Speaking Composable Kernel

Haocong Wang
(Representing the Composable Kernel Team @AMD)

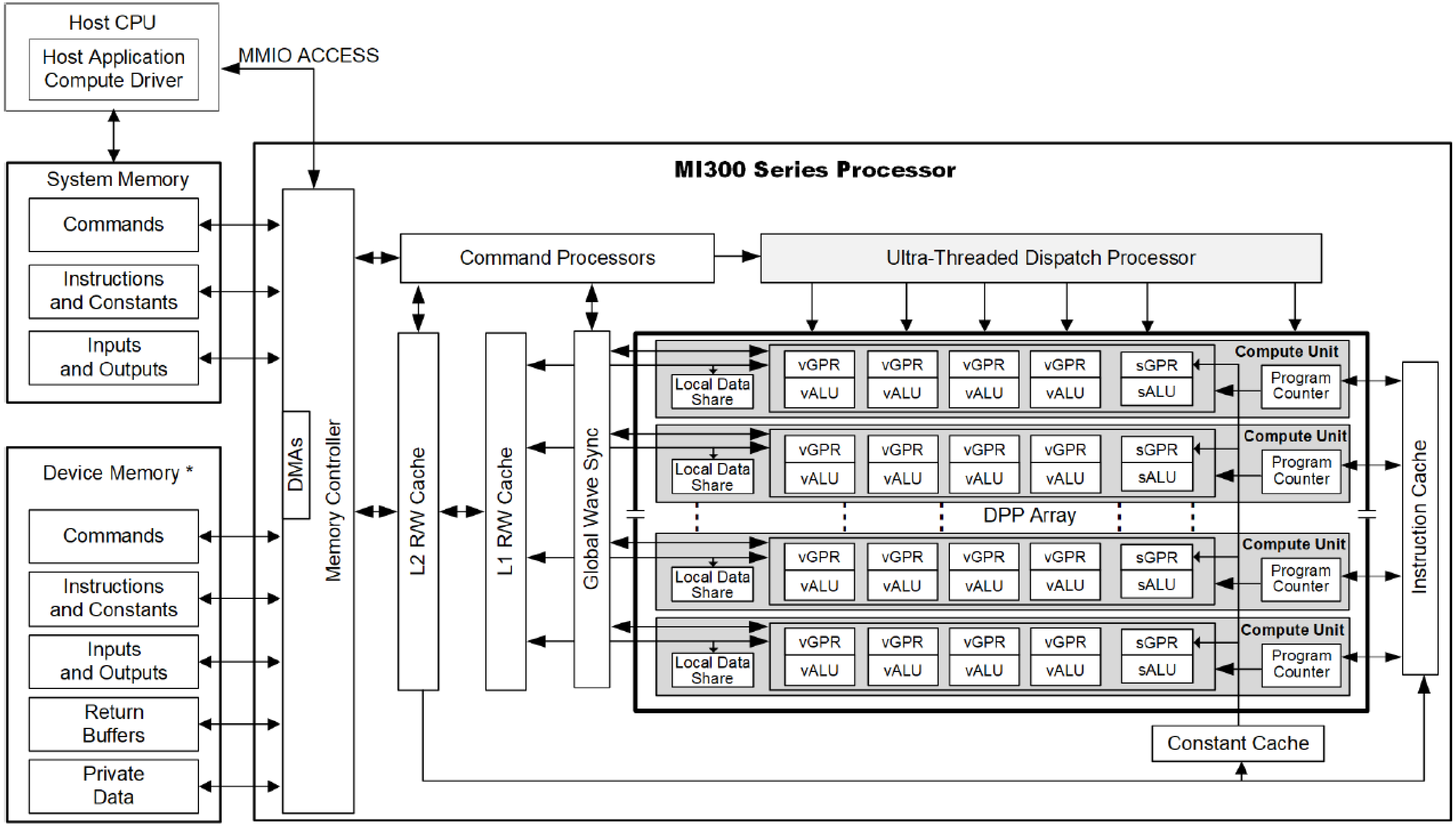
Courtesy: Chao Liu

July 20th, 2024

AMD
together we advance_

Agenda

- High GPU utilization is Challenging
- Composable Kernel
- Kernel customization
- ROCm Flash-Attention
- Q&A



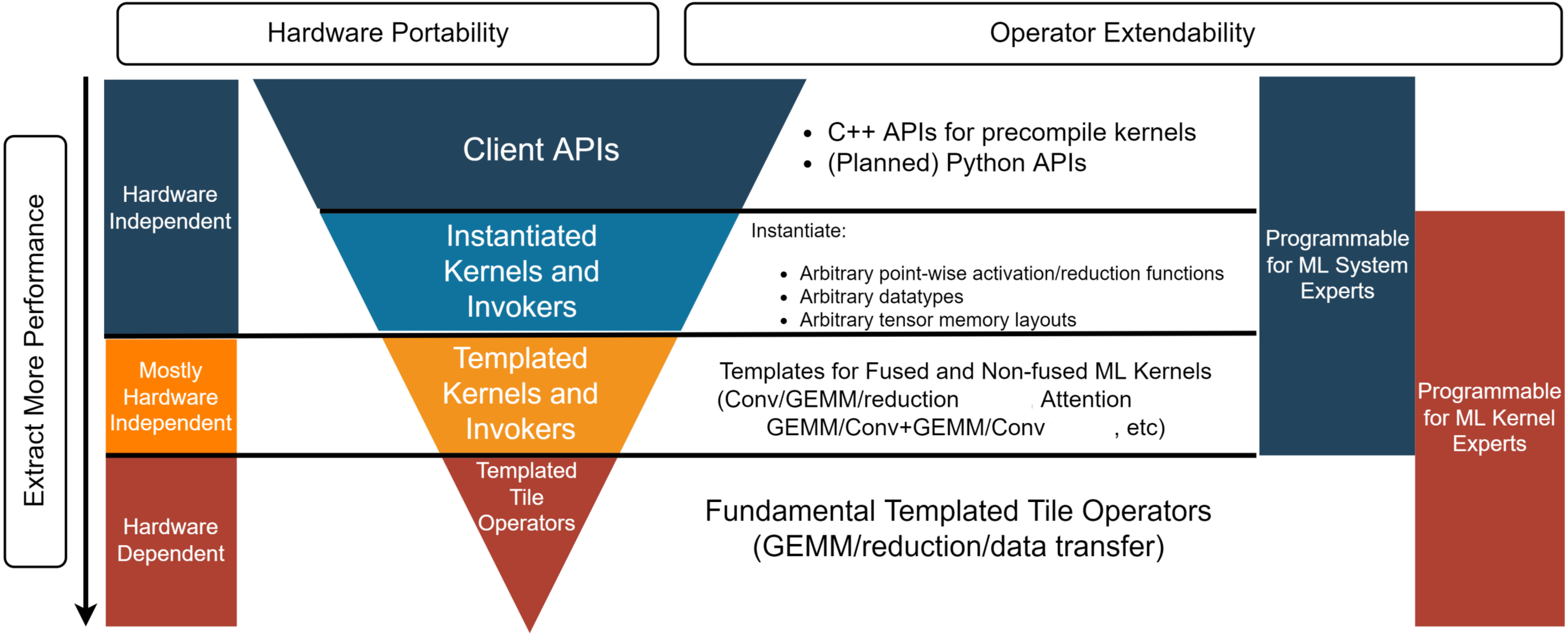
High GPU utilization is Challenging

Mapping custom AI workloads for high GPU utilization

- **Complexity of GPU programming model**
  - Complicated memory hierarchy, global memory, shared memory, register file, multiple-level cache…
  - Multiple types of compute units, VALU, DPP, matrix cores…
  - The rapid iteration of hardware, makes the above even harder for ordinary users, making it difficult to fully utilize the GPU's performance

- **Highly customizable algorithms**
  - CNN, high-dimensional, irregular image size, sophisticated mapping
  - Fused kernel, flash-attention for example, including multiple operations
  - The rapid development of machine learning has led to a surge in customized demands

- **Developer productivity**
  - Developers need a tool that is suitable for rapid development while also delivering high performance on GPU

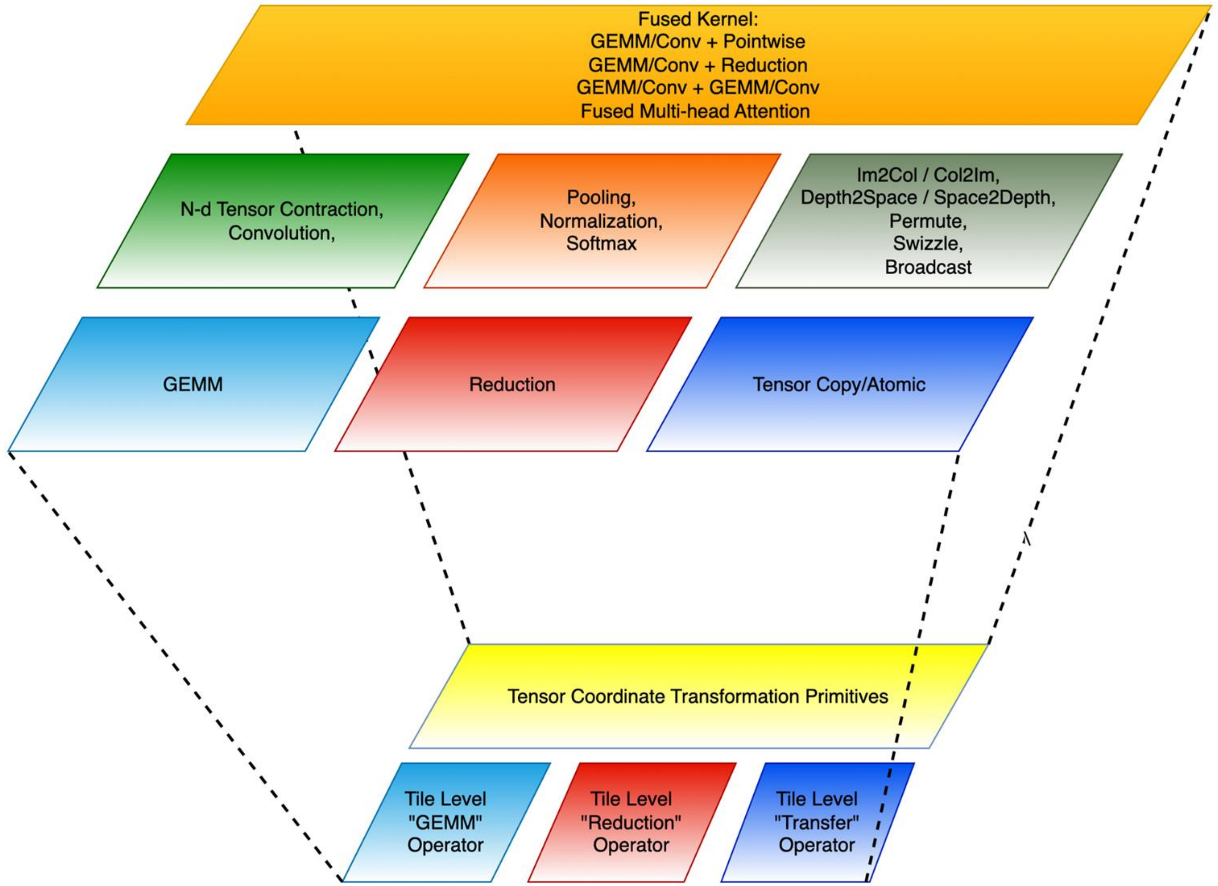


Composable Kernel

Design Philosophies

- At the core: a programming paradigm for Performance, Productivity & Portability
  - Systematic & Self-sufficient, Support all tensor operators, Express all optimization techniques, abstract all hardwave
  - Composable & Reusable coding component
- User productivity
  - Optimized hierarchical API calls
  - C++ templated
- Vendor optimized high performance code

Key Abstractions in CK



Coordinate Transformation Primitives

Reduce algorithm complexity

Tile Programming

Intuitive & Productive programming interface

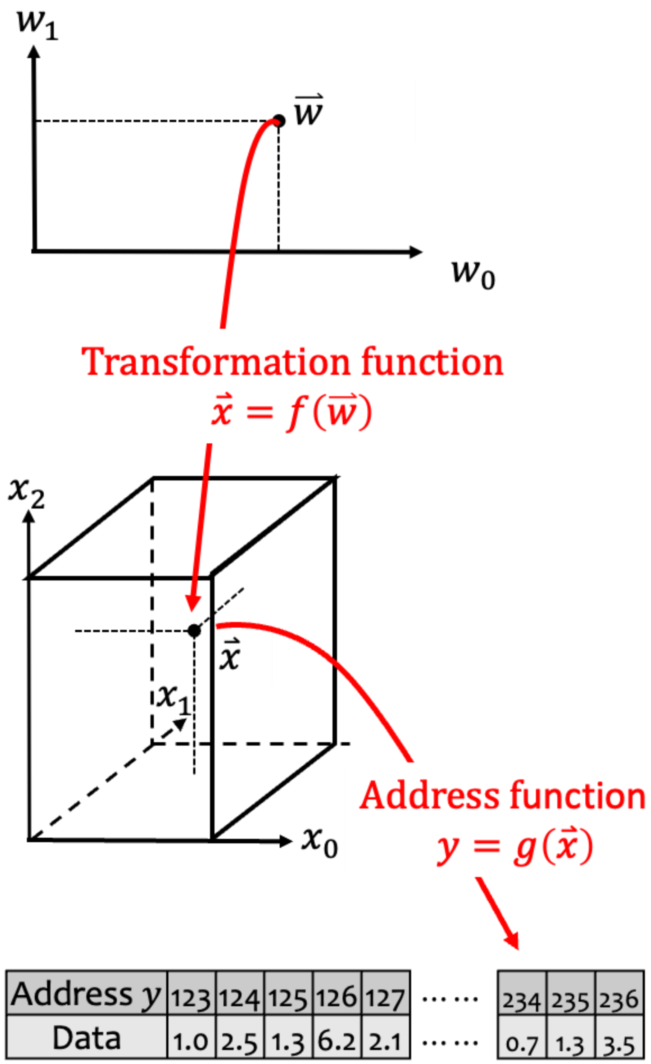
Coordinate Transformation Primitives

Abstraction

Transformed
Coordinate Space W

Naive
Coordinate Space X

Raw Memory
Coordinate Space Y



CK API

make_pad_transform()
make_merge_transform()
make_xor_transform()
make_passthrough_transform()
make_freeze_transform()
…

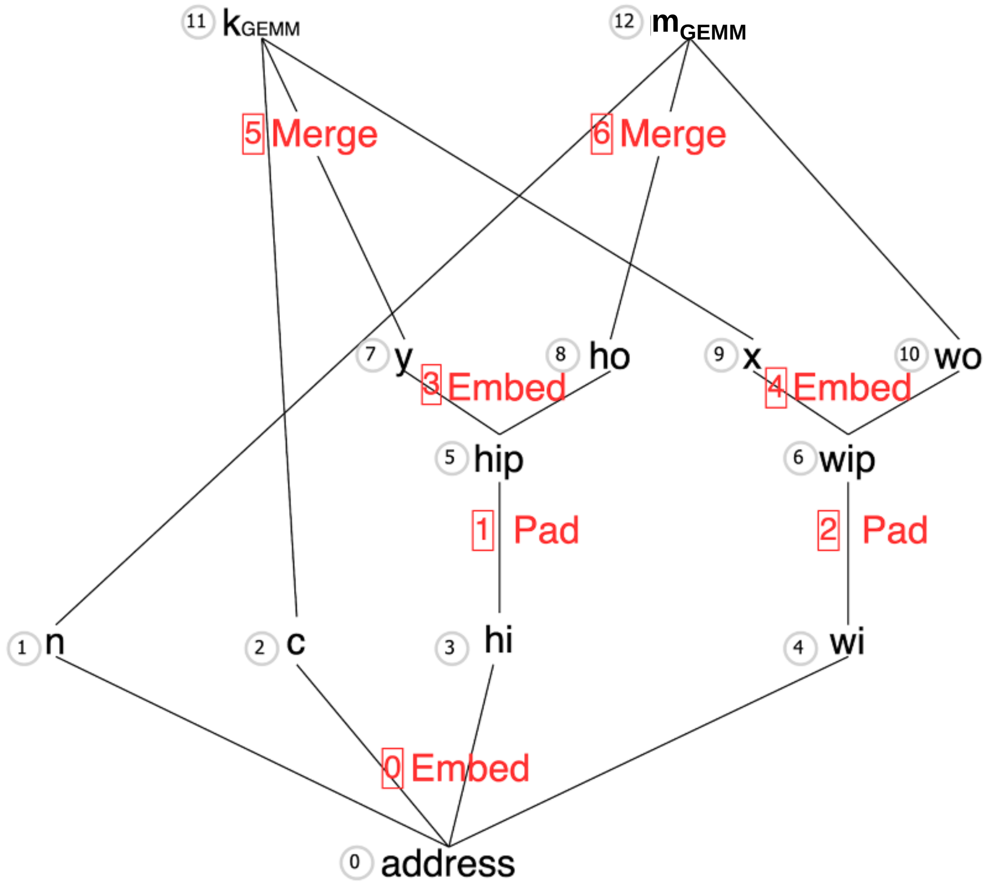
make_naive_tensor_view<>()

AddressSpaceEnum::Global
AddressSpaceEnum::Lds
AddressSpaceEnum::Vgpr…

InMemoryDataOperationEnum::Set
InMemoryDataOperationEnum::AtomicAdd

Coordinate Transformation Primitives

Implicit GEMM Convolution example



Im2Col Transformation Graph for Implicit GEMM Convolution

```
const auto a_n_hi_wi_c = make_naive_tensor_view_packed<AddressSpaceEnum::Global>(
            p_a_img, make_tuple(N, Hi, Wi, C), Number<32>{})

const auto a_n_hip_wip_c = transform_tensor_view(
            a_n_hi_wi_c,
            make_tuple(make_pass_through_transform(N),
                       make_pad_transform(Hi, InLeftPadH, InRightPadH),
                       make_pad_transform(Wi, InLeftPadW, InRightPadW),
                       make_pass_through_transform(C)),
            make_tuple(Sequence<0>{}, Sequence<1>{}, Sequence<2>{}, Sequence<3>{}),
            make_tuple(Sequence<0>{}, Sequence<1>{}, Sequence<2>{}, Sequence<3>{}));

const auto a_n_y_ho_x_wo_c = transform_tensor_view(
    a_n_hip_wip_c,
    make_tuple(
        make_pass_through_transform(N),
        make_embed_transform(make_tuple(Y, Ho), make_tuple(ConvDilationH, ConvStride
        make_embed_transform(make_tuple(X, Wo), make_tuple(ConvDilationW, ConvStride
        make_pass_through_transform(C)),
    make_tuple(Sequence<0>{}, Sequence<1>{}, Sequence<2>{}, Sequence<3>{}),
    make_tuple(Sequence<0>{}, Sequence<1, 2>{}, Sequence<3, 4>{}, Sequence<5>{}));

const auto src_gemmm_gemmk =
    transform_tensor_view(a_n_y_ho_x_wo_c,
                          make_tuple(make_merge_transform(make_tuple(N, Ho, Wo)),
                                     make_merge_transform(make_tuple(Y, X, C))),
                          make_tuple(Sequence<0, 2, 4>{}, Sequence<1, 3, 5>{}),
                          make_tuple(Sequence<0>{}, Sequence<1>{}));
```

Concise Im2Col implementation code with Coordinate Transformation Primitives

AMD
together we advance_

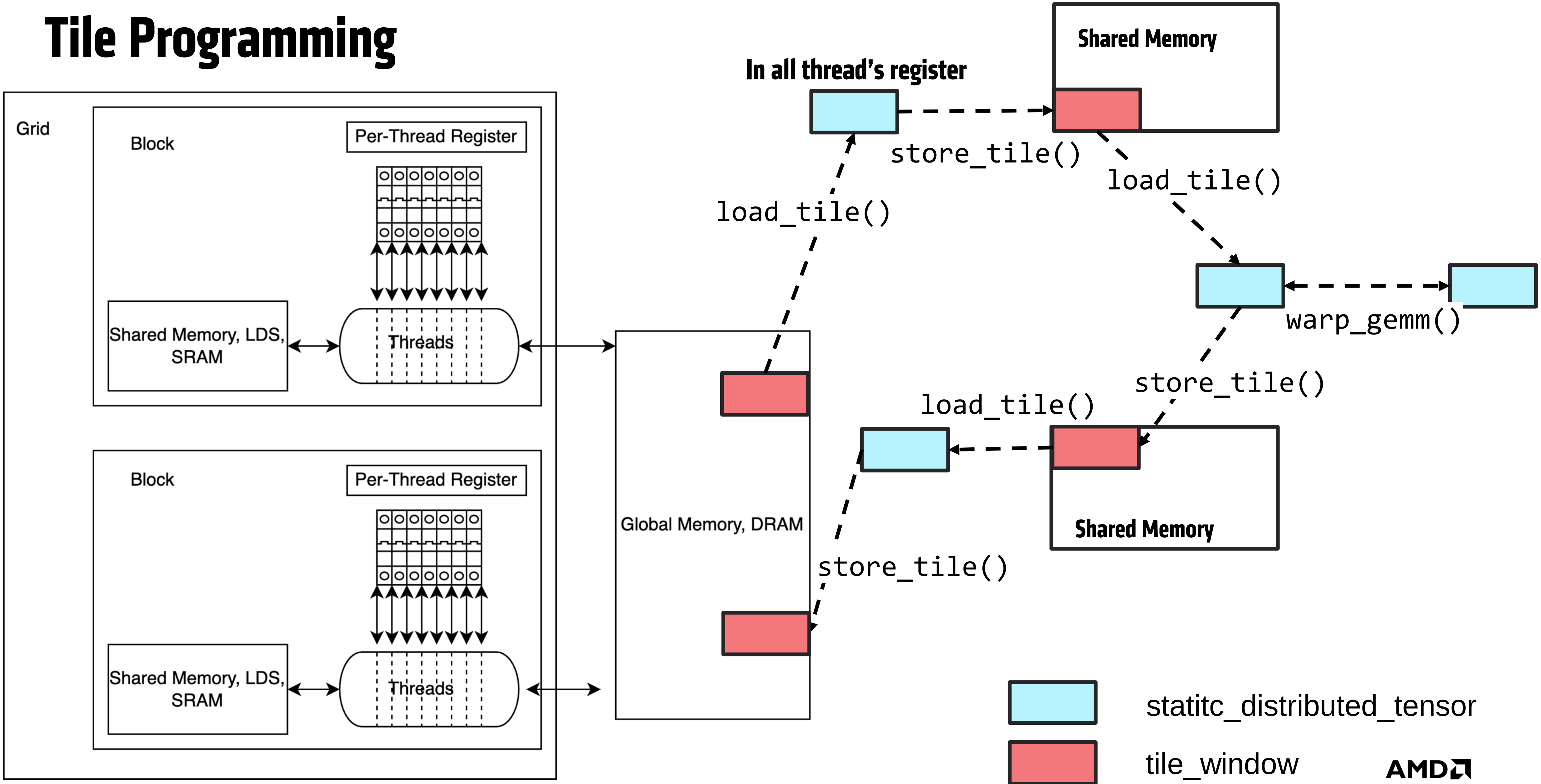
Tile Programming

- Goal: Non-compromised Productivity and Performance
  - Non-GPU experts can write functional kernels
  - GPU experts can hyper-optimize kernels
  - Architectural Independant code

- Design decision
  - Only tile level interface (both tensor operator and operand) to hide GPU complexity
  - Interface reflects conceptual data layout for math (not implementation)
  - Able to express all optimization

- Solution
  - Distributed Tensor - Tile level view data structure - regardless of private memory space
  - Generalize a set of reusable tile level operators and APIs
  - Policy - To inject optimization

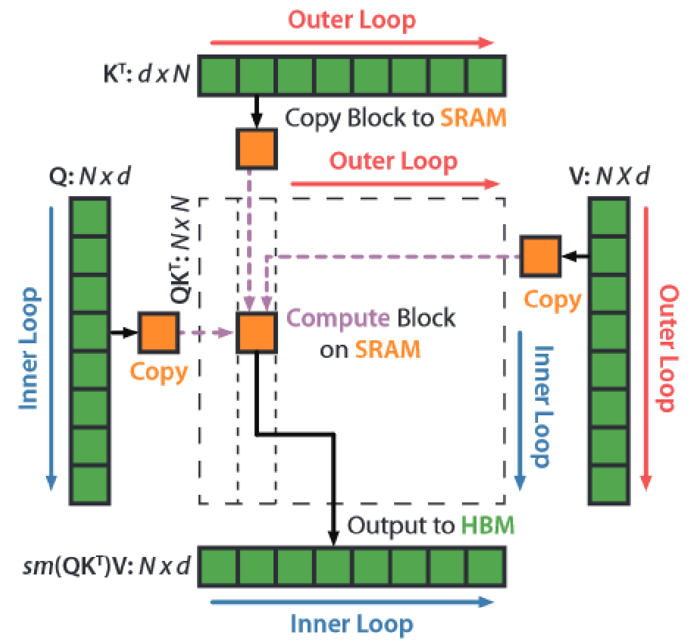
Tile Programming



[Public]

Tile Programming

Flash Attention forward pass example



for $1 \le j \le T_c$ **do**

Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

On chip, compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i\mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$

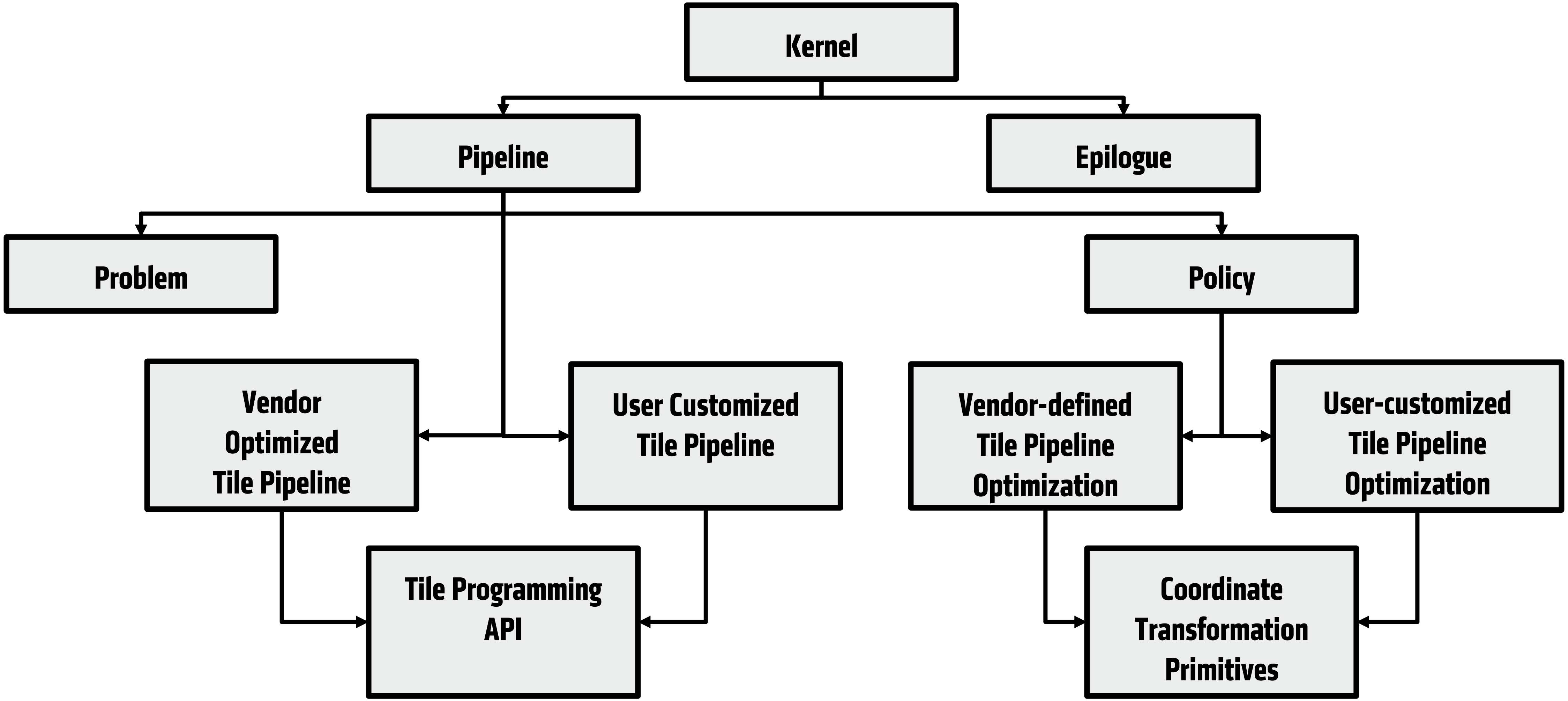
$\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$

$\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}}\ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$

On chip, compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1}\mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)}\mathbf{V}_j$.

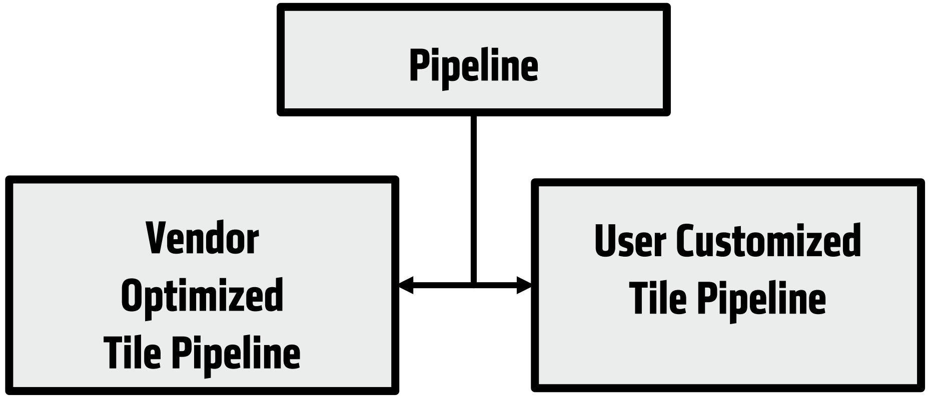
end for

```
// Sacc{j} = Q * K{j}
const auto s_acc =
    gemm0_pipeline(q_dram_window, k_dram_window, K0 / kK0PerBlock, smem_ptr);

// S{j}
const auto s =
    tile_elementwise_in(type_convert<SMPLComputeDataType, SaccDataType>, s_acc);

// m_local = rowmax(S{j})
auto m_local = block_tile_reduce<SMPLComputeDataType>(
    s, Sequence<1>{}, f_max, NumericLimits<SMPLComputeDataType>::Lowest());

block_tile_reduce_sync(m_local, f_max);

// m{j-1}
const auto m_old = m;

// m{j}
tile_elementwise_inout(
    [](auto& e0, auto e1, auto e2) { e0 = max(e1, e2); }, m, m_old, m_local);

// Pcompute{j}
auto p_compute =
    make_static_distributed_tensor<SMPLComputeDataType>(s.GetTileDistribution());

constexpr auto p_spans = decltype(p_compute)::GetDistributedSpans();

sweep_tile_span(p_spans[I0], [&](auto idx0) {
    constexpr auto i_idx = make_tuple(idx0);

    sweep_tile_span(p_spans[I1], [&](auto idx1) {
        constexpr auto i_j_idx = make_tuple(idx0, idx1);

        p_compute(i_j_idx) = math::exp(s[i_j_idx] - m[i_idx]);
    });
});

// rowsum(Pcompute{j})
auto rowsum_p = block_tile_reduce<SMPLComputeDataType>(
    p_compute, Sequence<1>{}, f_sum, SMPLComputeDataType{0});

block_tile_reduce_sync(rowsum_p, f_sum);
```

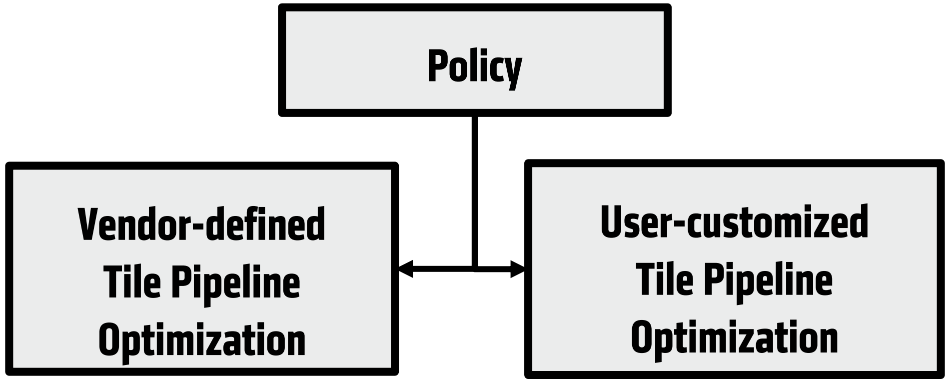


Kernel customization

Kernel customization

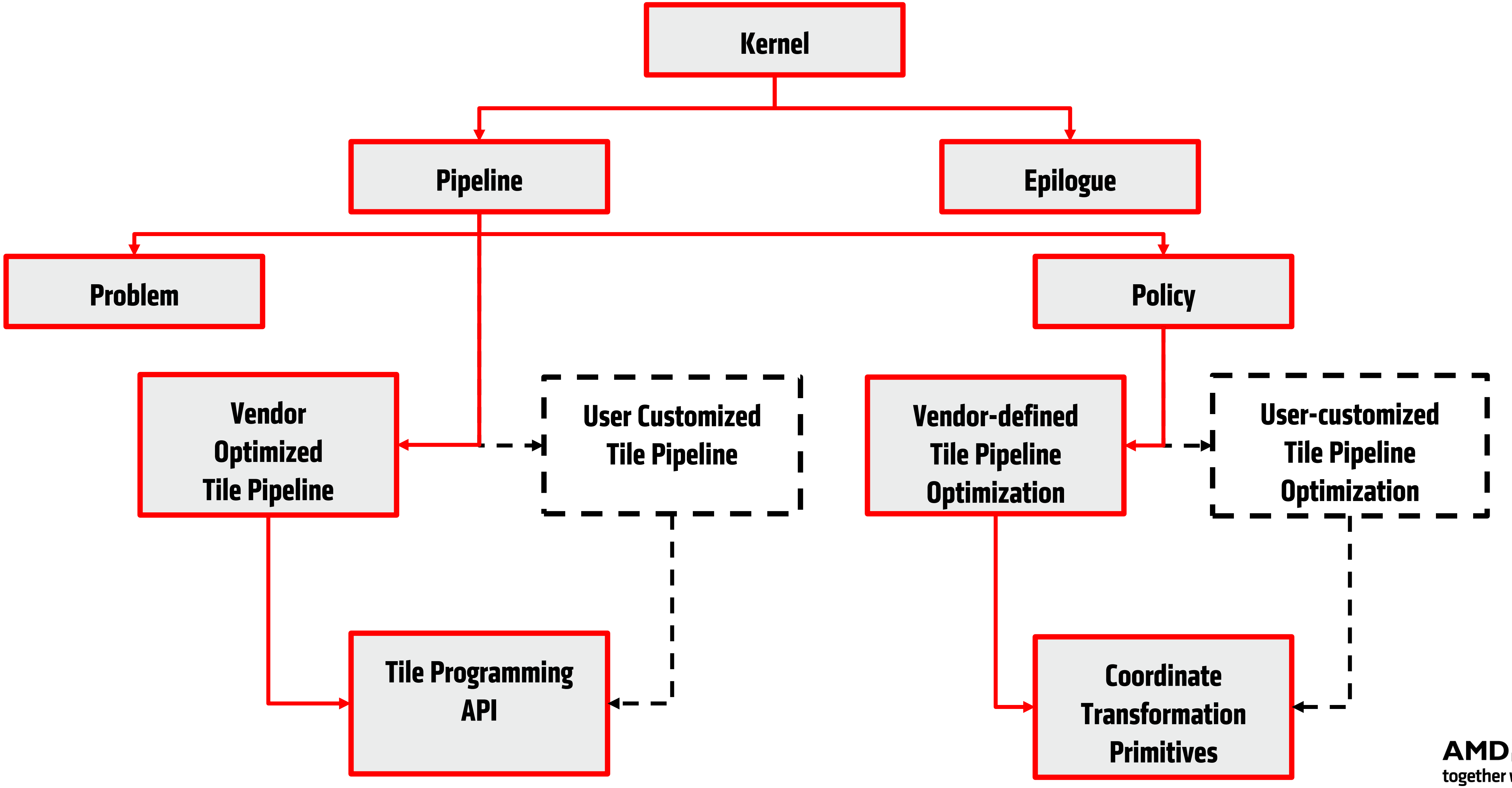


- **Pipeline**
  - Compose data movement and computation component
  - Tile Programming API to provide Intuitive & Productive programming interface

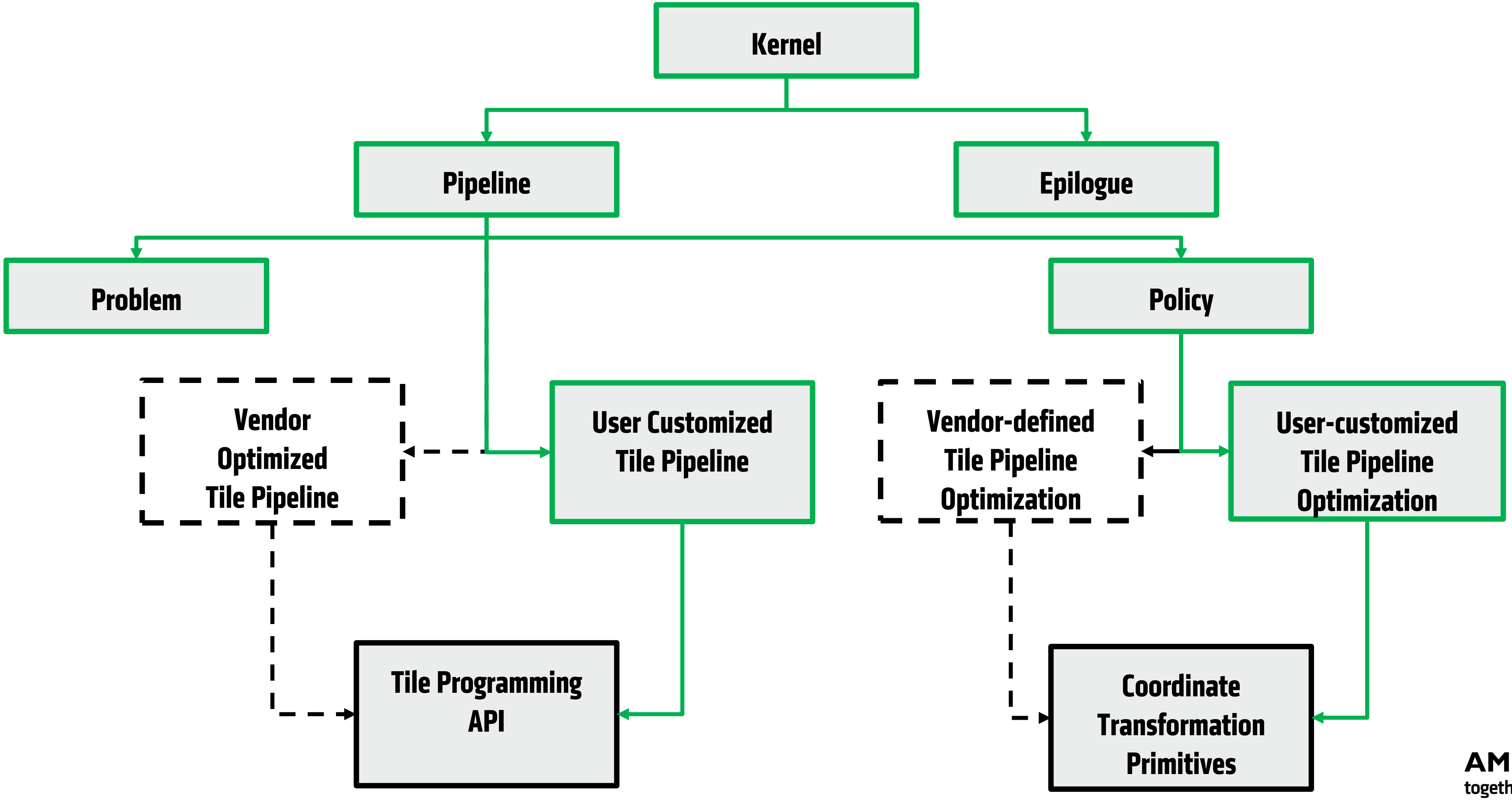


- **Policy**
  - Optimization applied to specific pipeline
  - Define data movement and computation component
  - Coordinate Transformation Primitives to hide complexity

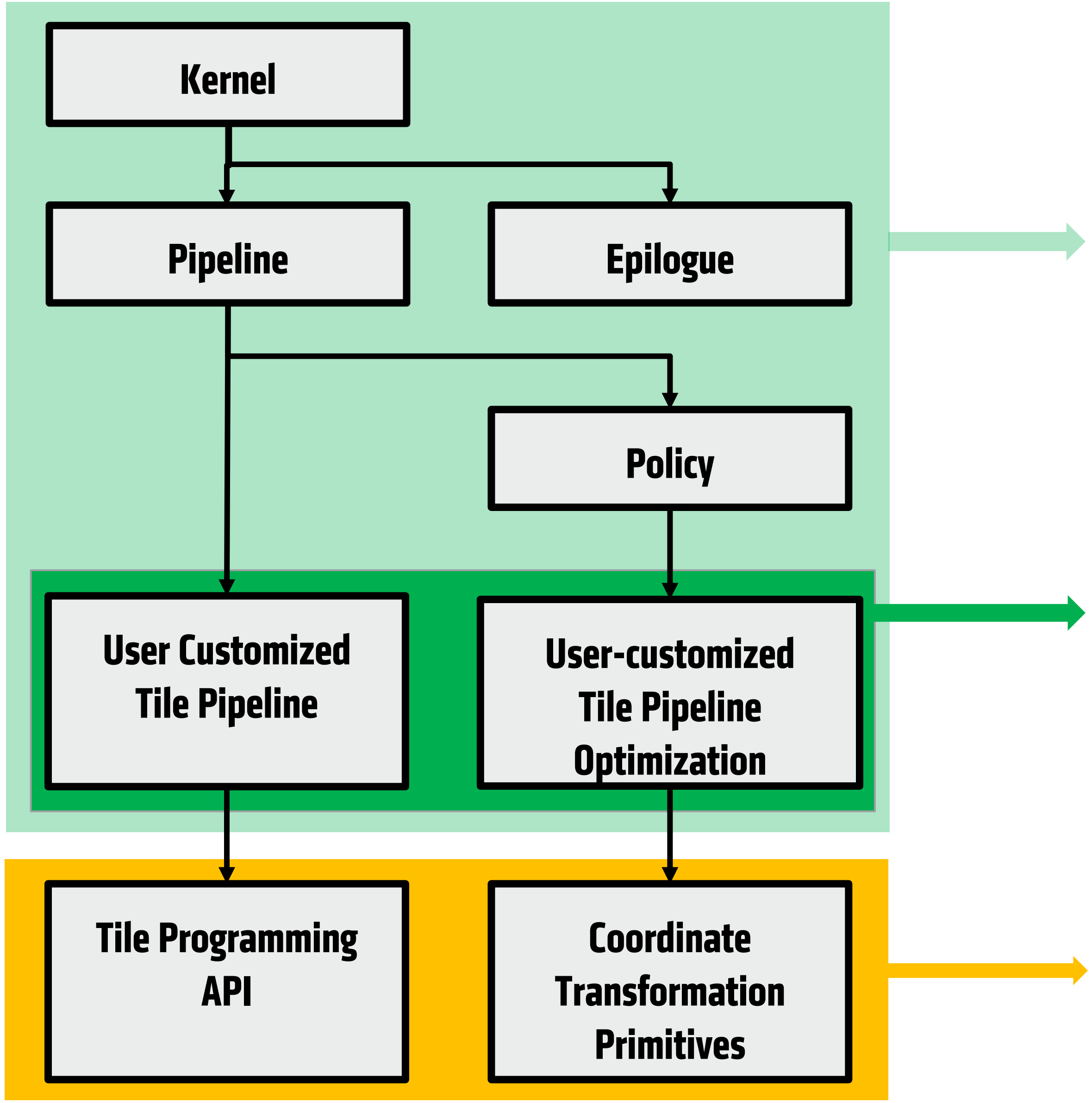
Vendor optimized kernel



User customized kernel



User customized kernel



If you want implement a new kernel

If you want implement a new optimization to existing kernel

AMDGPU expert maintain help you programming without tear

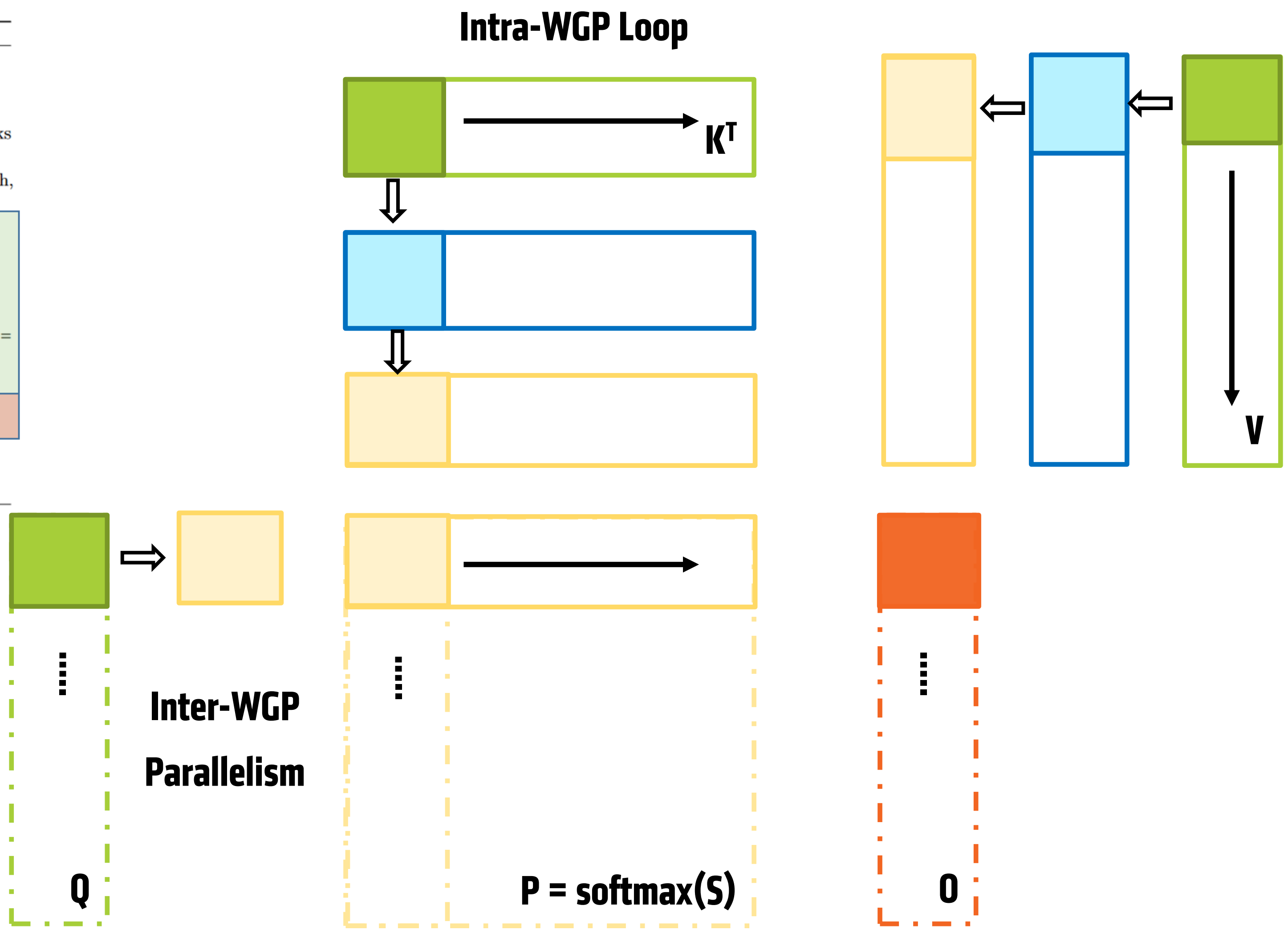
ROCm Flash-Attention

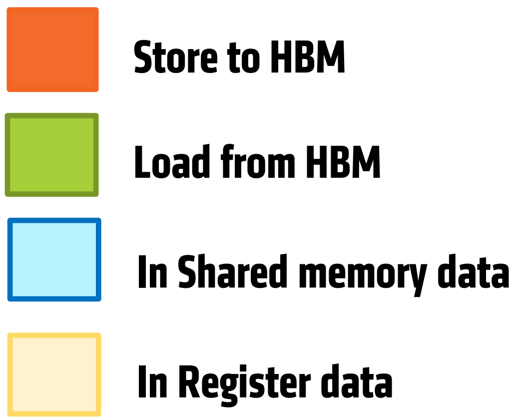
ROCm Flash-Attention

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
Initialize $\mathbf{O} = (0)_{N\times d} \in \mathbb{R}^{N\times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \dots, m_{T_r}$ of size B_r each.
for $1 \le j \le T_c$ **do**
  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
  for $1 \le i \le T_r$ **do**
    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
    On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
    On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1} (\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
  end for
end for
Return $\mathbf{O}$.





AMD
together we advance_

Impact

AMD Community › Blogs › AI › AMD Composable Kernel library: efficient fused ker...

AMD Composable Kernel library: efficient fused kernels for AI apps with just a few lines of code

Liu_Chao
Staff

6 0 14K

10-26-2022 02:42 PM

pytorch / torch / _inductor / config.py

Code Blame 1024 lines (787 loc) · 37.5 KB Raw

```
# Specify candidate backends for gemm autotune.
# Possible choices are combinations of: ATen, Triton, CUTLASS, CK, CPP.
# ATen: default Pytorch ATen kernels.
# Triton: Triton templates defined in torch inductor (AMD and NVidia GPUs).
# CUTLASS: Cutlass templates and kernels (NVidia GPUs only).
# CK: Composable Kernel templates and kernels (AMD Instinct GPUs only).
# CPP: CPP templates and kernels for CPU.
```

Support AMD ROCm on FlashAttention 2 #1010

Open rocking5566 wants to merge 34 commits into Dao-AILab:main from ROCm:ck_tile

CLASS xformers.ops.fmha.ck.FwOp [SOURCE]

xFormers' MHA kernel based on Composable Kernel.

CLASS xformers.ops.fmha.ck.BwOp [SOURCE]

xFormers' MHA kernel based on Composable Kernel.

AMD GPUOpen

Search GPUOpen...

HOME SOFTWARE DOCS

Home » News » Announcement

AMD support for Microsoft® DirectML optimization of Stable Diffusion

May 23, 2023

ECOSYSTEM

by Abhi Venigalla
on June 30, 2023

SHARE

Training LLMs with AMD MI250 GPUs and MosaicML

With the release of PyTorch 2.0 and ROCm 5.4, we are excited to announce that LLM training works out of the box on AMD MI250 accelerators with zero code changes and at high performance!

AMD Community › Blogs › Instinct Accelerators › Competitive performance claims and industry le

Competitive performance claims and industry leading Inference performance on AMD Instinct MI300X

Ian_Ferreira
Staff

10 0 30.5K

Accelerating Stable Diffusion Inference with ONNX Runtime

Tianlei Wu · Follow
Published in Microsoft Azure · 4 min read · May 10

AMD
together we advance_

Reference

1. Dao, Tri, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness." arXiv, June 23, 2022. https://doi.org/10.48550/arXiv.2205.14135.

To get access of Composable Kernel:
https://github.com/ROCm/composable_kernel

For Composable Kernel document:
https://rocm.docs.amd.com/projects/composable_kernel/en/latest/

AMDGPU architecture document:
https://gpuopen.com/amd-gpu-architecture-programming-documentation/

Q&A

COPYRIGHT AND DISCLAIMER

AMD