

# GPU Computing

Weijie Zhao

09/19/2023

# 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



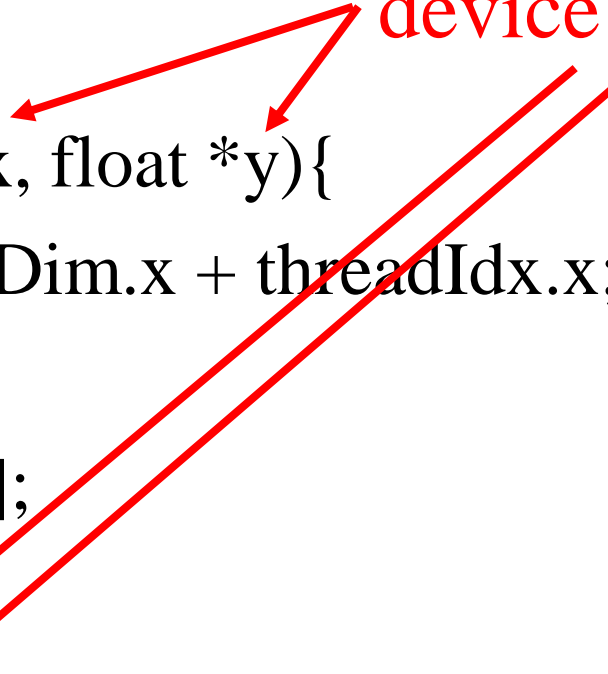
# CUDA Programming

```
__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);
```

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

```
__global__ void reduce(float *g_odata, float *g_idata, int n) {
    extern __shared__ float temp[]; // allocated on invocation
    int thid = threadIdx.x; int offset = 1;
    temp[2*thid] = g_idata[2*thid]; // load input into shared memory
    temp[2*thid+1] = g_idata[2*thid+1];
    for (int d = n>>1; d > 0; d >>= 1){ // build sum in place up the tree
        __syncthreads();
        if (thid < d) {
            int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;
            temp[bi] += temp[ai];
        }
        offset *= 2;
    }
    __syncthreads();
    if (thid == 0)
        *g_odata = temp[n-1];
}
```

```

__global__ void reduce(float *g_odata, float *g_idata, int n) {
extern __shared__ float temp[]; // allocated on invocation
int thid = threadIdx.x; int offset = 1;
temp[2*thid] = g_idata[2*thid]; // load input into shared memory
temp[2*thid+1] = g_idata[2*thid+1];
for (int d = n>>1; d > 0; d >>= 1){ // build sum in place up the tree
    __syncthreads();
    if (thid < d) {
        int ai = offset*(2*thid+1)-1;
        int bi = offset*(2*thid+2)-1;
        temp[bi] += temp[ai];
    }
    offset *= 2;
}
__syncthreads();
if (thid == 0)
    *g_odata = temp[n-1];
}

```

Dynamic shared  
memory allocation

reduce<<<1,nT,n>>>(d\_out,d\_in,n)

Shared memory size per block

Static:

\_\_shared\_\_ float temp[128];

```

__global__ void reduce(float *g_odata, float *g_idata, int n) {
extern __shared__ float temp[]; // allocated on invocation
int thid = threadIdx.x; int offset = 1;
temp[2*thid] = g_idata[2*thid]; // load input into shared memory
temp[2*thid+1] = g_idata[2*thid+1];
for (int d = n>>1; d > 0; d >>= 1){ // build sum in place up the tree
__syncthreads();
if (thid < d) {
int ai = offset*(2*thid+1)-1;
int bi = offset*(2*thid+2)-1;
temp[bi] += temp[ai];
}
offset *= 2;
}
__syncthreads();
if (thid == 0)
*g_odata = temp[n-1];
}

```

Dynamic shared  
memory allocation

$\text{reduce}\langle\langle\langle 1, nT, n \rangle\rangle\rangle(d\_out, d\_in, n)$

Shared memory size per block

Static:

`__shared__ float temp[128];`

# Device/Host Synchronization

```
reduce<<<1,nT,n>>>(d_sum,d_array,n);
```

```
for i = 0 to logn do
```

```
    sweep_down<<<1,nT,n>>>(d_array,n);
```

```
printf(“finished\n”);
```

```
cudaMemcpy(h_array,d_array,cudaMemcpyDeviceToHost);
```

```
printf(...);
```



# Device/Host Synchronization

```
reduce<<<1,nT,n>>>(d_sum,d_array,n);
```

```
for i = 0 to logn do
```

```
    sweep_down<<<1,nT,n>>>(d_array,n);
```

```
cudaDeviceSynchronize();
```

```
printf("finished\n");
```

```
cudaMemcpy(h_array,d_array,cudaMemcpyDeviceToHost);
```

```
printf(...);
```



**Implicit synchronization**

# CUDA Kernel Launch

- `kernel_name<<<nB,nT,shared_memory_size,stream>>>(…)`
- `cudaStream_t stream`
- `cudaStreamCreate(&stream)`
- `cudaMemcpyAsync(dst,src,size,stream)`
- `cudaStreamSynchronize(stream)`
- Default stream: 0

# Multiple GPU Support

- `CUDA_VISIBLE_DEVICES`
- `cudaError_t cudaSetDevice ( int device )`
- `__host__ __device__ cudaError_t cudaMalloc ( void** devPtr, size_t size )`
- `__host__ cudaError_t cudaMemcpyPeer ( void* dst, int dstDevice, const void* src, int srcDevice, size_t count )`
- `__host__ cudaError_t cudaMemcpyPeerAsync ( void* dst, int dstDevice, const void* src, int srcDevice, size_t count, cudaStream_t stream = 0 )`