AMD ROCm™ Basics & Optimization Overview

Joe Liu 刘仕洲
Jan 2026

together we advance_

Agenda

1. Introduction to the AMD ROCm™ Software Stack
2. Transitioning Workloads to AMD GPUs
3. Performance Optimization
   - Optimizing application using popular libraries
   - Profiling the models
   - Adding HIP kernel to implement a custom layer
4. Available Collaterals, Q&A

AMD ROCm

Optimized AI Software Stack

AI Models and **Algorithms**

PyTorch + Other Frameworks

AI Ecosystem optimized for AMD

Libraries

Compilers and **Tools**

Runtime

AMD ROCm | Open, Proven, Ready software stack

AMD GPU

Leadership performance

- Commitment to **Open-Source**
- **No Code Change** Execution
- Optimized for **Generative AI**

MIT/BSD License
Apache License
GPL License

AMD ROCm™ Software Stack

AI Solutions & Services
- Business Services
- Finetuned Models

Ecosystem
- mosaic^ML
- Hugging Face
- PyTorch
- TensorFlow
- ONNX
- Triton
- deepspeed
- OpenXLA

Open Software Platform
AMD ROCm

AMD GPU

Accelerated AI Math & Communication Libraries

HPC Focused
- rocALUTION
- rocSOLVER
- rocPRIM
- rocSPARSE
- rocThrust

AI Focused
- MIOpen
- MIGraphX
- rocWMMA
- hipBLASLt
- Composable Kernel

- rocBLAS
- rocRAND
- rocFFT
- RCCL

Compilers & Tools
- Compiler
- Profile/Tracer
- Debugger
- HIPIFY

Drivers Runtimes
RedHat, SLES & Ubuntu Device Drivers and Run-time

AMD together we advance_

Library and Compiler Based Optimization

1 Max Performance
Framework Operator Optimization

- PyTorch → Target Specific Optimizations
- TensorFlow → Target Specific Optimizations
- HIP
- Math Libraries | Communication Libraries
- HIPCC Compilation
- AMD GPUs

2 Max Portability
IR-based Optimization

- PyTorch | TensorFlow | JAX
- Vendor-agnostic Optimizations
  - OpenAI Triton
  - OpenXLA
- AMD EPYC™ CPU | AMD GPUs | ……

Agenda

1. Introduction to the AMD ROCm™ Software Stack
2. Transitioning Workloads to AMD GPUs
3. Performance Optimization
    - Optimizing application using popular libraries
    - Profiling the models
    - Adding HIP kernel to implement a custom layer
4. Available Collaterals, Q&A

Transitioning AI Workloads to AMD GPUs

	Nvidia		AMD ROCm™ SW
ML Frameworks Python	PyTorch ONNX TensorFlow	Drop-in (Out-of-the-box) Support →	PyTorch ONNX TensorFlow
ML Kernel Development C++, Triton IR	TRITON KERNELS → CUDA Triton Backend CUDA KERNELS → NVCC	Drop-in → Port / Optimize →	TRITON KERNELS → ROCM Triton Backend CUDA KERNELS → *HIPIFY* → HIPCC
ML Libraries C++	cuBLASlt, cuSparse, cuFFT, NCCL, cuDNN…	Mirror Equivalent Libraries →	hipBLASlt, rocSparse, rocFFT, RCCL, MIOpen…

ROCm™ Software: Can You Spot a Difference?

NVIDIA CUDA

```python
import torch
import torch.nn as nn

# Get cpu or gpu device for training.
device = "cuda:0" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

# Define model
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
                nn.Linear(28 * 28, 512),
                nn.ReLU(),
                nn.Linear(512, 512),
                nn.ReLU(),
                nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = Network().to(device)
print(model)
```

AMD ROCm™ Software

```python
import torch
import torch.nn as nn

# Get cpu or gpu device for training.
device = "cuda:0" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

# Define model
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
                nn.Linear(28 * 28, 512),
                nn.ReLU(),
                nn.Linear(512, 512),
                nn.ReLU(),
                nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = Network().to(device)
print(model)
```


together we advance_

PyTorch 2.8 Easily Enabled on AMD GPUs

Step 1. Install ROCm™ Software (Driver and SDK)

Step 2. Install Pip Wheel From Pytorch.Org

Step 3. Run Existing Code -- No Changes Required

PyTorch Build	Stable (2.9.1)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 12.6	CUDA 12.8	ROCm 6.4	CPU
Run this Command	*pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm6.4*			

```python
import torch
import torch.nn as nn

# Get cpu or gpu device for training.
device = "cuda:0" if torch.cuda.is_available() else
"cpu"
print(f"Using {device} device")

# Define model
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
                nn.Linear(28 * 28, 512),
                nn.ReLU(),
                nn.Linear(512, 512),
                nn.ReLU(),
                nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = Network().to(device)
print(model)
```

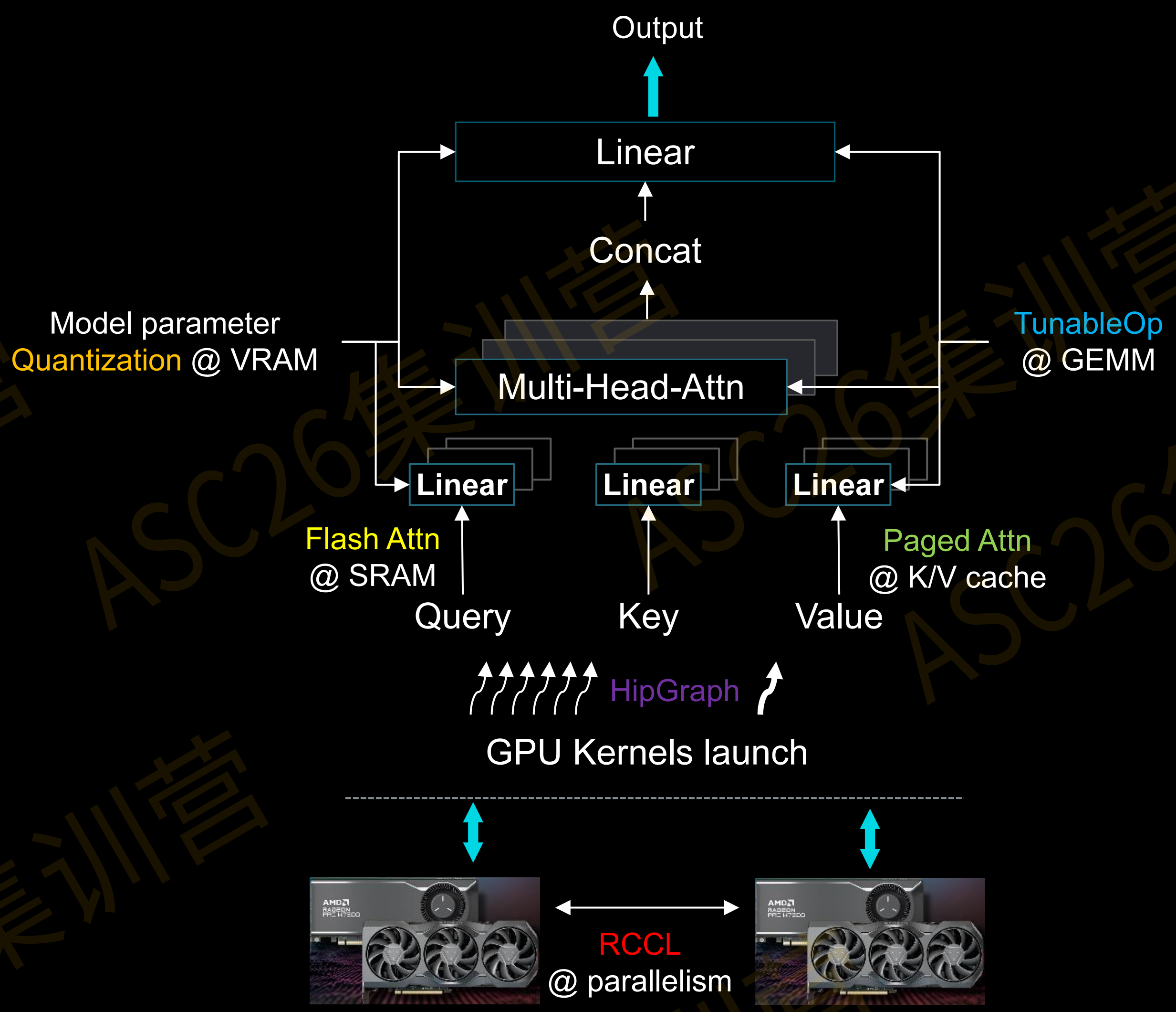
Optionally Install Docker containers from:

- *rocm/pytorch:latest*
- *rocm/pytorch-nightly:latest*

Agenda

1. Introduction to the AMD ROCm™ Software Stack
2. Transitioning Workloads to AMD GPUs
3. Performance Optimization
   - Optimizing application using popular libraries
   - Profiling the models
   - Adding HIP kernel to implement a custom layer
4. Available Collaterals, Q&A

Inference Challenges and Optimization Opportunities



Flash Attention, Xformers

- Tiling of input sequence in GPU SRAM to reduce VRAM data movement

Paged Attention

- Partitioned KV cache into fixed size blocks to reduce memory usage

GEMM Optimization – PyTorch TunableOp

- Automatic selection of the best performing GEMM kernels

Graph Optimization – HipGraph

- Launch multiple kernels through a single CPU operation

Collective communication – RCCL

- Collective Ops across multiple devices to support Tensor/Pipeline parallel

Quantization – GPTQ, Bitsandbytes

- Weight-only compression to reduce video memory footprint

AMD
together we advance_

Portability - Libraries

ATTENTION

XFORMERS | FLASH_ATTN | AITER

```python
import xformers.ops as xops

out = xops.memory_efficient_attention(q,
                                      k,
                                      v,
                                      attn_bias=None,
                                      op       =None)
```

```python
from flash_attn import flash_attn_varlen_func

#  batch and sequence dimensions merged into a single dimension
q, k, v = (rearrange(x, "b s ... -> (b s) ...")
              for x in [q, k, v])

out = flash_attn_varlen_func(q,
                             k,
                             v,
                             cu_seqlens_q=cu_seqlens,
                             cu_seqlens_k=cu_seqlens,
                             max_seqlen_q=max_seqlen,
                             max_seqlen_k=max_seqlen)
```

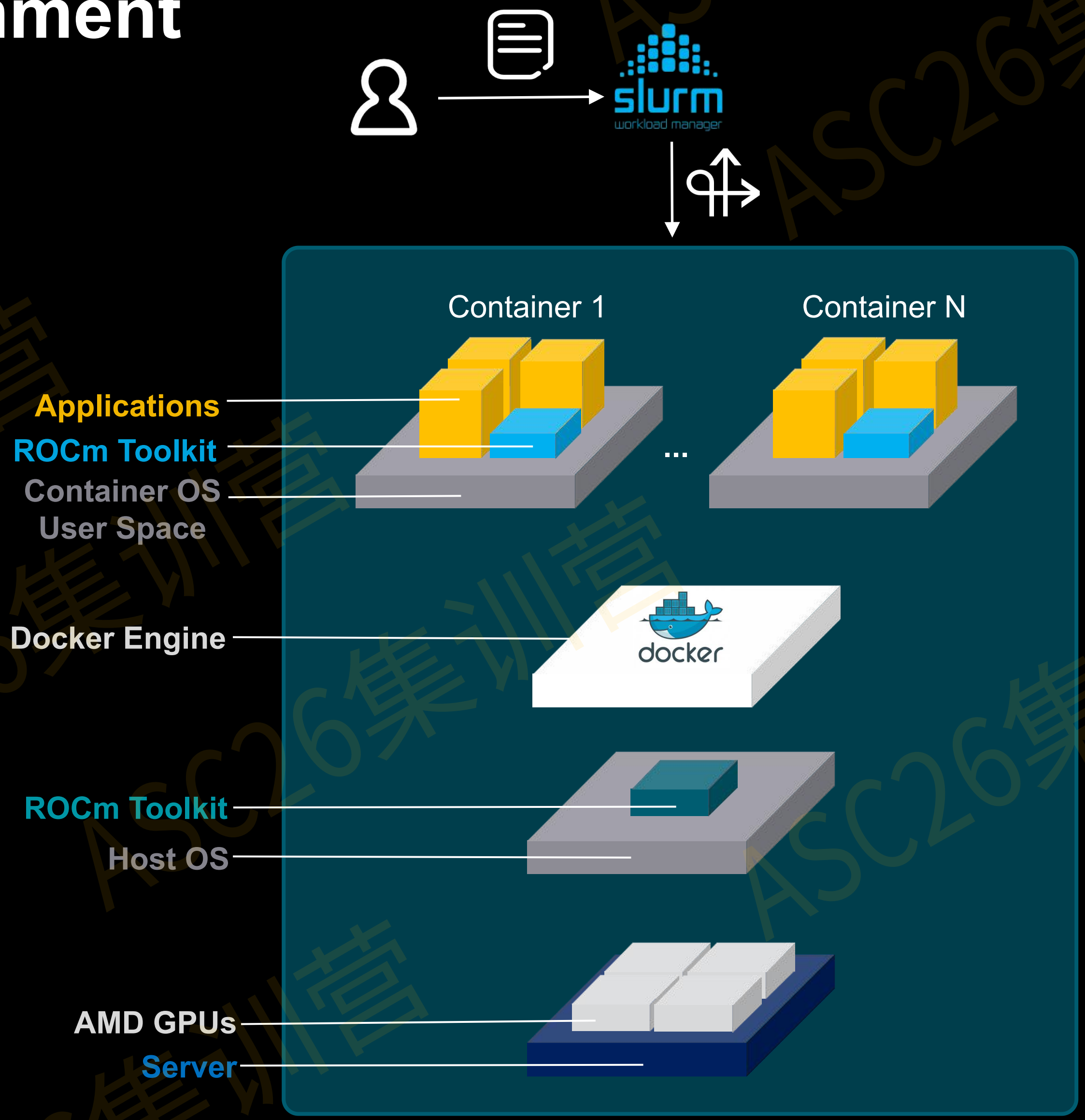
```python
from aiter import flash_attn_varlen_func

#  batch and sequence dimensions merged into a single dimension
q, k, v = (rearrange(x, "b s ... -> (b s) ...")
              for x in [q, k, v])

out = flash_attn_varlen_func(q,
                             k,
                             v,
                             cu_seqlens_q=cu_seqlens,
                             cu_seqlens_k=cu_seqlens,
                             max_seqlen_q=max_seqlen,
                             max_seqlen_k=max_seqlen)
```

Agenda

1. Introduction to the AMD ROCm™ Software Stack
2. Transitioning Workloads to AMD GPUs
3. Performance Optimization
   - Optimizing application using popular libraries
   - Profiling the models
   - Adding HIP kernel to implement a custom layer
4. Available Collaterals, Q&A

The Components in the Environment

- User submits jobs (sbatch / srun)
- Slurm scheduling layer
  - Allocate nodes / CPU / GPU
  - Launch the container runtime
- Container layer (Docker / Apptainer)
  - Application
  - ROCm user space (HIP Runtime / rocBLAS / MIOpen)
- Host driver layer
  - ROCm driver + kernel
  - /dev/kfd, /dev/dri device mapping
- Hardware layer
  - AMD GPUs & CPUs



[Public]

The Profiling Tools and Visualization - rocm-smi

- A command-line utility and library provided by ROCm for monitoring the following AMD GPU status:
  - Power, temperature, clocks (gfx/mem), voltage, fan speed
  - GPU utilization, memory usage (VRAM/GTT), PCIe link speed/width
- Typical usages:

```
# Show a quick summary of all GPUs
rocm-smi

# Detailed power, temps, clocks, and utilization
rocm-smi --showpower --showtemp --showclocks --showuse

# Memory usage and PCIe info
rocm-smi --showmemuse -showbus

# List GPU processes
rocm-smi -showpids

# Real-time monitoring (refresh every 0.1s)
watch -n 0 rocm-smi
```

```
watch -c rocm-smi --showclocks
```

```
Every 2.0s: rocm-smi --showclocks


========================== ROCm System Management Interface ==========================
============================== Current clock frequencies ==============================
GPU[0]          : dcefclk clock level: 0: (145Mhz)
GPU[0]          : fclk clock level: 1: (1000Mhz)
GPU[0]          : mclk clock level: 0: (96Mhz)
GPU[0]          : sclk clock level: 1: (0Mhz)
GPU[0]          : socclk clock level: 1: (600Mhz)
GPU[0]          : pcie clock level: 0 (16.0GT/s x16)
GPU[1]          : dcefclk clock level: 0: (145Mhz)
GPU[1]          : fclk clock level: 1: (1000Mhz)
GPU[1]          : mclk clock level: 0: (96Mhz)
GPU[1]          : sclk clock level: 1: (0Mhz)
GPU[1]          : socclk clock level: 1: (600Mhz)
GPU[1]          : pcie clock level: 0 (16.0GT/s x16)
GPU[2]          : dcefclk clock level: 0: (145Mhz)
GPU[2]          : fclk clock level: 1: (1000Mhz)
GPU[2]          : mclk clock level: 0: (96Mhz)
GPU[2]          : sclk clock level: 1: (0Mhz)
GPU[2]          : socclk clock level: 1: (600Mhz)
GPU[2]          : pcie clock level: 0 (16.0GT/s x16)
GPU[3]          : dcefclk clock level: 0: (145Mhz)
GPU[3]          : fclk clock level: 1: (1000Mhz)
GPU[3]          : mclk clock level: 0: (96Mhz)
GPU[3]          : sclk clock level: 1: (0Mhz)
GPU[3]          : socclk clock level: 1: (600Mhz)
GPU[3]          : pcie clock level: 0 (16.0GT/s x16)
GPU[4]          : dcefclk clock level: 0: (145Mhz)
GPU[4]          : fclk clock level: 1: (1000Mhz)
GPU[4]          : mclk clock level: 0: (96Mhz)
GPU[4]          : sclk clock level: 1: (0Mhz)
GPU[4]          : socclk clock level: 1: (600Mhz)
GPU[4]          : pcie clock level: 0 (16.0GT/s x16)
GPU[5]          : dcefclk clock level: 0: (145Mhz)
GPU[5]          : fclk clock level: 1: (1000Mhz)
GPU[5]          : mclk clock level: 0: (96Mhz)
GPU[5]          : sclk clock level: 1: (0Mhz)
GPU[5]          : socclk clock level: 1: (600Mhz)
GPU[5]          : pcie clock level: 0 (16.0GT/s x16)
GPU[6]          : dcefclk clock level: 0: (145Mhz)
GPU[6]          : fclk clock level: 1: (1000Mhz)
GPU[6]          : mclk clock level: 0: (96Mhz)
GPU[6]          : sclk clock level: 1: (0Mhz)
GPU[6]          : socclk clock level: 1: (600Mhz)
GPU[6]          : pcie clock level: 0 (16.0GT/s x16)
GPU[7]          : dcefclk clock level: 0: (145Mhz)
GPU[7]          : fclk clock level: 0: (601Mhz)
GPU[7]          : mclk clock level: 0: (96Mhz)
GPU[7]          : sclk clock level: 1: (0Mhz)
GPU[7]          : socclk clock level: 0: (500Mhz)
GPU[7]          : pcie clock level: 0 (16.0GT/s x16)
=======================================================================================
================================== End of ROCm SMI Log ================================
```


AMD
together we advance_

The Profiling Tools and Visualization

PyTorch

PyTorch Profiler

- https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html

```
import torch
from torch.profiler import profile, record_function, ProfilerActivity
```

AMD ROCm

ROCProfiler

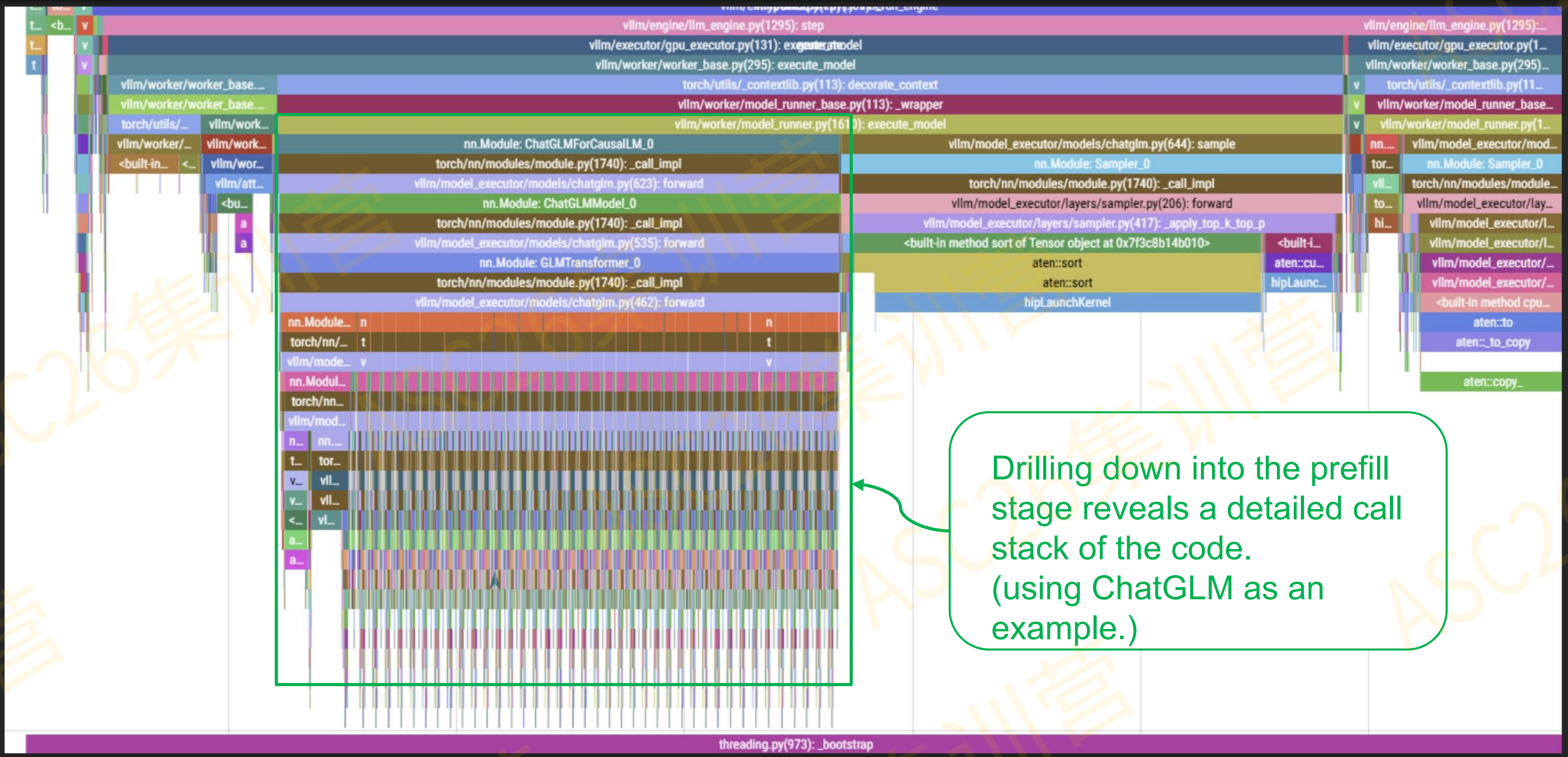
- https://rocm.docs.amd.com/projects/rocprofiler/en/latest/install/install.html
- *rocprof and rocprofv2 are included as standard components of the ROCm distribution*

```
rocprof -d outputFolder --hip-trace ./Matrixtranspose
```

- *ROCTracer API is a library that requires minor code modification in the application to be traced but provides greater flexibility*

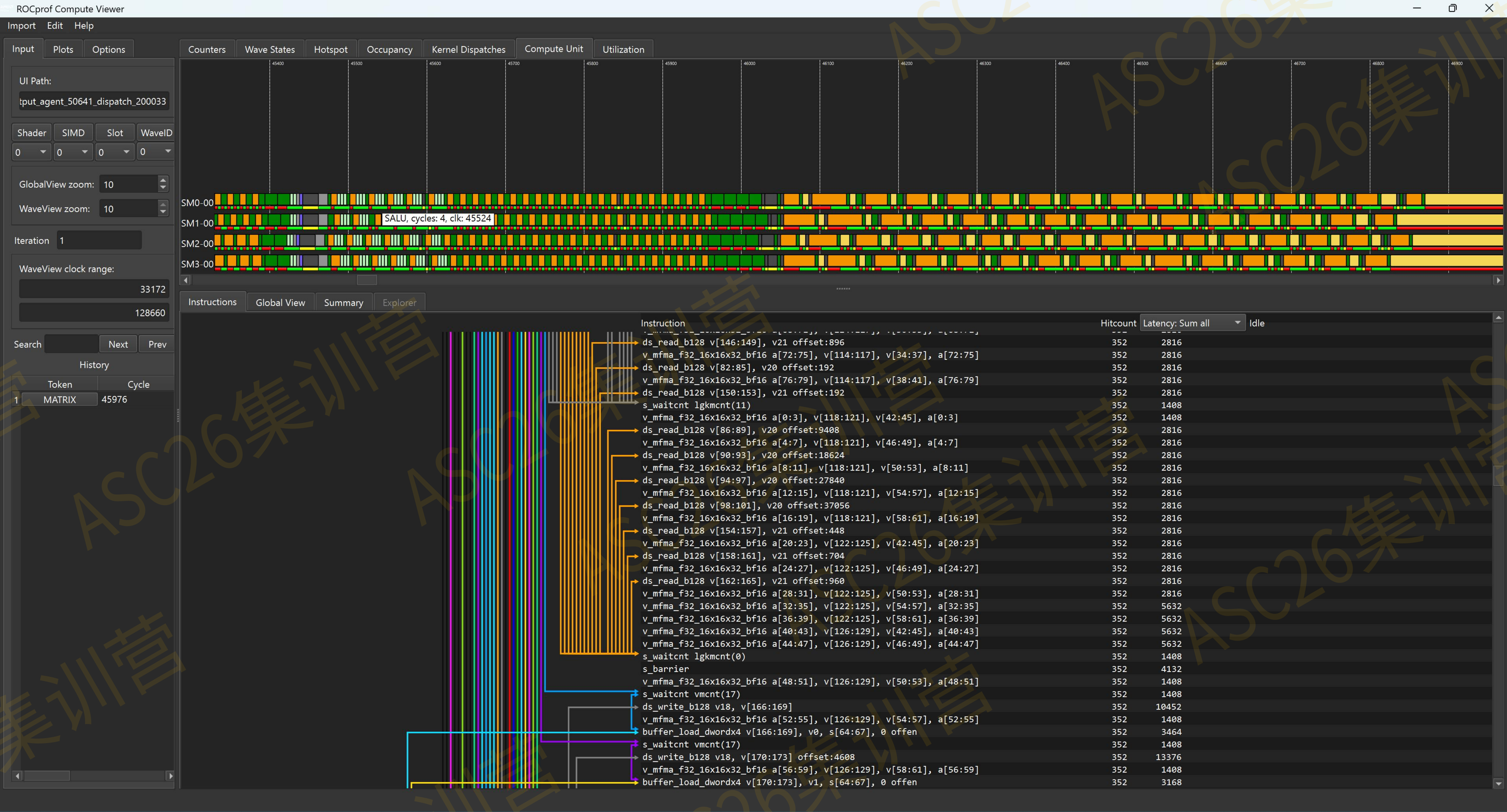
Perfetto

The Profiling Tools and Visualization - Samples



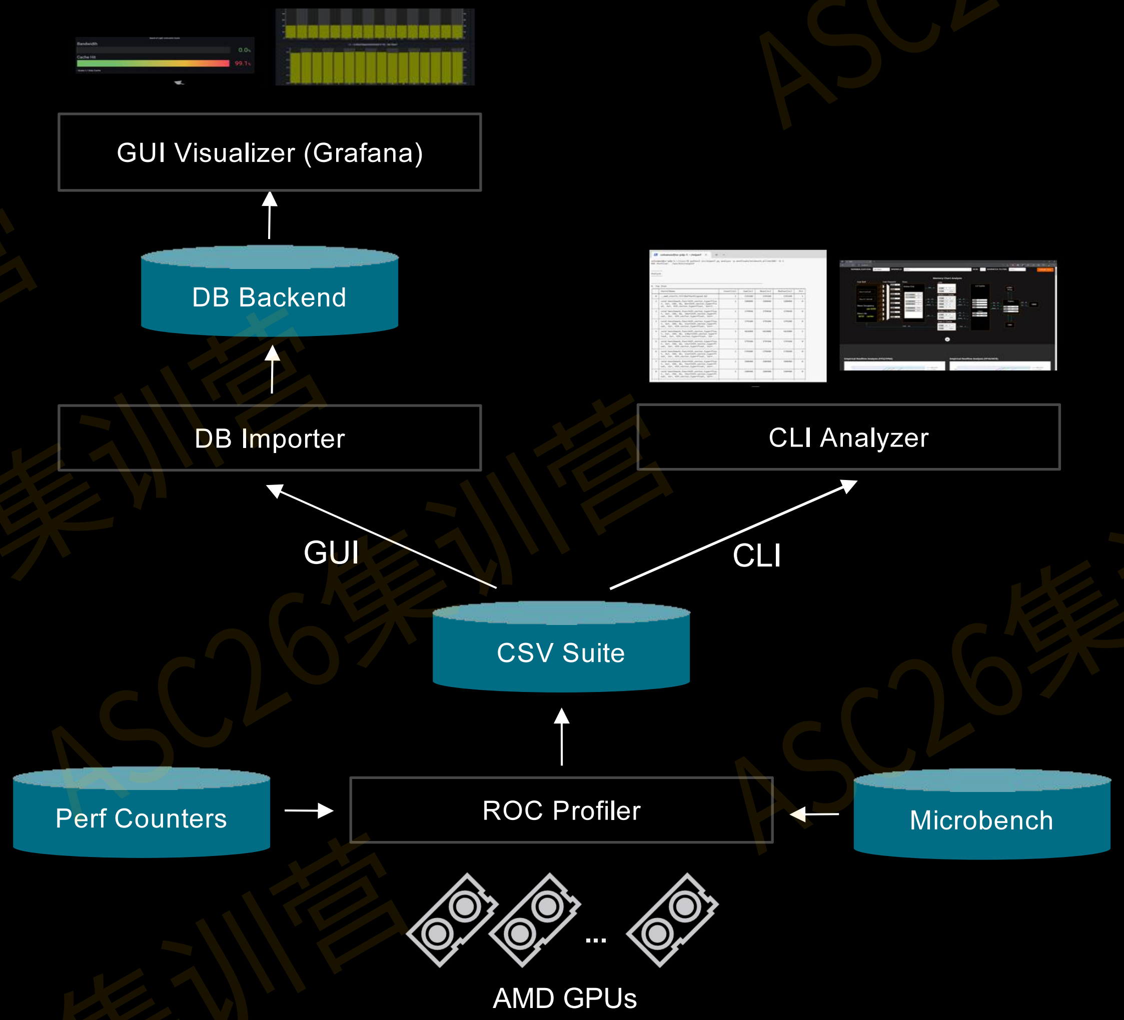
Drilling down into the prefill stage reveals a detailed call stack of the code.
(using ChatGLM as an example.)

AMD
together we advance_


ROCprof Compute Viewer
Import Edit Help
Input Plots Options
Counters Wave States Hotspot Occupancy Kernel Dispatches Compute Unit Utilization
UI Path:
tput_agent_50641_dispatch_200033
Shader SIMD Slot WaveID
GlobalView zoom: 10
WaveView zoom: 10
Iteration 1
WaveView clock range:
33172
128660
Search Next Prev
History
Token Cycle
1 MATRIX 45976
SM0-00
SM1-00
SM2-00
SM3-00
SALU, cycles: 4, clk: 45524
Instructions Global View Summary Explorer
Instruction Hitcount Latency: Sum all Idle
ds_read_b128 v[146:149], v21 offset:896 352 2816
v_mfma_f32_16x16x32_bf16 a[72:75], v[114:117], v[34:37], a[72:75] 352 2816
ds_read_b128 v[82:85], v20 offset:192 352 2816
v_mfma_f32_16x16x32_bf16 a[76:79], v[114:117], v[38:41], a[76:79] 352 2816
ds_read_b128 v[150:153], v21 offset:192 352 2816
s_waitcnt lgkmcnt(11) 352 1408
v_mfma_f32_16x16x32_bf16 a[0:3], v[118:121], v[42:45], a[0:3] 352 1408
ds_read_b128 v[86:89], v20 offset:9408 352 2816
v_mfma_f32_16x16x32_bf16 a[4:7], v[118:121], v[46:49], a[4:7] 352 2816
ds_read_b128 v[90:93], v20 offset:18624 352 2816
v_mfma_f32_16x16x32_bf16 a[8:11], v[118:121], v[50:53], a[8:11] 352 2816
ds_read_b128 v[94:97], v20 offset:27840 352 2816
v_mfma_f32_16x16x32_bf16 a[12:15], v[118:121], v[54:57], a[12:15] 352 2816
ds_read_b128 v[98:101], v20 offset:37056 352 2816
v_mfma_f32_16x16x32_bf16 a[16:19], v[118:121], v[58:61], a[16:19] 352 2816
ds_read_b128 v[154:157], v21 offset:448 352 2816
v_mfma_f32_16x16x32_bf16 a[20:23], v[122:125], v[42:45], a[20:23] 352 2816
ds_read_b128 v[158:161], v21 offset:704 352 2816
v_mfma_f32_16x16x32_bf16 a[24:27], v[122:125], v[46:49], a[24:27] 352 2816
ds_read_b128 v[162:165], v21 offset:960 352 2816
v_mfma_f32_16x16x32_bf16 a[28:31], v[122:125], v[50:53], a[28:31] 352 2816
v_mfma_f32_16x16x32_bf16 a[32:35], v[122:125], v[54:57], a[32:35] 352 5632
v_mfma_f32_16x16x32_bf16 a[36:39], v[122:125], v[58:61], a[36:39] 352 5632
v_mfma_f32_16x16x32_bf16 a[40:43], v[126:129], v[42:45], a[40:43] 352 5632
v_mfma_f32_16x16x32_bf16 a[44:47], v[126:129], v[46:49], a[44:47] 352 5632
s_waitcnt lgkmcnt(0) 352 1408
s_barrier 352 4132
v_mfma_f32_16x16x32_bf16 a[48:51], v[126:129], v[50:53], a[48:51] 352 1408
s_waitcnt vmcnt(17) 352 1408
ds_write_b128 v18, v[166:169] 352 10452
v_mfma_f32_16x16x32_bf16 a[52:55], v[126:129], v[54:57], a[52:55] 352 1408
buffer_load_dwordx4 v[166:169], v0, s[64:67], 0 offen 352 3464
s_waitcnt vmcnt(17) 352 1408
ds_write_b128 v18, v[170:173] offset:4608 352 13376
v_mfma_f32_16x16x32_bf16 a[56:59], v[126:129], v[58:61], a[56:59] 352 1408
buffer_load_dwordx4 v[170:173], v1, s[64:67], 0 offen 352 3168

The Profiling Tools and Visualization – Omniperf

- Core Omniperf profiler
  - Raw performance counters via application using ROCProfiler
  - Hierarchical roofline data is obtained by a set of micro-benchmarks
- Grafana server for Omniperf
  - Database: Raw performance counters are imported into a MongoDB
  - Grafana GUI: It displays the relevant performance metrics and visualization by retrieving the data from database
- Omniperf Standalone GUI Analyzer
  - Omniperf provides a standalone GUI to enable basic performance analysis without the need to import data into a database instance.
- Features
  - Speed-of-Light (SOL)
  - Hardware Block-level SOL Evaluations
  - Roofline Analysis
  - ...



https://rocm.docs.amd.com/projects/omniperf/en/docs-6.2.1/what-is-omniperf.html

Agenda

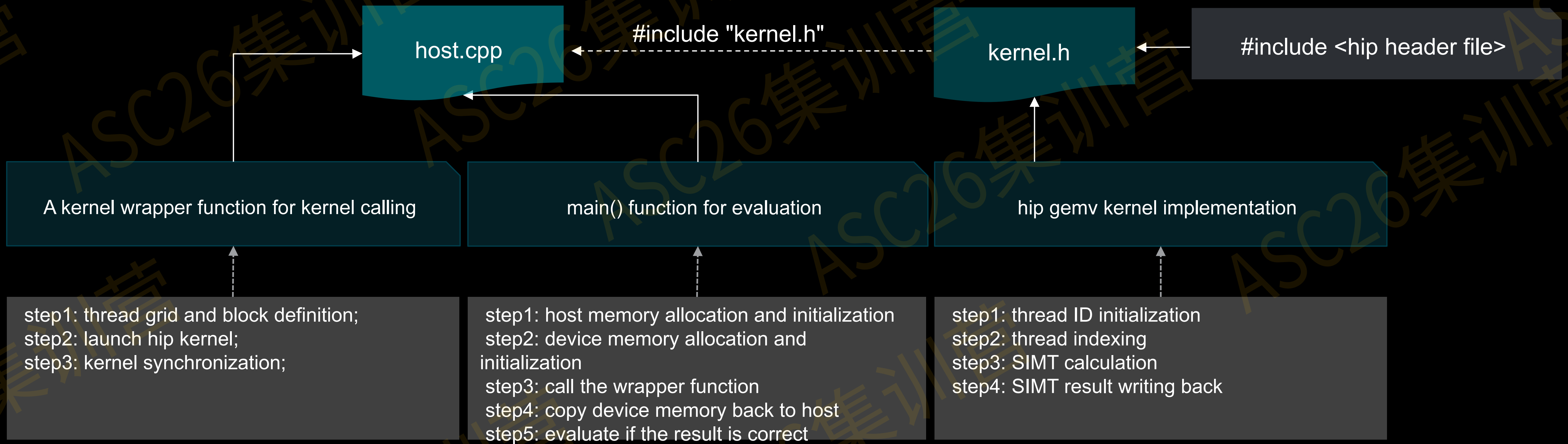
1. Introduction to the AMD ROCm™ Software Stack
2. Transitioning Workloads to AMD GPUs
3. Performance Optimization
   - Optimizing application using popular libraries
   - Profiling the models
   - Adding HIP kernel to implement a custom layer
4. Available Collaterals, Q&A

ROCm Core - Custom HIP GEMV Kernel “hello world” sample

- Given a matrix (M x N), a vector (N x 1), GEMV(matrix, vector) produces an output vector (M x 1)
- GPU kernel (kernel.h) launched from host (host.cpp) explores the GPU compute capability by a single instruction multiple threads (SIMT) design

---

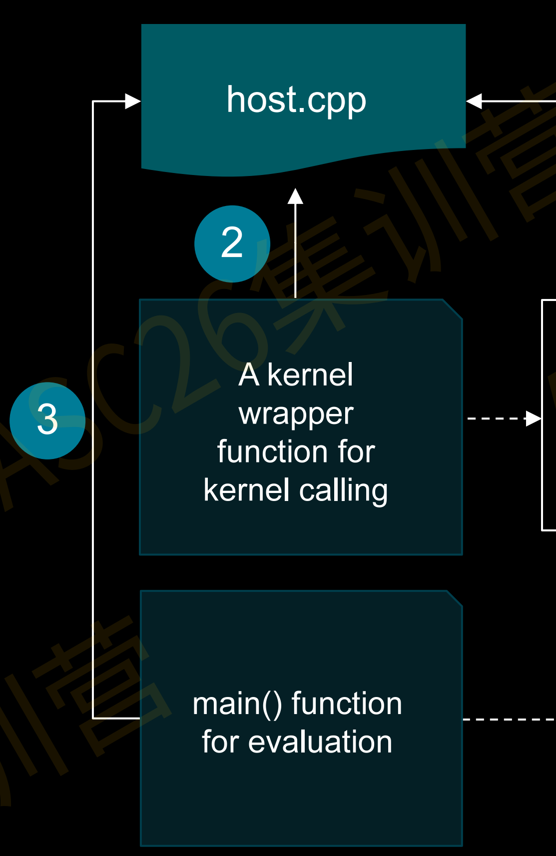
The Implementation Structure of a HIP GEMV Kernel



https://github.com/amd/GenAI-contest/tree/main/03-HIP_LLM_Acceleration/hip_basics

HIP GEMV Host Code Design

- Given a matrix (128 x 4), a vector (4 x 1), GEMV(matrix, vector) produces an output vector (128 x 1),
  - A simple thread parallelism is to employ 128 threads to compute 128 rows in parallel




```
void demo_gemv_v0(float *mat, float *vec, float *res) {

dim3 grid_dim (1, 1);
dim3 block_dim(128, 1);

Kernel_gemv_v0<<<grid_dim, block_dim>>>(mat, vec, res);

hipDeviceSynchronize();
}
```

```
(at::Tensor mat, at::Tensor vec, at::Tensor res)
```

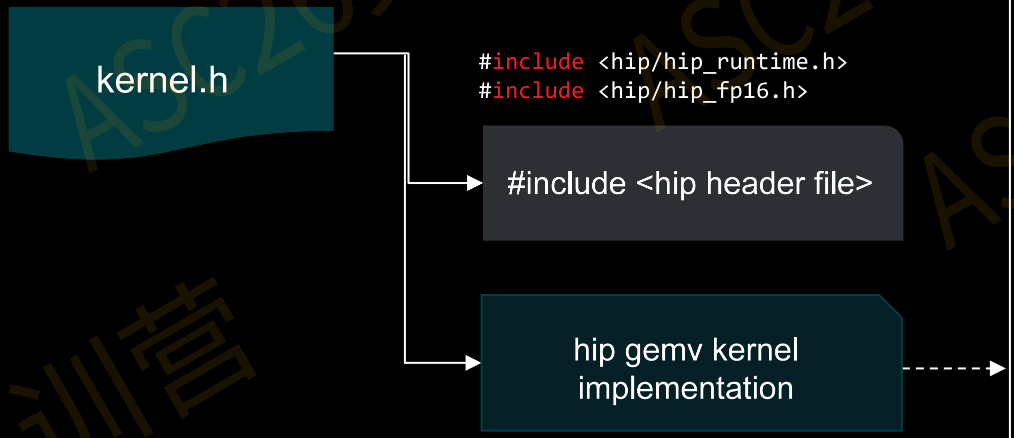
```
reinterpret_cast<half *>(mat.data_ptr<at::Half>()),
reinterpret_cast<half *>(vec.data_ptr<at::Half>()),
reinterpret_cast<half *>(res.data_ptr<at::Half>())
```

```
int main() {

int mat_rows = 128;
int vec_cols = 4;

// Allocate memory on CPU
float* mat = (float*)malloc(sizeof(float) * mat_rows * vec_cols);
float* vec = (float*)malloc(sizeof(float) * vec_cols);
float* res = (float*)malloc(sizeof(float) * mat_rows);

// Fill in some data into mat and vec
for (int i = 0; i< mat_rows * vec_cols; ++i)
    mat[i] = (float)1.f;
for (int i = 0; i< vec_cols; ++i)
    vec[i] = (float)2.f;

// Allocate memory on GPU
float *d_mat, *d_vec, *d_res;
hipMalloc((void **)&d_mat, mat_rows * vec_cols * sizeof(float));
hipMalloc((void **)&d_vec, vec_cols * sizeof(float));
hipMalloc((void **)&d_res, mat_rows * sizeof(float));

// Host to Device
hipMemcpy(d_mat, mat, (mat_rows * vec_cols) * sizeof(float),
hipMemcpyHosttoDevice);
hipMemcpy(d_vec, vec, (vec_cols) * sizeof(float), hipMemcpyHosttoDevice);

// Launch kernel
demo_gemv_v0(d_mat, d_vec, d_res);

// Device to Host
hipMemcpy(res, d_res, (mat_rows) * sizeof(float), hipMemcpyDeviceToHost);

/ /Print result
for (int i=0; i< mat_rows; ++i)
    printf("%f ", res[i]);
}
```

HIP GEMV Kernel Design

$$
\begin{bmatrix} 1,1 & 1,2 & 1,3 & 1,4 \\ 2,1 & 2,2 & 2,3 & 2,4 \\ 3,1 & 3,2 & 3,3 & 3,4 \\ & \cdots\cdots & & \\ 128,1 & 128,2 & 128,3 & 128,4 \end{bmatrix}
\times
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}
=
\begin{matrix} 1,1 \; 1 & + & 1,2 \; 2 & + & 1,3 \; 3 & + & 1,4 \; 4 \\ 2,1 \; 1 & + & 2,2 \; 2 & + & 2,3 \; 3 & + & 2,4 \; 4 \\ 3,1 \; 1 & + & 3,2 \; 2 & + & 3,3 \; 3 & + & 3,4 \; 4 \\ & & & \cdots\cdots & & & \\ 128,1 \; 1 & + & 128,2 \; 2 & + & 128,3 \; 3 & + & 128,4 \; 4 \end{matrix}
=
\begin{matrix} 1 \\ 2 \\ 3 \\ \\ 128 \end{matrix}
$$

Matrix　　Vector




```
__global__ void kernel_gemv_v0(float *mat, float *vec, float* res) {
    unsinged int tid = threadIdx.x;
    unsigned int row = tid;
    unsigned int start_idx = 4 * row;

    float mat_h0 = mat[start_idx];
    float mat_h1 = mat[start_idx + 1];
    float mat_h2 = mat[start_idx + 2];
    float mat_h3 = mat[start_idx + 3];

    float vec_h0 = vec[0];
    float vec_h1 = vec[1];
    float vec_h2 = vec[2];
    float vec_h3 = vec[3];

    float sum = 0.0;
    sum += (mat_h0) * (vec_h0);
    sum += (mat_h1) * (vec_h1);
    sum += (mat_h2) * (vec_h2);
    sum += (mat_h3) * (vec_h3);

    res[row] = sum;
}
```

```
(half *mat, half *vec, half *res)

  float mat_h0 = mat[start_idx];
  float mat_h1 = mat[start_idx + 1];
  float mat_h2 = mat[start_idx + 2];
  float mat_h3 = mat[start_idx + 3];

  float vec_h0 = vec[0];
  float vec_h1 = vec[1];
  float vec_h2 = vec[2];
  float vec_h3 = vec[3];

  float sum = 0.0;
  sum += __half2float(mat_h0) * __half2float(vec_h0);
  sum += __half2float(mat_h1) * __half2float(vec_h1);
  sum += __half2float(mat_h2) * __half2float(vec_h2);
  sum += __half2float(mat_h3) * __half2float(vec_h3);

  res[row] = __half2float sum;
```

```
hipcc --offload-arch=gfx1100 host.cpp -o gemv_v0
./gemv_v0
```


AMD
together we advance_

Performance Optimization – Instruction Throughput

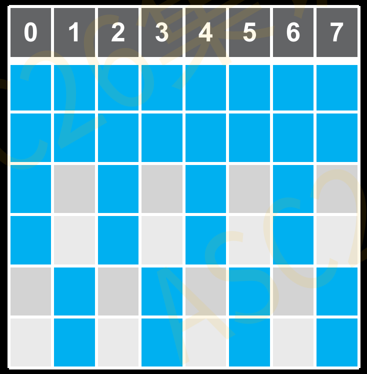
Control Flow & Divergence

- A wave executes in lockstep. If threads in a wavefront take different branches of an if/else, the GPU executes both paths, masking off threads, leading to divergence and wasted cycles.

Example:

```
int i = blockIdx.x * blockDim.x + threadIdx.x;

if (i % 2 == 0)
{
    // half the threads do this
    out[i] = in[i] * 2.0f;
}
else
{
    // half the threads do this
    out[i] = in[i] * 3.0f;
}
```

8 Threads (for simplicity)



Use Efficient Operations

- Some arithmetic operations are more expensive than others. For example, multiplication is typically faster than division.

Trade Precision for Speed

- *Consider using single-precision arithmetic instead of double-precision if possible.*

Leverage Intrinsic Functions

- Intrinsic functions are predefined functions available in HIP that can often be executed faster than equivalent arithmetic operations.

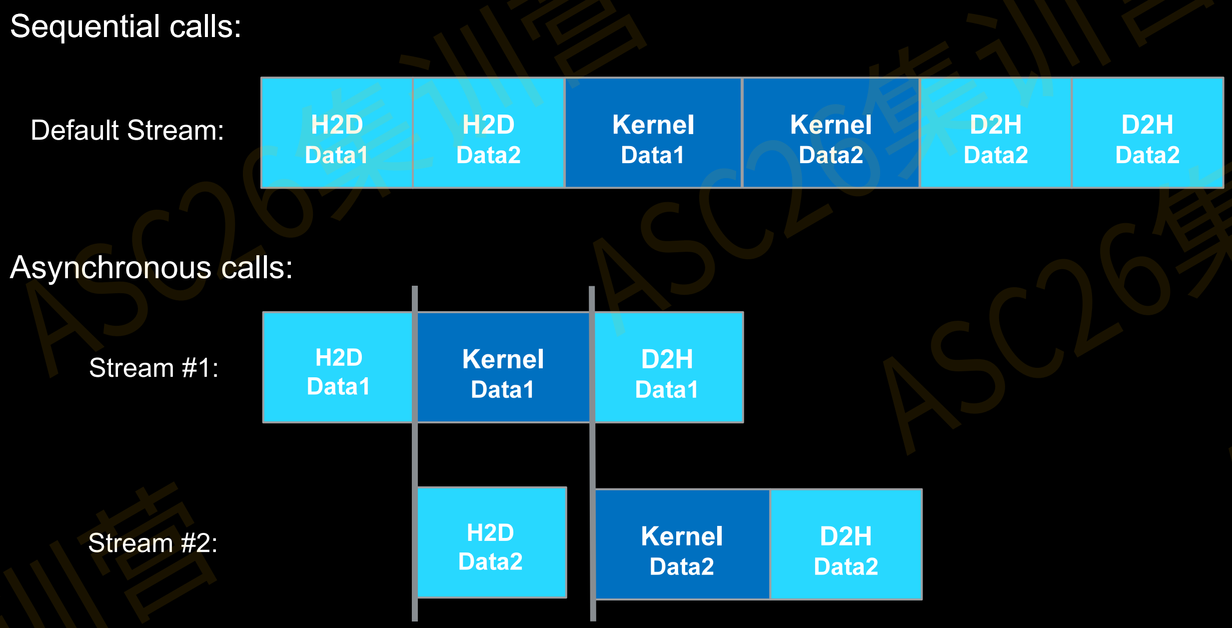
https://rocm.docs.amd.com/projects/HIP/en/latest/how-to/performance_guidelines.html

AMD
together we advance_

Performance Optimization – Parallel Execution

Application Level

- Use asynchronous calls and streams to overlap host/device work. Send serial work to CPU and parallel work to GPU.



Device Level

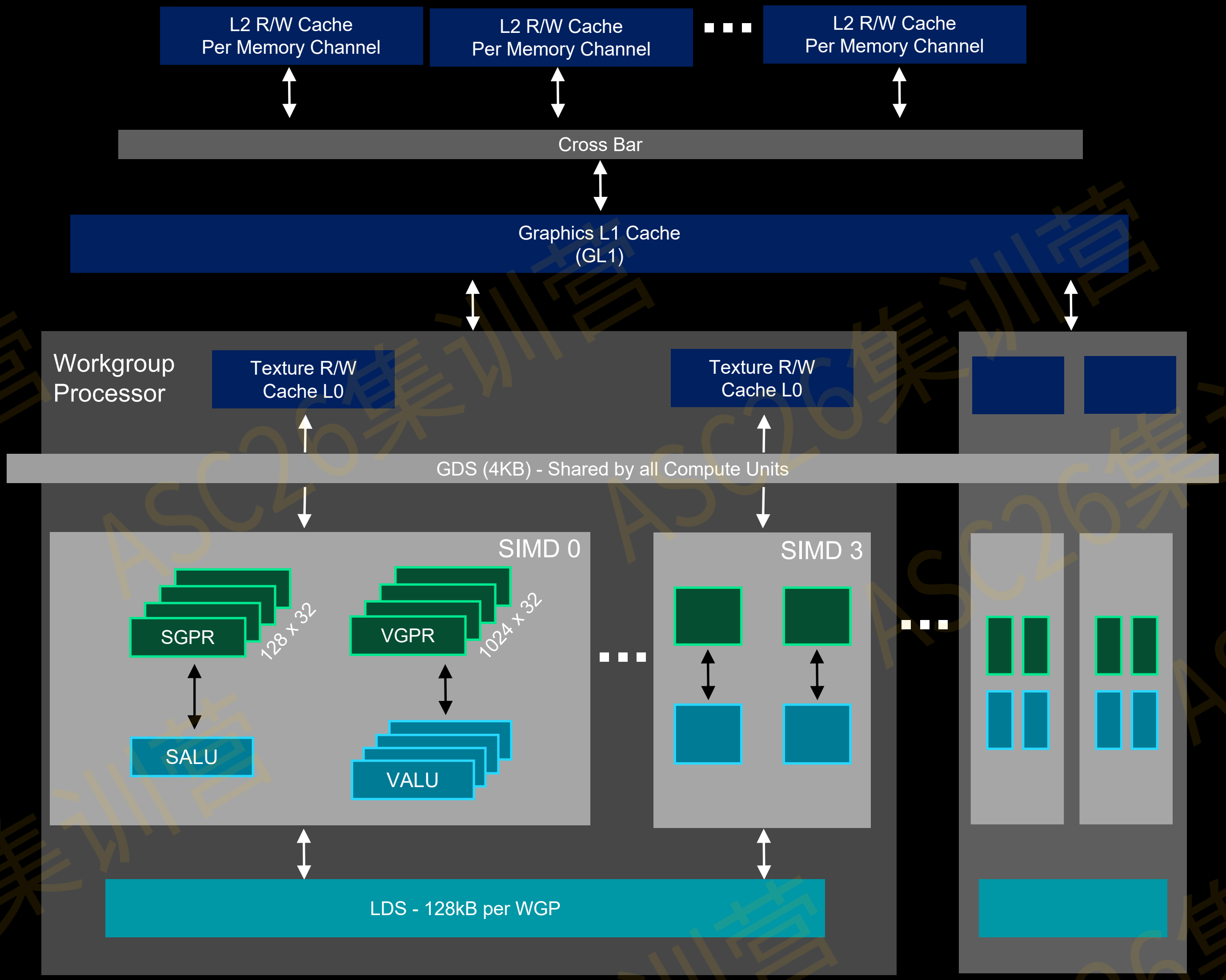
- Maximize utilization by executing enough kernels concurrently while avoiding resource contention.

Multiprocessor Level

- At its best every clock cycle has an instruction from a warp is ready for execution. This could either be another independent instruction of the same warp or an instruction of another warp.

https://rocm.docs.amd.com/projects/HIP/en/latest/how-to/performance_guidelines.html
https://rocm.docs.amd.com/projects/HIP/en/latest/how-to/hip_runtime_api/asynchronous.html

Performance Optimization – Memory Throughput



Local Data Share (LDS)

- On- chip shared memory for fast communication and data reuse, often used as a software cache or for cooperative access to off- chip memory.

Global Data Share (GDS)

- Small on- chip memory shared across all WGPs and waves of a kernel. It provides hardware support of append/consume patterns and control data for compute kernels, reduction operations, etc.

Device Memory Hierarchy (L2 → L1 → L0)

- Multiple L2 cache channels feed read- only L1 and per- WGP L0 caches for off- chip memory accesses. Specialized cache- less load instructions allow direct device memory reads when needed, while caches improve reuse and aggregate scattered accesses.

https://docs.amd.com/v/u/en-US/rdna3-shader-instruction-set-architecture-feb-2023_0

Performance Optimization – Memory Throughput

Local Data Share (LDS)

Bank Conflict: It occurs when multiple threads in the same wave access the same bank in shared memory. In this case, accesses get serialized, leading to inferior performance.

$$\text{bank} = \left(\frac{\textit{address in bytes}}{4}\right) \bmod 32$$

(Sample: For AMD GCN architecture)

Optimizations:

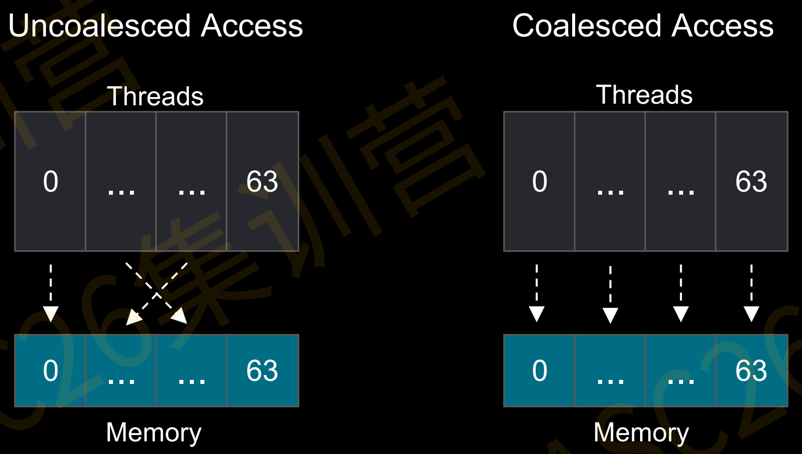
- Padding: Change the bank mapping

```
__shared__ float tile[32][33];
```

- XOR Preshuffle: Permute the column indices for each row using XOR.
- Use CK Tile abstractions: They automatically handles bank conflict avoidance.
- Consider access patterns: Design algorithms with bank-friendly patterns.

Device Memory

Coalescing: A memory access pattern is coalesced when consecutive threads access consecutive addresses. The hardware can combine them into fewer and wider transactions.



Optimizations:

- Avoid strided access: Array of Structures (AoS) → Structures of Arrays (SoA).
- Align or pad data: Achieve reading/writing contiguous segments.

https://rocm.docs.amd.com/projects/HIP/en/latest/understand/programming_model.html
https://rocm.docs.amd.com/projects/composable_kernel/en/latest/conceptual/ck_tile/hardware/lds_bank_conflicts.html

Agenda

1. Introduction to the AMD ROCm™ Software Stack
2. Transitioning Workloads to AMD GPUs
3. Performance Optimization
   - Optimizing application using popular libraries
   - Profiling the models
   - Adding HIP kernel to implement a custom layer
4. Available Collaterals, Q&A

AMD ROCm™ Software Developer Hub

Initiative to Educate and Increase ROCm™ Software Stack User Base and Adoption

High-level Overview

Familiarize yourself with the ecosystem
General introduction of ROCm Software

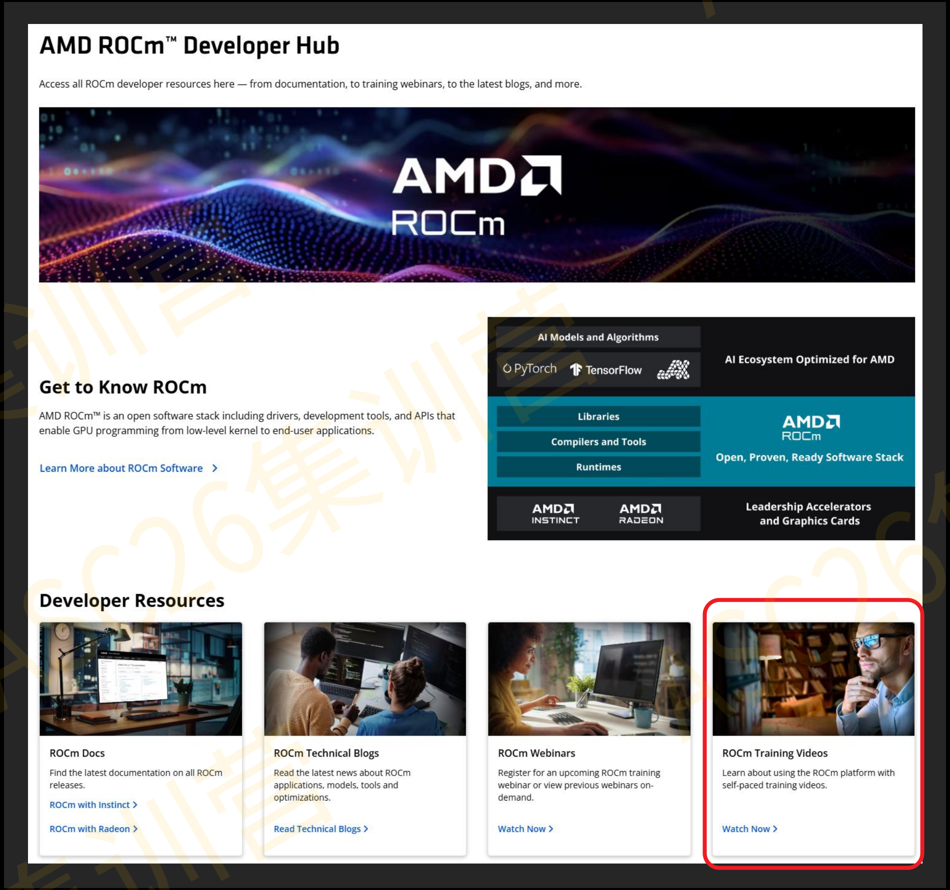
Increase Understanding

Attend ROCm webinars
View one of the many training videos

Build Comprehension

Purchase ROCm textbook
See the latest news on ROCm blog

ROCm Developer Hub

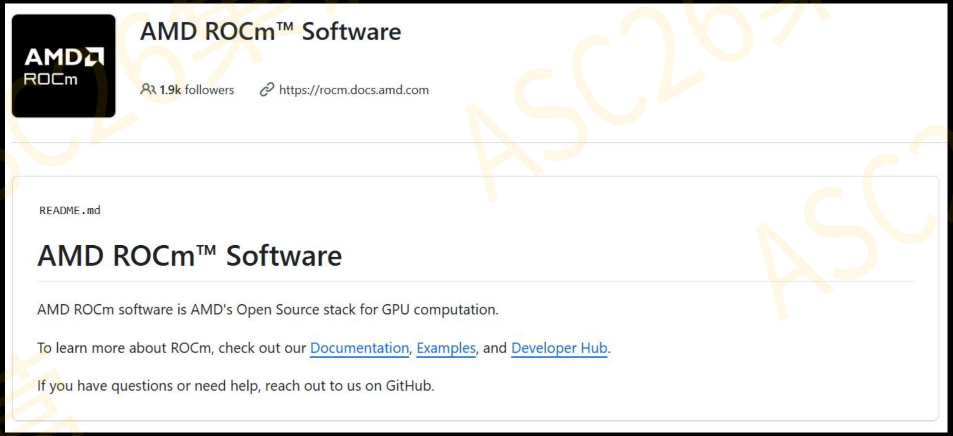


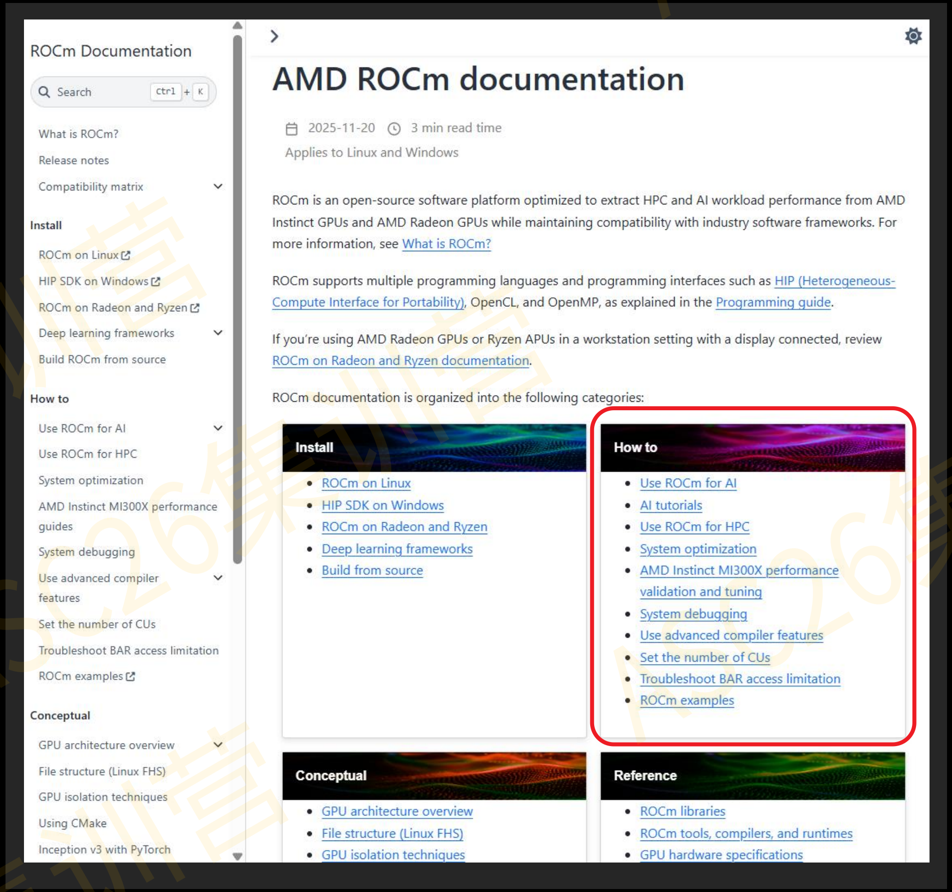
AMD ROCm™ Documentation & Github Repository

Playground for Professional Developers

Dive Deeper

Refer to ROCm documentation
Make contributions to all major components on Github





ROCm Github Organization

AMD together we advance_

AMD

Join Us

Registration Page

AMD Dev Assistant

AMD
AI Developer Program

https://account.amd.com/en/forms/registration/ai-dev-program.html

Disclaimer and Attributions

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

AMD
together we advance_

AMD