

# GPU Computing

Weijie Zhao

09/08/2022

# GPU Architecture

- 108 Streaming Multi-processor (SM)
- 40 GB High-Bandwidth Memory (HBM)
  - 1555 GB/sec
  - 6912 FP32 CUDA cores
- 432 Tensor Cores, TensorFloat-32(TF32) Dense Tensor (156 TFLOPs)
- 192KB \* 108 L1 Cache
- 40960 KB L2 Cache

# GPU Scheduling

- SIMT (Single Instruction Multiple Thread)
- Warp
- Dangerous to implement critical section (Pre Volta)
- Independent Thread Scheduling (After Volta)

# CUDA Programming

- Kernel
- Grid
- Block
- Thread
- Warp
- Host Memory
- Device Memory
  - Global Memory
  - Shared Memory

# CUDA Programming

```
__global__
void saxpy(int n, float a, float *x, float *y){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a * x[i] + y[i];
}
```

# CUDA Programming

```
__global__
void saxpy(int n, float a, float *x, float *y){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a * x[i] + y[i];
}
saxpy<<<nB, nT>>>(n, a, x, y);
```

# CUDA Programming

```
__global__
void saxpy(int n, float a, float *x, float *y){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a * x[i] + y[i];
}
saxpy<<<nB, nT>>>(n, a, x, y);
```



device memory

# CUDA Programming

```
__global__  
void saxpy(int n, float a, float *x, float *y){  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
    if (i < n)  
        y[i] = a * x[i] + y[i];  
}  
saxpy<<<nB, nT>>>(n, a, x, y);
```

The diagram illustrates the memory access pattern of the SAXPY kernel. It shows two parallel red arrows originating from the variable 'y' in the code. One arrow points from 'y' to 'y[i]', representing the reading of the current element from device memory. The other arrow points from 'y' to 'y[i]' via the assignment statement, representing the writing of the updated value back to device memory.

# CUDA Programming

```
__global__
void saxpy(int n, float a, float *x, float *y){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a * x[i] + y[i];
}
saxpy<<<nB, nT>>>(n, a, x, y);
```

device memory

float\* x;  
cudaMalloc(&x, n \* sizeof(float));

cudaError\_t cudaMalloc ( void\*\* devPtr, size\_t size )

# Host Memory vs. Device Memory

- `cudaMalloc`, `cudaFree`
- `cudaError_t cudaMemcpy( void* dst, const void* src, size_t count, cudaMemcpyKind kind )`
  - `cudaMemcpyHostToHost = 0`
    - Host -> Host
  - `cudaMemcpyHostToDevice = 1`
    - Host -> Device
  - `cudaMemcpyDeviceToHost = 2`
    - Device -> Host
  - `cudaMemcpyDeviceToDevice = 3`
    - Device -> Device
  - `cudaMemcpyDefault = 4`
    - Direction of the transfer is inferred from the pointer values. Requires unified virtual addressing

# CUDA Compilation

- nvcc a.cu -o a.out -O3 -Xptxas -O3 -arch=native
- cuda-gdb
  - -g -G (without optimizations)
  - info cuda threads
  - cuda thread 0
- cuda-memcheck
- nvprof
  - nvvp

# Scan

- Inclusive scan
- Exclusive scan
- Naïve scan
- Work-efficient scan