AMD GPU Introduction

PP 2025 LAB5

國立清華大學叢集電腦競賽團隊
Student Cluster Competition Team of NTHU

Outline

Basic Knowledge of AMD GPU

ROCm API

ROCm Profiler

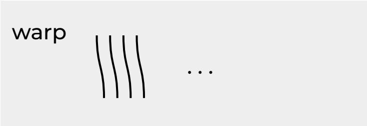
Basic Knowledge of AMD GPU

What is Thread, Block, Grid?

Thread

The basic unit of work executed by the GPU.

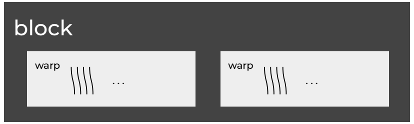
Threads are group in **warp**, threads in the same warp must execute the same instruction simultaneously.



Block

Block(thread block) is a group of threads that can cooperate and communicate with each other.

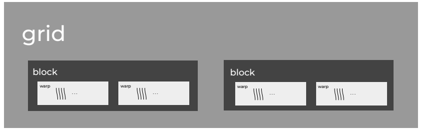
All threads in a single block are mapped to and executed on the same **CU**(compute unit).



Grid

The highest-level organizational structure for threads executing a kernel function, representing a single launch of that kernel.

A grid consists of multiple thread blocks, which execute the kernel concurrently.

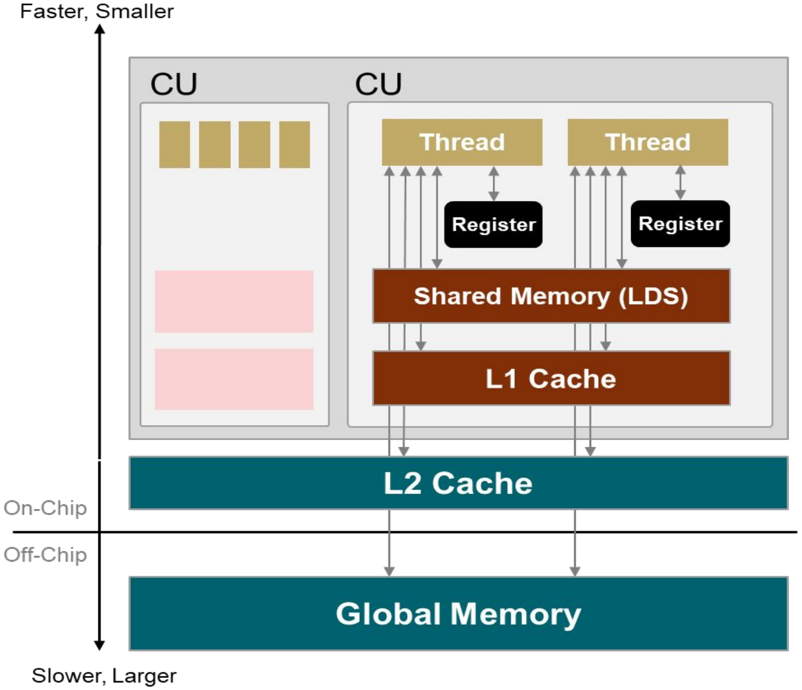


What is Warp and Cu?

- Warp
  - 32/64 threads per warp in amd gpu
  - threads in same warp execute the same instruction simultaneously
  - avoid warp divergence is important

- Cu(Compute Unit)
  - each Cu can handle multiple thread blocks
  - but holds only limited number of warps
  - Cu shares the same space of L1 cache and shared memory

Memory Hierarchy



Thread Wide

This is the lowest and most confined level of the hierarchy.

- Data is private to a single thread and is stored in registers.
- A thread cannot see or access the registers of other threads.
- Actions of one thread have no effect or visibility to any other thread at this level.

Block Wide

A block is a group of threads that execute together on the same CU.

- Threads in the block can cooperate and share data.
- Achieved through shared memory and synchronization.

Device Wide

This is the highest level of the hierarchy and includes all threads and all blocks running on the entire GPU.

- All threads and blocks can access global memory.
- Data sharing through L2 cache and global memory.

Some Different with Cuda

- NVIDIA GPUs have a fixed warp size of 32 threads.
- AMD GPUs typically use a warp size of 64 threads, though some architectures are 32.
- NVIDIA allows the shared-memory size per SM to be configured dynamically, whereas AMD’s is fixed at 64 KB per Cu.

ROCm API

HIP Compiler and Header

Hip compiler: hipcc

Hip runtime header: include <hip/hip_runtime.h>

Compile hip code: hipcc -o <xxx> <xxx.hip>

SMI

rocm-smi

```
amd@mi100-1:~$ rocm-smi


======================================= ROCm System Management Interface =======================================
================================================= Concise Info =================================================
Device  Node  IDs              Temp    Power  Partitions          SCLK    MCLK     Fan     Perf  PwrCap  VRAM%  GPU%
              (DID,     GUID)  (Edge)  (Avg)  (Mem, Compute, ID)
================================================================================================================
0       11    0x738c,   23480  32.0°C  34.0W  N/A, N/A, 0         300Mhz  1200Mhz  20.78%  auto  290.0W  0%     0%
1       10    0x738c,   64802  31.0°C  34.0W  N/A, N/A, 0         300Mhz  1200Mhz  20.78%  auto  290.0W  0%     0%
2       9     0x738c,   59802  31.0°C  34.0W  N/A, N/A, 0         300Mhz  1200Mhz  20.78%  auto  290.0W  0%     0%
3       8     0x738c,   44650  31.0°C  34.0W  N/A, N/A, 0         300Mhz  1200Mhz  20.78%  auto  290.0W  1%     0%
4       13    0x738c,   30589  33.0°C  39.0W  N/A, N/A, 0         300Mhz  1200Mhz  20.78%  auto  290.0W  0%     0%
5       12    0x738c,   15436  31.0°C  36.0W  N/A, N/A, 0         300Mhz  1200Mhz  20.78%  auto  290.0W  0%     0%
================================================================================================================
============================================= End of ROCm SMI Log ==============================================
```

GPU Info

rocminfo

```
Workgroup Max Size:      1024(0x400)
Workgroup Max Size per Dimension:
  x                        1024(0x400)
  y                        1024(0x400)
  z                        1024(0x400)
Grid Max Size:           4294967295(0xffffffff)
Grid Max Size per Dimension:
  x                        4294967295(0xffffffff)
  y                        4294967295(0xffffffff)
  z                        4294967295(0xffffffff)
FBarrier Max Size:       32
```

Allocate Device Global Memory

- hipError_t **hipMalloc**(void** devPtr, size_t size);
  - devPtr – pointer to a pointer that will receive the device address.
  - size – number of bytes to allocate in GPU global memory.
  - Returns a hipError_t status code.
- hipError_t **hipMemcpy** (void* dst, const void* src, size_t size, hipMemcpyKind kind);
  - dst / src – destination and source pointers.
  - size – number of bytes to copy.
  - kind – direction of the copy.
  - Returns a hipError_t status code.

Don’t forget to **hipFree**!

Allocate Device Global Memory

Device Pointer

Host Pointer

```
void solve(const float* logits, const int* true_labels, float* loss, int N, int C) {
    float *d_logits, *d_loss;
    int *d_true_labels;

    hipMalloc(&d_logits, N * C * sizeof(float));
    hipMalloc(&d_true_labels, N * sizeof(int));
    hipMalloc(&d_loss, sizeof(float));

    hipMemcpy(d_logits, logits, N * C * sizeof(float), hipMemcpyHostToDevice);
    hipMemcpy(d_true_labels, true_labels, N * sizeof(int), hipMemcpyHostToDevice);
    hipMemcpy(d_loss, loss, sizeof(float), hipMemcpyHostToDevice);
```

Memory copy from host to device

```
hipMemcpy(loss, d_loss, sizeof(float), hipMemcpyDeviceToHost);
```

Memory copy from device to host

Launch Kernel Function

hipError_t **hipLaunchKernelGGL**(

```
hipError_t hipLaunchKernelGGL(
    kernelFunc,                  // __global__ kernel name
    dim3 gridDim,                // grid size (x,y,z)
    dim3 blockDim,               // block size (x,y,z)
    size_t sharedMemBytes,       // optional dynamic shared memory (bytes)
    hipStream_t stream,          // stream handle (0 for default)
    kernelArgs...                // arguments to the kernel
);
```

OR

necessary optional

```
kernelName<<<gridDim, blockDim, sharedMemBytes, stream>>>(arg1, arg2, ...);
```

Other APIs

Most HIP APIs work the same as CUDA—just press Ctrl + F and replace cuda with hip.

ROCm Profiler

ROCProf Compute

rocprof-compute profile -n <xxx> -- ./<xxx> <args>

rocprof-compute analyze -p workloads/<xxx>/MI100/

- profile - run and record profiling data
  - --name (-n) : The profiler will create a folder like workloads/<xxx>/
  - --kernel (-k) : profile specific kernel
- analyze - open and inspect data from a previous run
  - --path (-p) : inspect data generate by profile
- database - import/export data for Grafana or other database-backed viewers

https://rocm.docs.amd.com/projects/rocprofiler-compute/en/latest/how-to/use.html

ROCProf Compute

0.1 Top Kernels

	Kernel_Name	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
0	phase3(int, int*, int) [clone .kd]	157.00	422091878.00	2688483.30	2688487.00	96.16
1	phase2(int, int*, int) [clone .kd]	157.00	11097161.00	70682.55	71200.00	2.53
2	phase1(int, int*, int) [clone .kd]	157.00	5131208.00	32682.85	29920.00	1.17
3	initializeDeviceDist(int*, int) [clone .kd]	1.00	601441.00	601441.00	601441.00	0.14
4	updateDeviceDist(int*, int const*, int, int) [clone .kd]	1.00	20480.00	20480.00	20480.00	0.00

ROCProf Compute

gpu_model	MI100
gpu_arch	gfx908
gpu_l1	16
gpu_l2	8192
cu_per_gpu	120
simd_per_cu	4
se_per_gpu	8
wave_size	64
workgroup_max_size	1024

Size of L1 Cache(KB) per CU

Size of L2 Cache(KB) in whole GPU

Number of Cus in whole GPU

wave = warp

workgroup = threadblock

ROCProf Compute

rocprof-compute analyze -p workloads/<baseline>/MI100/ \
-p workloads/<compare>/MI100/

```
0. Top Stats
0.1 Top Kernels
```

	Kernel_Name	Count	Count	Abs Diff	Sum(ns)	Sum(ns)
0	phase3(int, int*, int) [clone .kd]	157.00	235.0 (49.68%)	78.00	421864145.00	941825411.0 (123.25%)
1	phase2(int, int*, int) [clone .kd]	157.00	235.0 (49.68%)	78.00	11090903.00	17447075.0 (57.31%)
2	phase1(int, int*, int) [clone .kd]	157.00	235.0 (49.68%)	78.00	5124330.00	7125766.0 (39.06%)
3	initializeDeviceDist(int*, int) [clone .kd]	1.00	1.0 (0.0%)	0.00	603361.00	1368001.0 (126.73%)
4	updateDeviceDist(int*, int const*, int, int) [clone .kd]	1.00	1.0 (0.0%)	0.00	20480.00	8000.0 (-60.94%)

ROCProfv2

rocprofv2 -i <xxx(metric.txt)> -o <xxx> -d <xxx> ./<xxx> <args>

- --input (-i) : specify a input file for metric profiling
- metric.txt : your metric input file
  - Use **rocprofv2 --list-counters** to check all the usable metrics
  - pmc : The rows in the text file beginning with pmc: are the group of metrics you are interested in collecting
  - **Example metric file:** cat > metrics.txt <<'EOF' pmc: SQ_WAVES EOF
- --output-file-name (-o) : profile output csv filename
- --output-directory (-d) : profile output directory
- You will get a result_<xxx>.csv, download it and check the profile result

ROCm Systems Profiler

rocprof-sys-run --trace -- ./<xxx> <args>

Download this : `perfetto-trace-1256288.proto`

Open it on perfetto : Open trace file

You will get :

SPEC

THE END

😐😐😐