# GPU Computing

Weijie Zhao

02/08/2023

# HW1 Review

- 21/25 submissions
- 9/21 correct solutions
- Fastest solution:

| | | |
|---|---|---|
| Mahendra Singh Thapa | 192.56s | merge sort with std::sort for 1m elements |

- Runner-ups:

| | | |
|---|---|---|
| Zohair Raza Hassan | 203.93s | split into 24 pieces then std::sort |
| Pujan Thapa | 210.06s | hand-written qsort with omp task |
| Ye Zheng | 212.17s | counting sort with atomic add |

- Solutions no slower than 385.12s will get 15 pts

```
xxxtargzlog 8 245.64 [0.01, 0.01, 0.02, 0.04, 0.04, 0.04, 0.07, 120.0, 120.0, 5.41] [8, 9]
```

- Random seed for generator 12356789
- All grades will be finalized at the end of 2/12

## 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

reduce<<<1,nT,n>>>(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(…);
```

$\xleftarrow{}$ 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 )