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

TileLang

TILE LANGUAGE

Bridge Programmability and Performance in Modern AI Workloads

Why TileLang?

系统上，我们沿着Tile抽象的研究路线持续进行着的探索...

Rammer	Roller	Welder	Ladder	PipeThreader
OSDI 2020	OSDI 2022	OSDI 2023	OSDI 2024	OSDI 2025

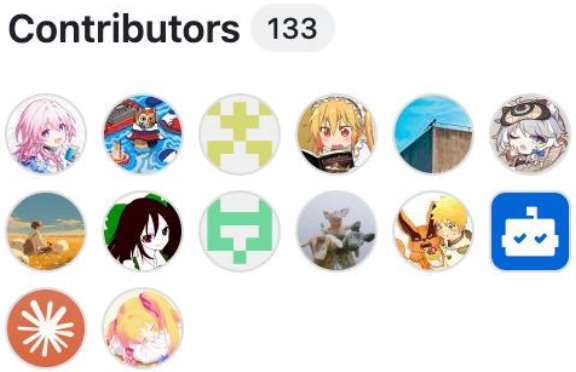
为什么要做TileLang?

更容易地写出高性能Kernel

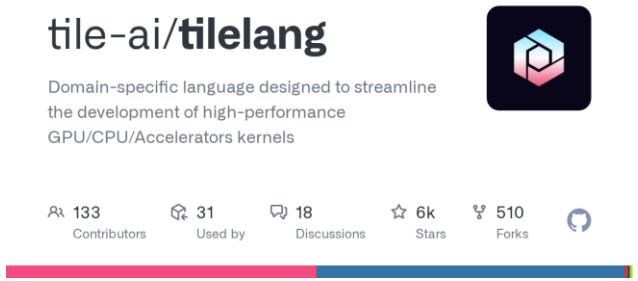
将思维中抽象的Tile具体化

TileLang 发展现状

133位开源代码贡献者



2025年2月开源，6K stars



DeepSeek V3.2 Exp 开源TileLang算子

在新模型的研究过程中，需要设计和实现很多新的 GPU 算子。我们使用高级语言 TileLang 进行快速原型开发，以支持更深入的探索。在最后阶段，以 TileLang 作为精度基线，逐步使用底层语言实现更高效的版本。因此，本次开源的主要算子包含 TileLang 与 CUDA 两种版本。**我们建议社区在进行研究性实验时，使用基于 TileLang 的版本以方便调试和快速迭代。**

华为昇腾硬件后端适配

社区其他开发者/贡献者

字节跳动	小米
阿里巴巴	Nvidia
AMD	高校研究人员

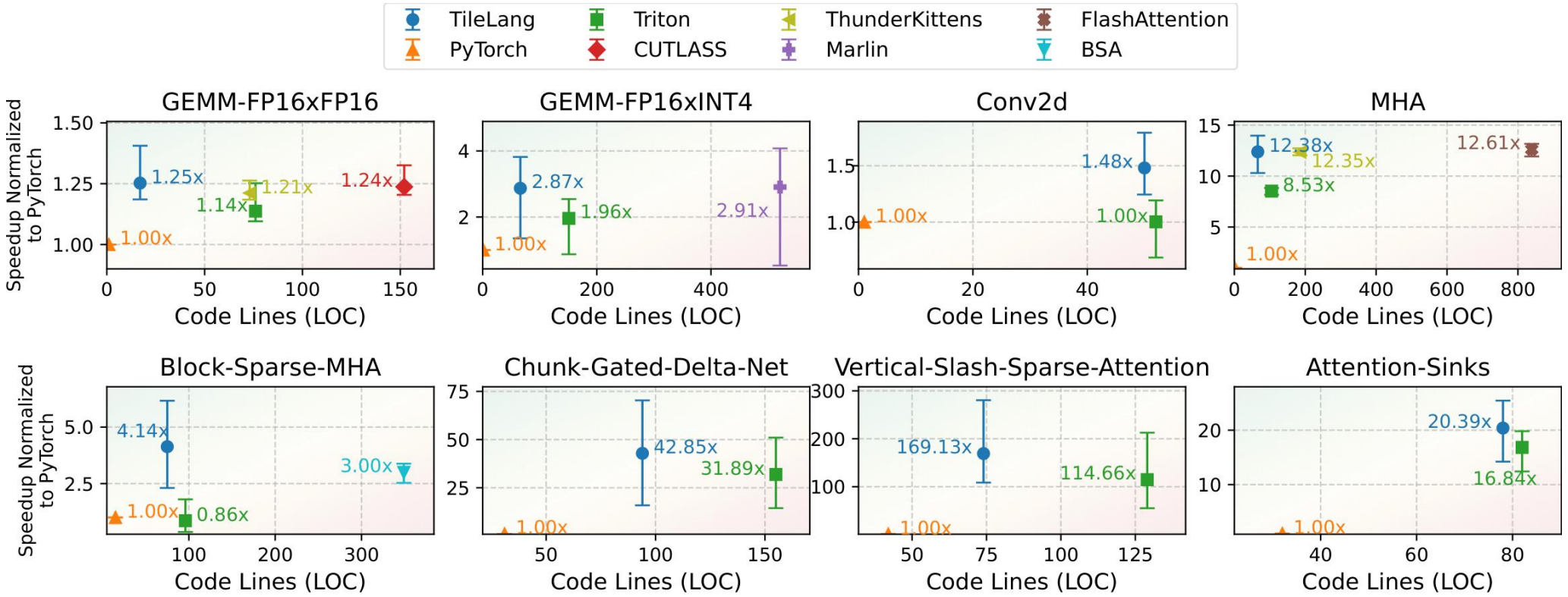
NPU DS-V3.2-Exp TileLang算子开发实践：

https://gitcode.com/cann/cann-recipes-infer/blob/master/docs/models/deepseek-v3.2-exp/deepseek_v3.2_exp_tilelang_operator_guide.md

TileLang-Ascend开源社区：

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

TileLang优势: 超级好的Kernel性能



https://github.com/deepseek-ai/TileKernels

TileLang编写DeepSeek FlashMLA

```
with T.Kernel(batch, heads // min(block_H, kv_group_num), threads=256) as (bx, by):
  QShared = T.alloc_shared([block_H, dim], dtype)
  SShared = T.alloc_shared([block_H, block_N], dtype)
  QPEShared = T.alloc_shared([block_H, pe_dim], dtype)
  acc_s = T.alloc_fragment([block_H, block_N], accum_dtype)
      ... skip initialization statements
  T.annotate_layout({O_shared: T.make_swizzle_layout()})
  for k in T.Pipelined(loop_range, num_stages=2):
    T.copy(KV[bx, k * block_N:k * block_N + block_N, cur_kv_head, :],
           KVShared)
    T.copy(K_pe[bx, k * block_N:k * block_N + block_N, cur_kv_head, :],
           KPE_shared)
    T.clear(acc_s)
    T.gemm(QShared, KVShared, acc_s, transpose_B=True)
    T.gemm(QPEShared, KPE_shared, acc_s, transpose_B=True)
    T.copy(scores_max, scores_max_prev)
    T.fill(scores_max, -T.infinity(accum_dtype))
    T.reduce_max(acc_s, scores_max, dim=1, clear=False)
    for i in T.Parallel(block_H):
      scores_scale[i] = T.exp2(scores_max_prev[i] * scale - scores_max[i] * scale)
    for i, j in T.Parallel(block_H, block_N):
      acc_s[i, j] = T.exp2(acc_s[i, j] * scale - scores_max[i] * scale)
    T.reduce_sum(acc_s, scores_sum, dim=1)
    T.copy(acc_s, SShared)
    for i in T.Parallel(block_H):
      logsum[i] = logsum[i] * scores_scale[i] + scores_sum[i]
    for i, j in T.Parallel(block_H, dim):
      acc_o[i, j] *= scores_scale[i]
    T.gemm(SShared, KVShared, acc_o)
  for i, j in T.Parallel(block_H, dim):
    acc_o[i, j] /= logsum[i]
  T.copy(acc_o, O_shared)
  T.copy(O_shared, Output[bx, by * BLOCK_H:by * BLOCK_H + BLOCK_H, :])
```

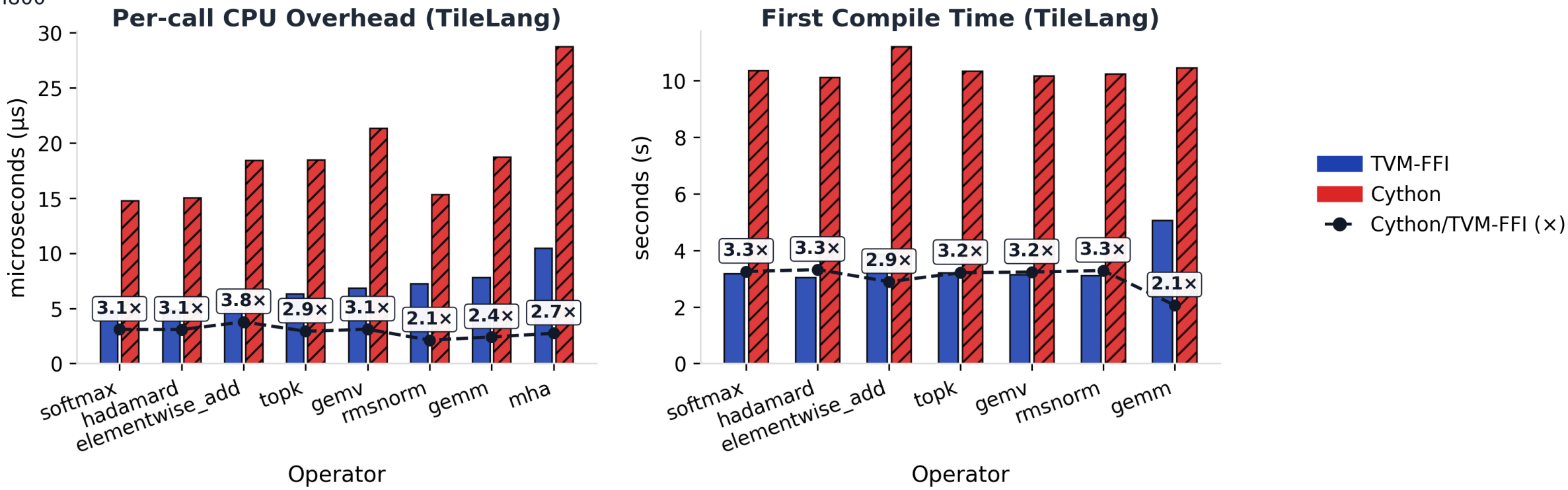
1. Decide Tile Configuration — Recommend
2. Define Buffer on desired memory Layer — Recommend
3. Define Desired Data Layout — Inference
4. Auto Pipeline Scheduling — Inference
5. Warp Partitioning — Recommend
6. Auto utilize High Performance instruction — Inference

(核心 500+ cute代码) tilelang 使用50行左右的kernel程序可以获得FlashMLA 95% 左右的性能。

TileLang优势: 超低的CPU Overhead

- 更高效的Python Binding: TVM-FFI
- Host Check代码生成, 将Tensor检查挪到C++侧

Device: H800



TileLang优势: 一个 wheel，通吃多 Python & 多 CUDA

- 高性能Kernel接口无需绑定Pybind，依赖固定torch版本（unlike triton.）
- Python Stabel ABI: Python3.8+ 以后都用一个Wheel
- CUDA Stubs: CUDA 11/12/13 都用一个Wheel

TileLang优势: 数值精度和一致性

- 默认关闭fast math（unlike triton.）
- 可以显式使用fast math计算: T.__exp
- 可以显式使用高精度计算: T.ieee_sqrt
- Annotate_layout实现Bit Identical

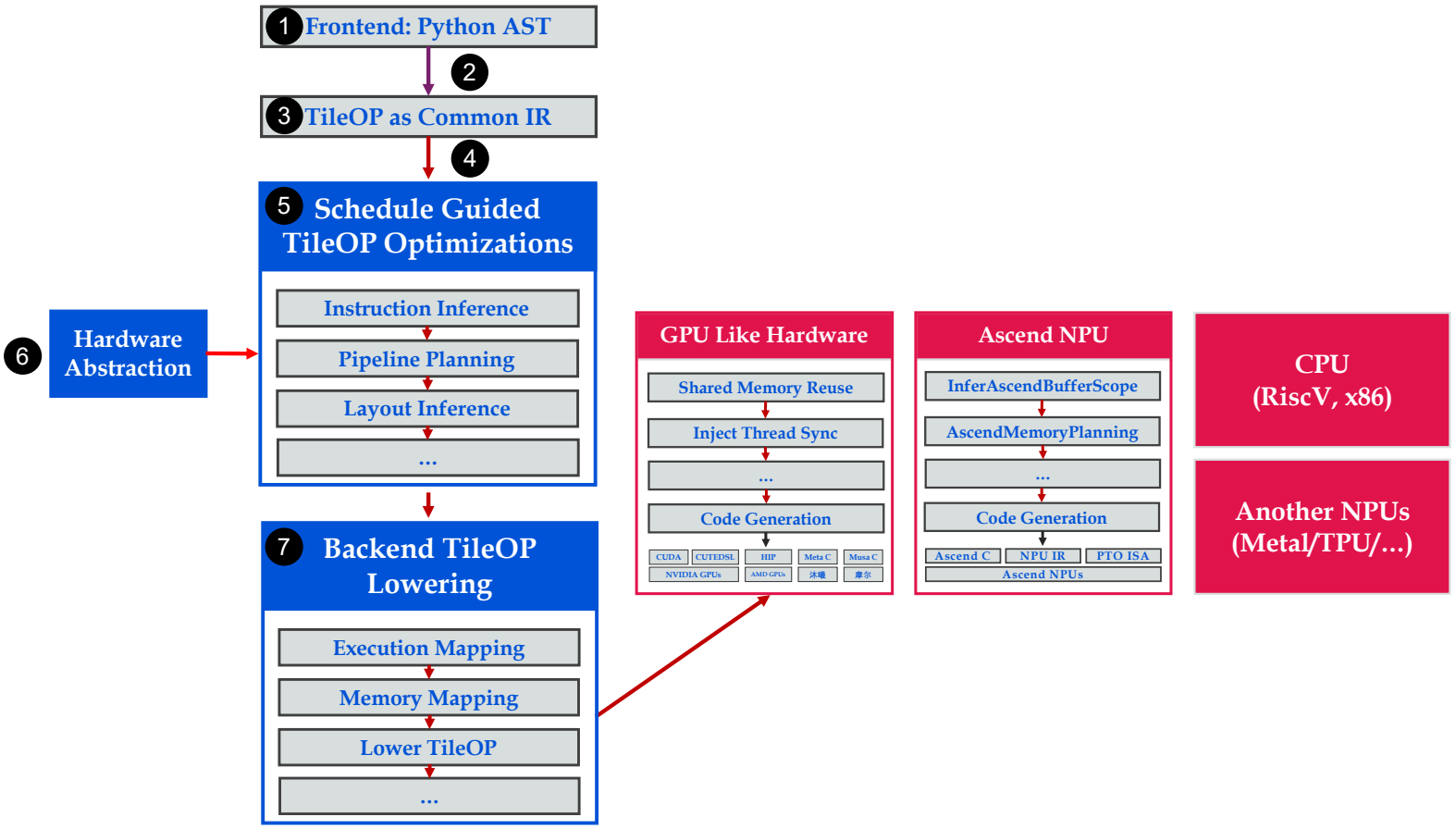
TileLang 最新硬件支持

- T.tcgen05(block scale)和tensor memory
- 2SM support/CLC -- GEMM 1700TFLOPS
- Blackwell 上的 FA 支持 (FA4 schedule WIP)
- MI300X 特性支持 (copy.async, wmma, codegen, …)

TileLang 系统更新

- Z3 SMT solver: 整数表达式规范分析，均衡计算开销和表达能力
- Layout系统: 高维GEMM layout, fully-replicated
- 层次化 reduction intrinsics
- 前端编译选项和 DumpIR

TileLang 硬件后端



NVIDIA

AMD

Ascend

METAX 沐曦

摩尔线程 MOORE THREADS

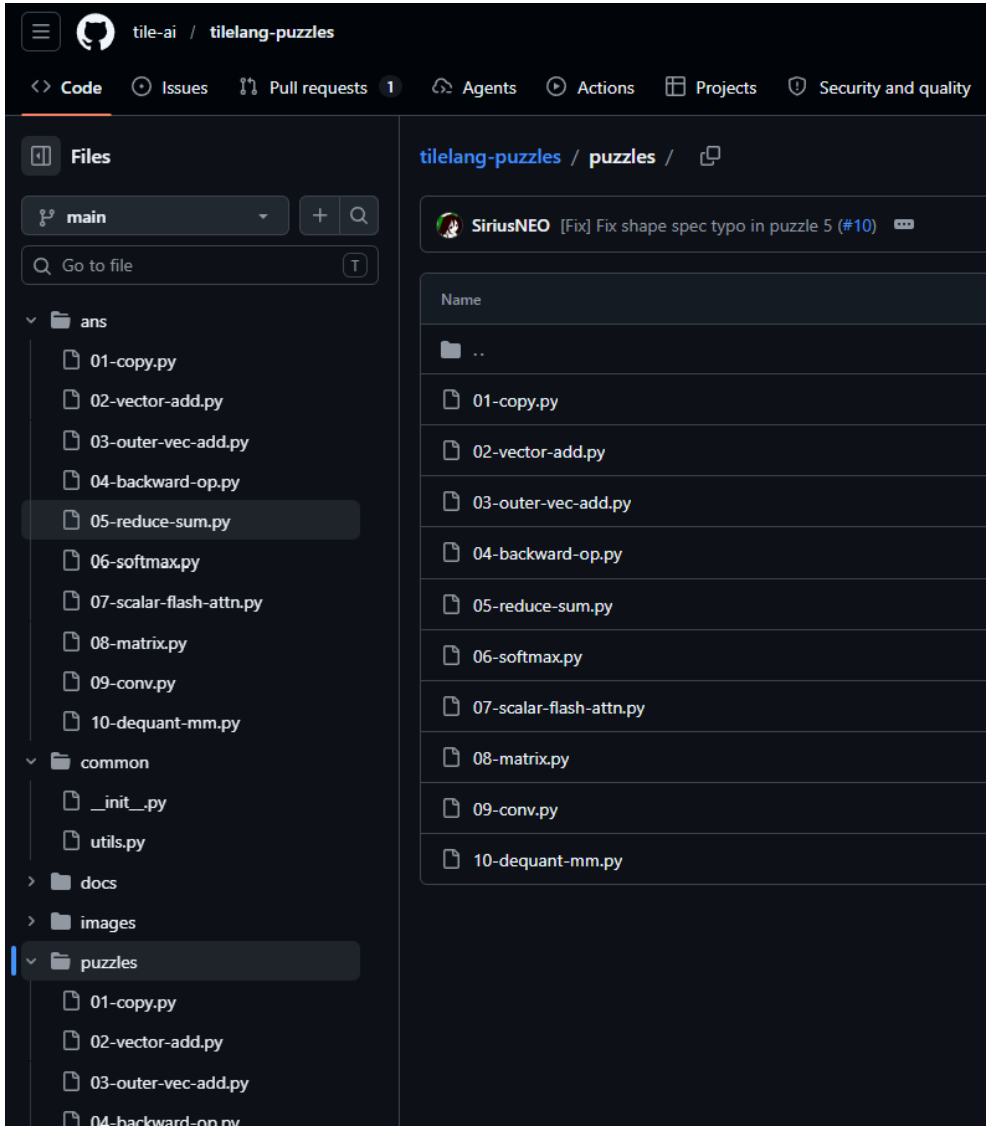
...

TileLang Puzzles: From beginner to experts

1. 为拥抱更广阔的开源社区，促进社区共建，Tile-AI 公开了 TileLang DSL 的学习指南 TileLang puzzles；通过由易到难的算子书写，由浅入深地了解 TileLang 的设计特性和高性能调优方法，为 TileLang 适配更广泛的硬件特性奠定基础

2. 目前的公开 puzzles 有 10 个，从简单的 copy 到复杂的 dequant gemm，覆盖 TileLang 的主流编程范式和原语，同时配备答案和测试脚本，易于上手

3. 沐曦 MetaX 社区针对这些 puzzles 设计和封装了适配 MetaX 的版本，方便同学们学习和练习 https://www.gitlink.org.cn/ccf-ai-infra/tilelang-puzzles

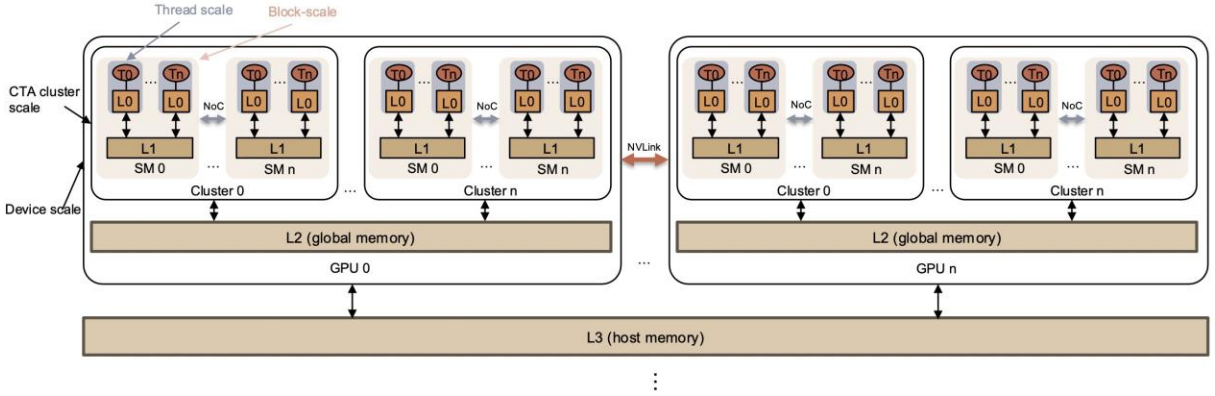


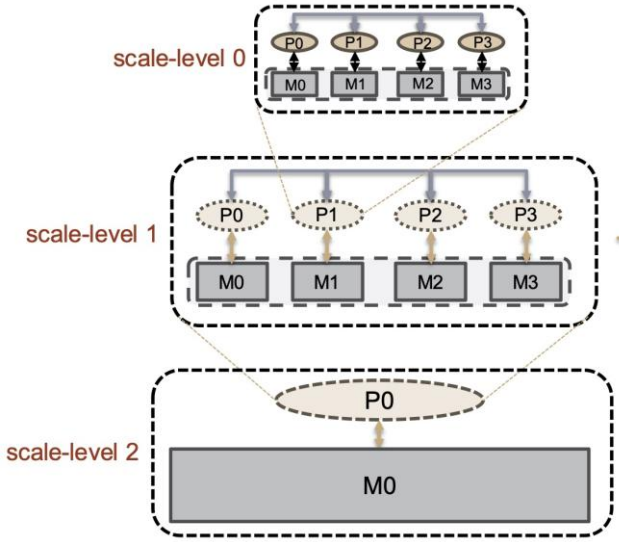
TileScale: 统一的分布式算子编程

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

大模型对可扩展性的需求，催生分布式芯片与多设备/多节点的热度
尤其是细粒度(tile粒度)的通算融合.

Hierarchical Distributed Architecture (HDA)

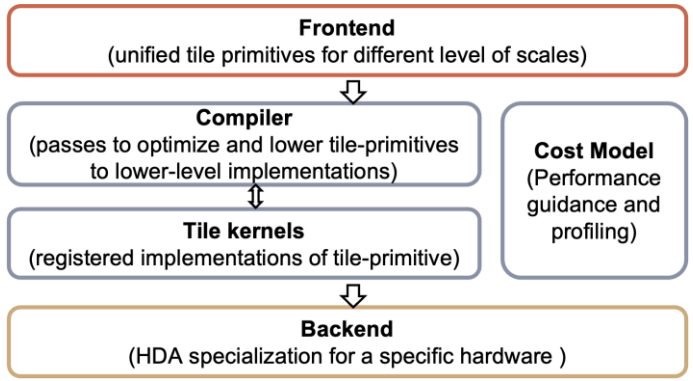




Tile Primitives	Type
T.gemm T.relu, T.reduce_sum, …	compute
T.copy, T.alloc, T.view, …	memory
T.copy, T.barrier, T.signal, T.allreduce, T.shift, …	communicate
T.Scale, T.Parallel, T.Pipeline, …	schedule

以Tile为基础的scale编程接口

System Overview



跨节点通信示例

```
# communication example: send data to neighbor GPU
with T.Scale("device") as dev_id, dev_num:
    T.copy(remote_B, local_A, dst=(dev_id + 1) % dev_num)
    T.barrier()
```

Cluster间的矩阵乘法示例

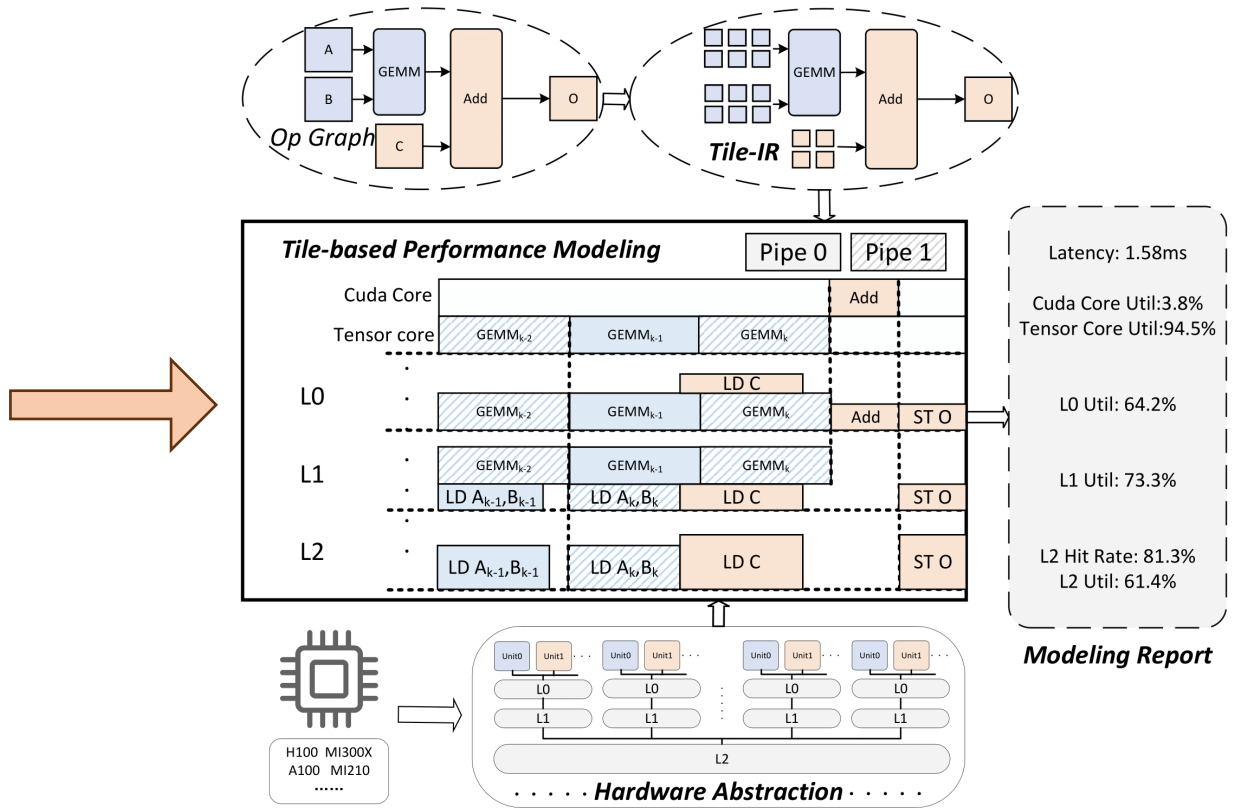
```
# cluster-level GEMM example
with T.Kernel(
    cta_cluster=(2),
    block=(block_M, block_N),
    threads=256
):
    with T.Scale("cta_cluster"):
        T.gemm(A, B, C)
```

TileSight: Cost Model and Auto Tuning

1. 专门用于加速大规模复杂Kernel（如 FlashAttention 和 FlashMLA）在大语言模型（LLM）中的性能优化，直接估计硬件利用率等信息。

2. 轻量级的自动调优框架，旨在为 GPU、CPU 和加速器等多种后端生成和评估高效的 tile 配置（即 tiling 策略或调度提示），帮助开发者快速找到性能优越的调度策略，减少手动调优时间

```python
@T.prim_func
def matmul_relu_kernel(
        A: T.Tensor((M, K), dtype),
        B: T.Tensor((K, N), dtype),
        C: T.Tensor((M, N), dtype),
):
    # Initialize Kernel Context
    with T.Kernel(T.ceildiv(N, block_N), T.ceildiv(M, block_M), threads=128) as (bx, by):
        A_shared = T.alloc_shared((block_M, block_K), dtype)
        B_shared = T.alloc_shared((block_K, block_N), dtype)
        C_local = T.alloc_fragment((block_M, block_N), accum_dtype)

        # Enable rasterization for better L2 cache locality (Optional)
        # T.use_swizzle(panel_size=10, enable=True)

        # Clear local accumulation
        T.clear(C_local)

        for ko in T.Pipelined(T.ceildiv(K, block_K), num_stages=3):
            # Copy tile of A
            # This is a sugar syntax for parallelized copy
            T.copy(A[by * block_M, ko * block_K], A_shared)

            # Copy tile of B
            T.copy(B[ko * block_K, bx * block_N], B_shared)

            # Perform a tile-level GEMM on the shared buffers
            # Currently we dispatch to the cute/hip on Nvidia/AMD GPUs
            T.gemm(A_shared, B_shared, C_local)

        # relu
        for i, j in T.Parallel(block_M, block_N):
            C_local[i, j] = T.max(C_local[i, j], 0)

        # Copy result back to global memory
        T.copy(C_local, C[by * block_M, bx * block_N])
```



Tile-AI RoadMap内的其他项目

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

TileLang生态中的Flash-Linear-Attention/FlagGems

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

用TileLang编写MegaKernels

谢谢！