTask>> edges;
};
```

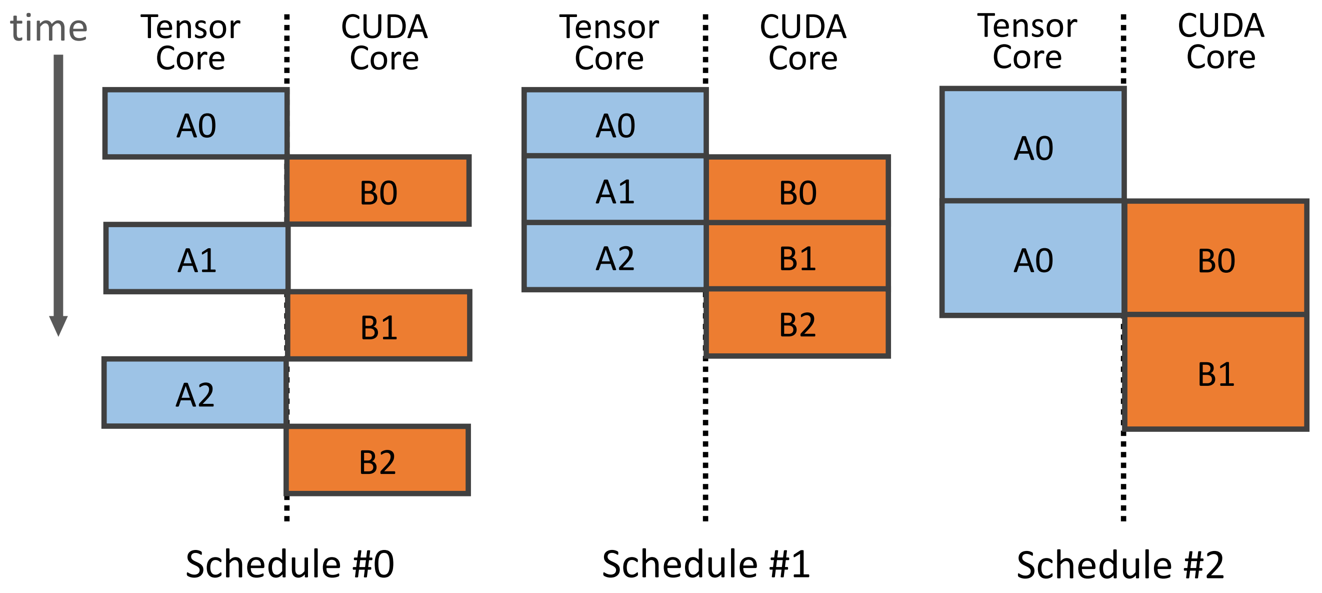
Example: MatMul+Sum



Execution plan construction and scheduling primitives

- Mapping sTask-graph to sEUs
  - Which sTask should execute on which EU/sEU?
  - When should each sTask be executed?

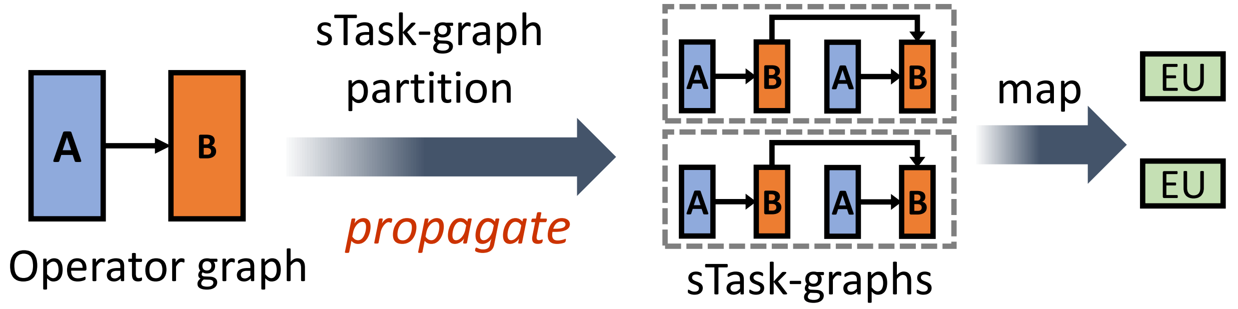
- Scheduling primitives
  - Append
  - Wait
  - Propagate



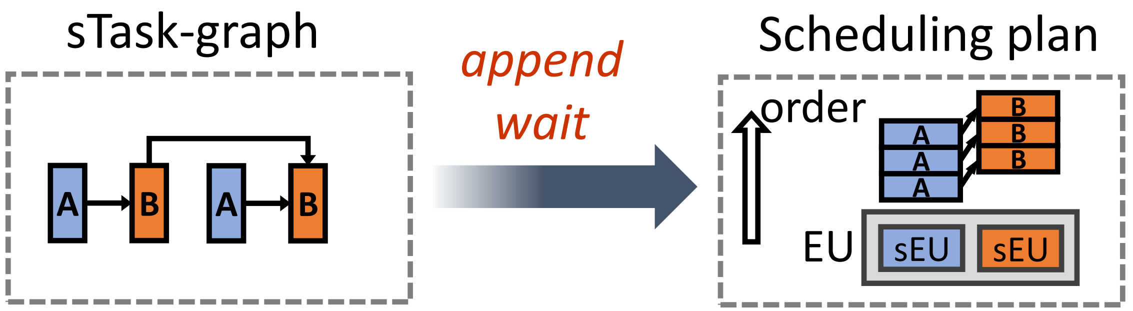
```
void Append(sTask s, <EU u, sEU v>);
void Wait(sTask_id s, list<sTask_id> t);
void Propagate(sTaskGraph g, TileShape shape);
```

Efficient search policy

- Two-level scheduling
  - Inter-EU scheduling
    - sTasks-graph partition across EUs



  - Intra-EU scheduling
    - How should the sTasks run across sEUs within an EU

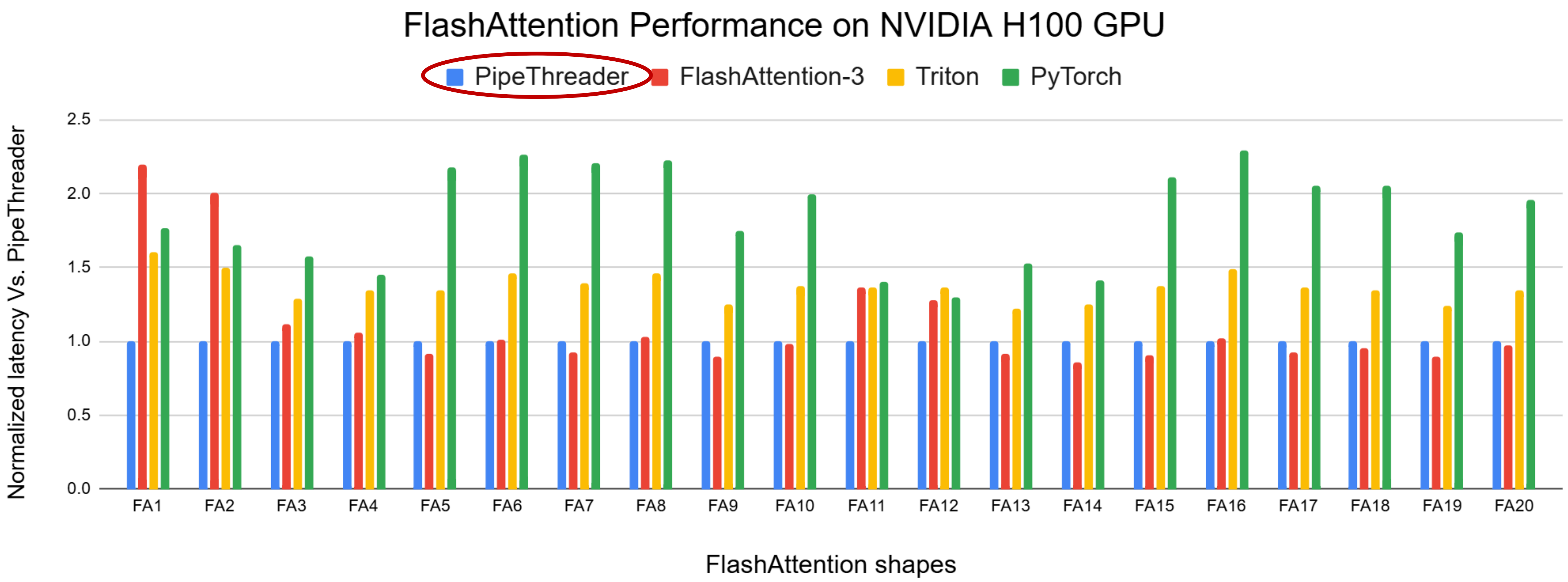


Example: FlashAttention

```
def FlashAttention(q_data, k_data, v_data):
    q = load(q_data)
    for i in T.Pipelined(loop_range):
        k = load(k_data[i]) # load_k_i
        acc_s = mma(q, k)    # mma_qk_i
        P = softmax(acc_s)   # softmax_i
        v = load(v_data[i]) # load_v_i
        acc_o = rescale(acc_o, param) # rescale_i
        acc_o += mma(P, v)   # mma_pv_i
```

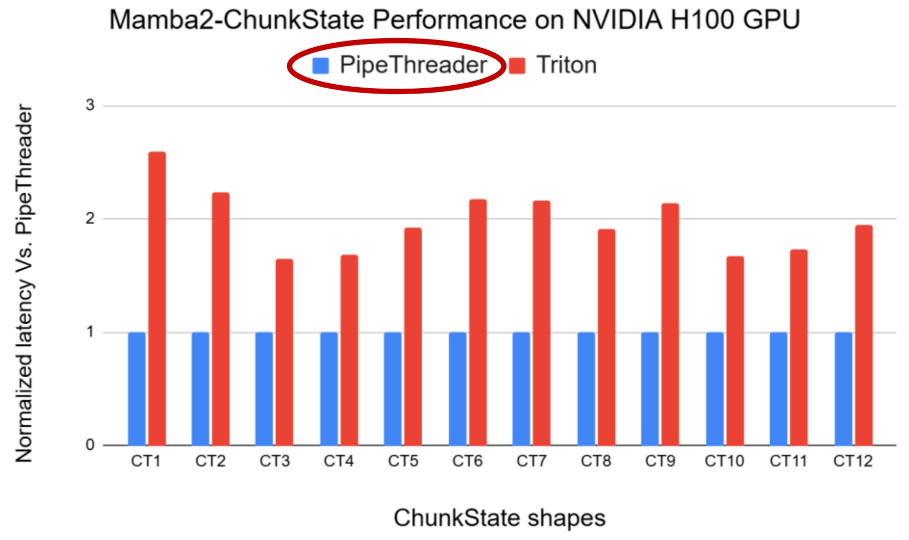
Adapting to diverse model configurations

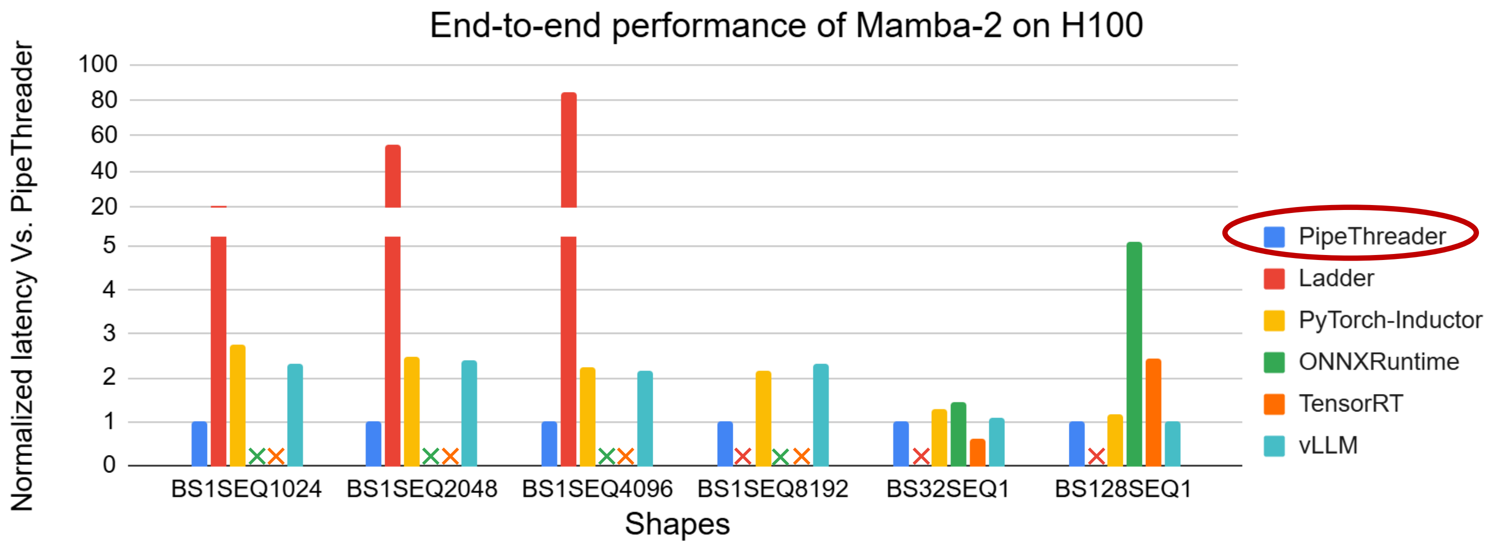
- Match or exceed the performance of FlashAttention-3 across a wide range of configurations
  - Up to 2.18x speedup vs. FlashAttention-3

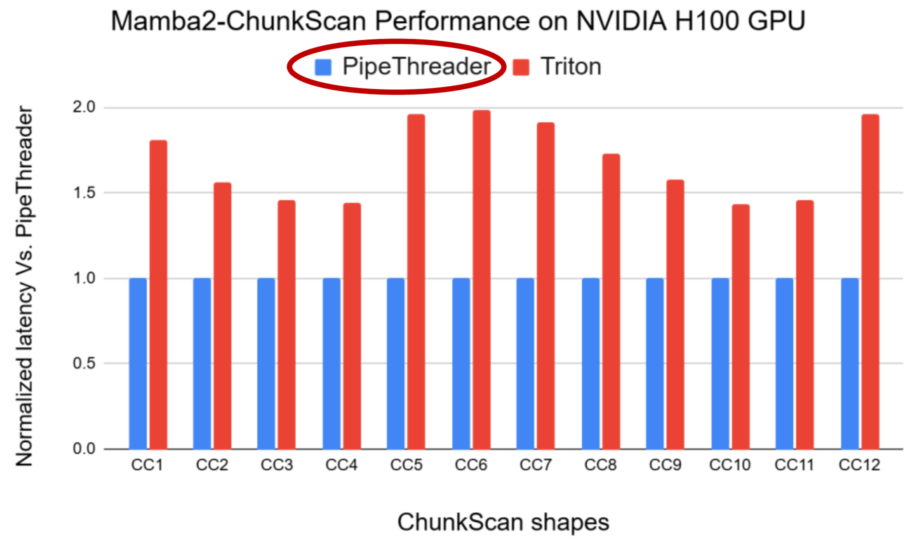


Optimization for emerging models

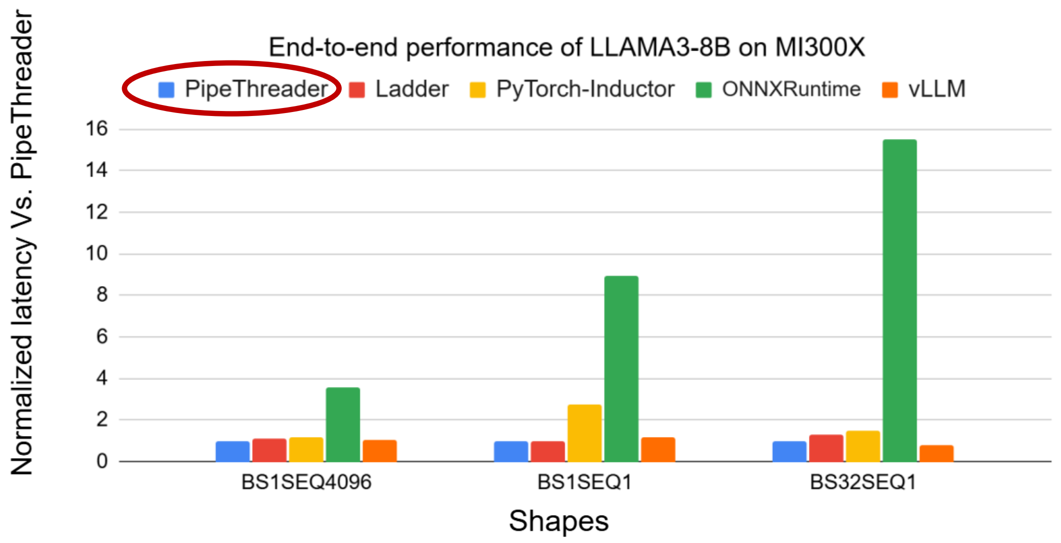
- Find better schedule on Mamba2
  - 1.71x and 1.98x avg. speedup over Triton on linear attention operations (*ChunkState* and *ChunkScan*)
  - 1.92x/45.93x avg. end-to-end speedup over PyTorch-Inductor/Ladder

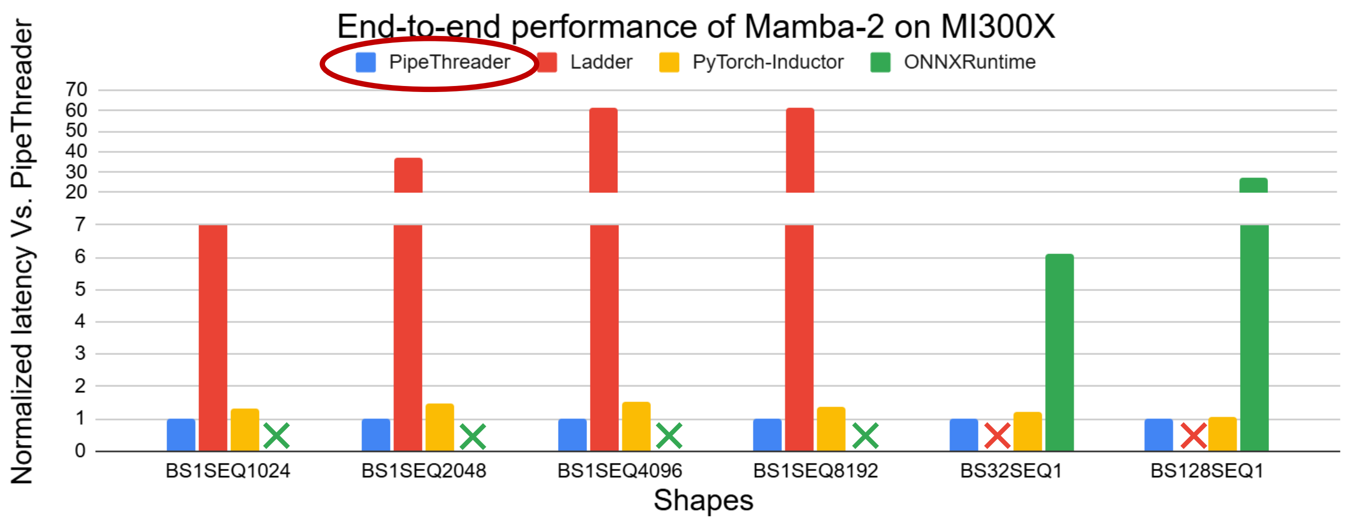






Extending to less-studied hardware (AMD MI300X)





- LLAMA3: 1.48x/1.07x avg. speedup over PyTorch-Inductor/Ladder
- Mamba2: 1.31x/32.93x avg. speedup over PyTorch-Inductor/Ladder

Data of LLAMA and Mamba2 are evaluated on a single layer (more details and evaluation in paper).

Conclusion

- Hardware schedulers are no longer sufficient for efficient pipeline execution
- PipeThreader proposes:
  - sEU: expose heterogeneous specialized execution units of modern AI accelerators
  - sTask and sTask-graph: expose fine-grained pipeline parallelism at tile level
  - Scheduling primitives: build efficient pipeline schedules
- PipeThreader has been integrated into ***TileLang***, a DSL for high-performance AI kernel development

TileLang
TILE LANGUAGE

TileLang GitHub Repo

https://github.com/tile-ai/tilelang

Thank you!