Day 2:

- GPU Structure *and* the Problem of Branch Divergence
- Implementation the reduction algorithm on GPU

Dr. Tuan-Tu Tran
trant@uni-mainz.de

Part 1: GPU Structure *and* the Problem of Branch Divergence

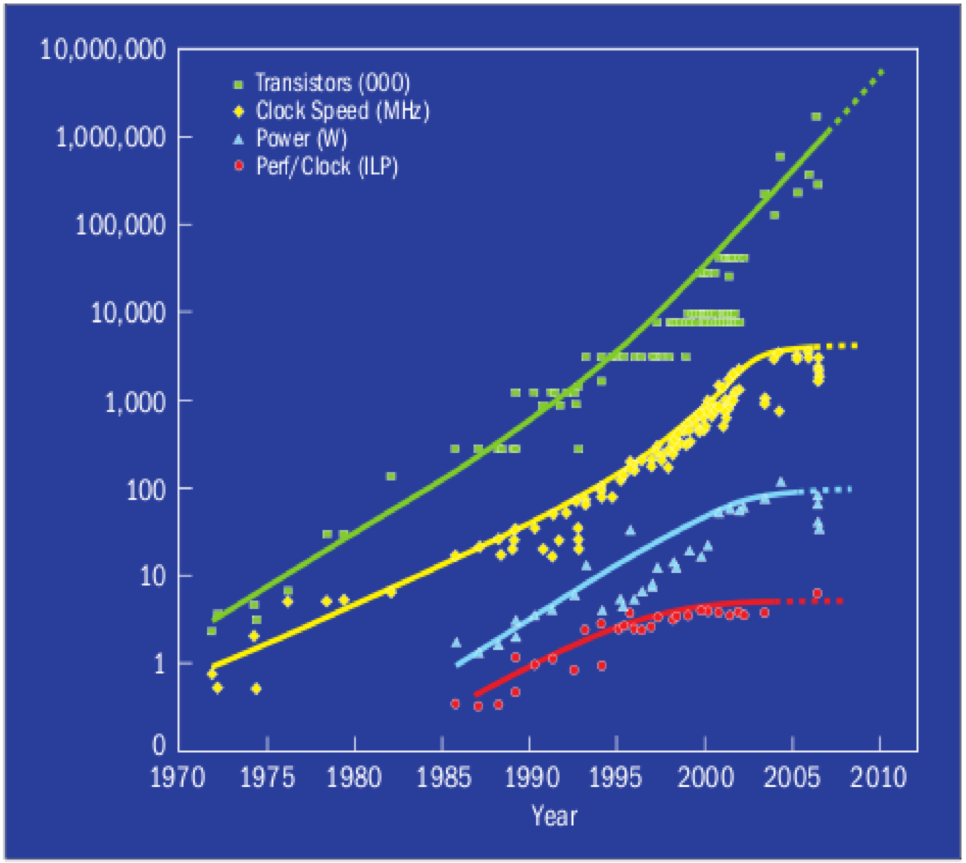
Why multi/many-core?

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Computing Power:

- Number of transitors on a chip: has doubled every two years, since 1965
- Frequency : has not increased anymore, since the beginning of 2000s

➔ Performance increased by integrating multiple “cores” on a chip.



(image source: [3])

Multicore and Manycore

- Meaning: multi- ~ many-
- Actually:
  - Multicore CPU: 2 cores, 4 cores, 8 cores
  - Manycore GPU: 100 to 1000 cores

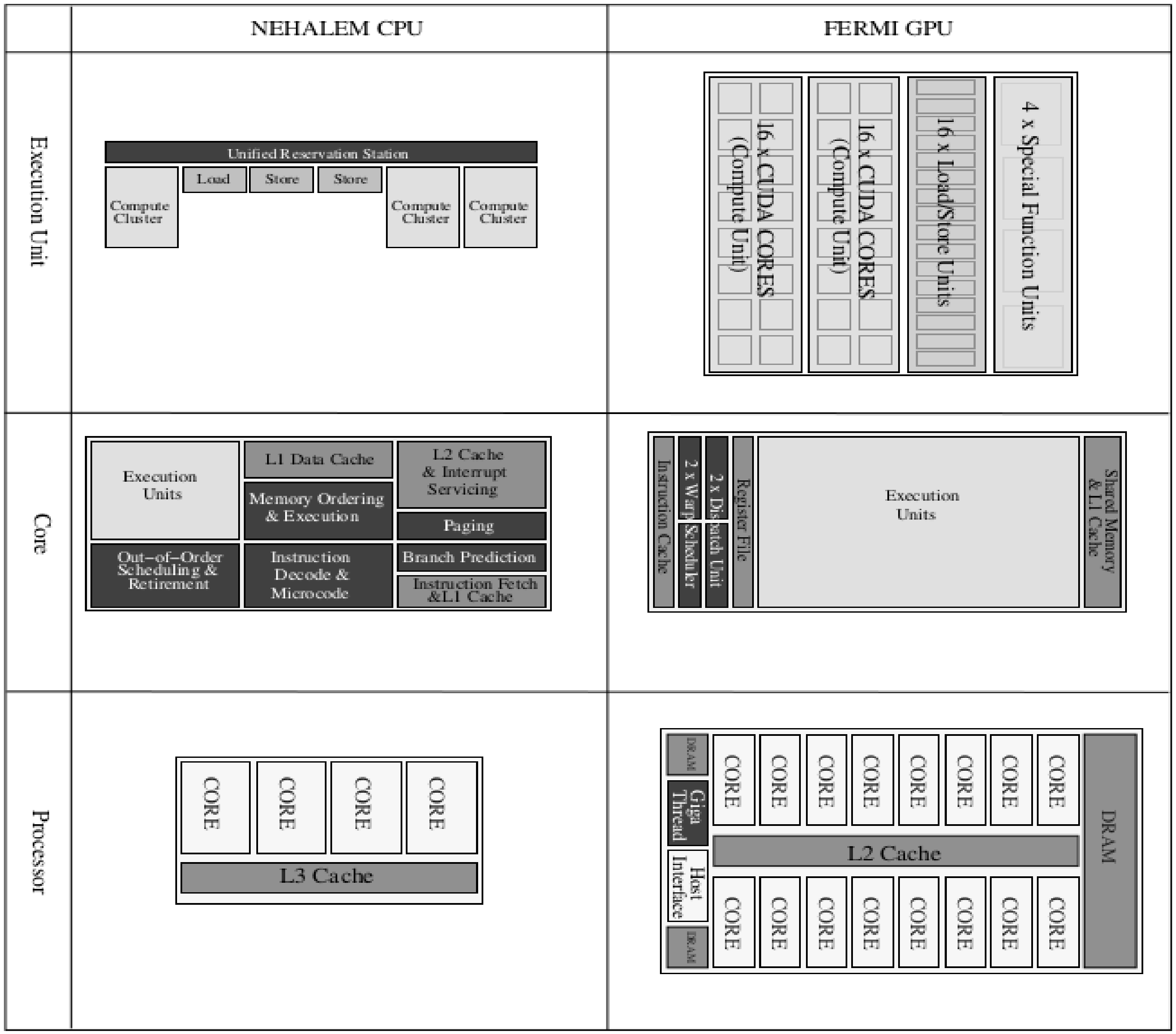
?

Multicore and Manycore

- Meaning: multi- ~ many-
- Actually:
  - Multicore CPU: 2 cores, 4 cores, 8 cores
  - Manycore GPU: 100 to 1000 cores

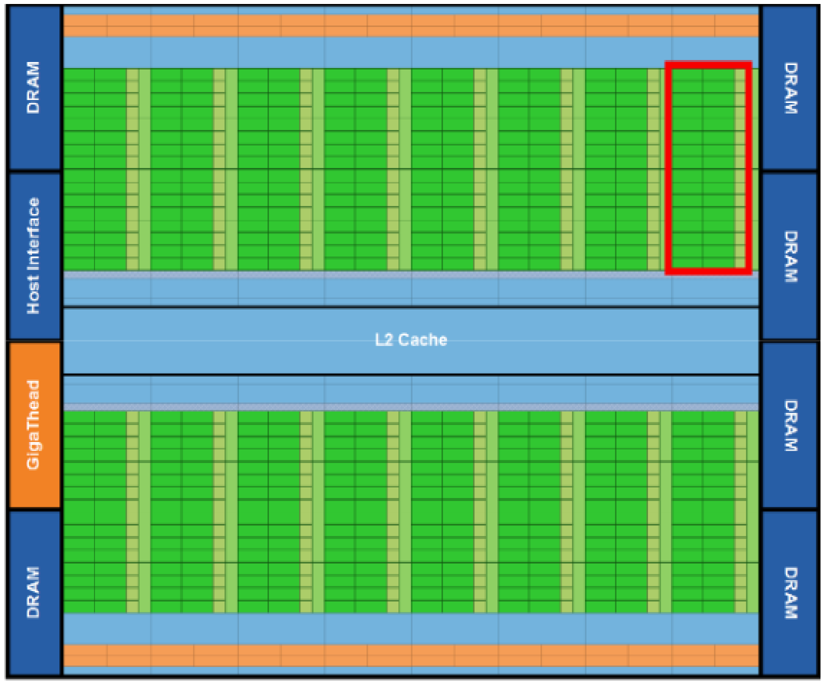
The GPU cores has
- *lower frequency*
- *simpler structure*

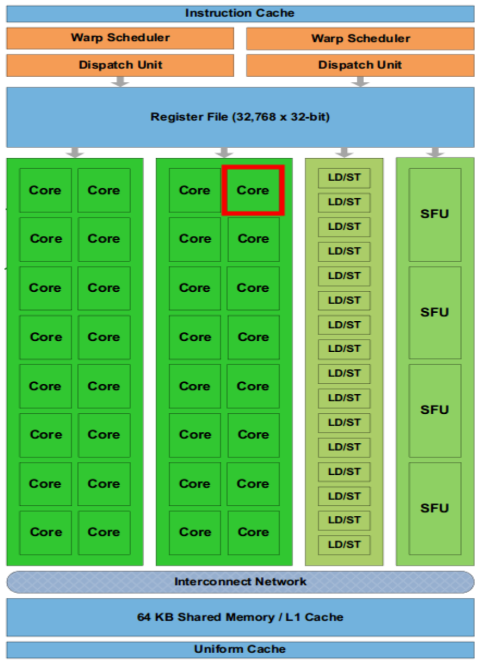
than the CPU cores

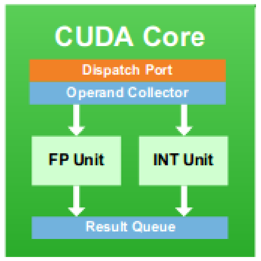


(Architecture comparation between Intel NEHALEM CPU and NVIDIA Fermi GPU at three levels: processor, core and execution unit, based on [5] and [6])

Example: Inside a Fermi GPU





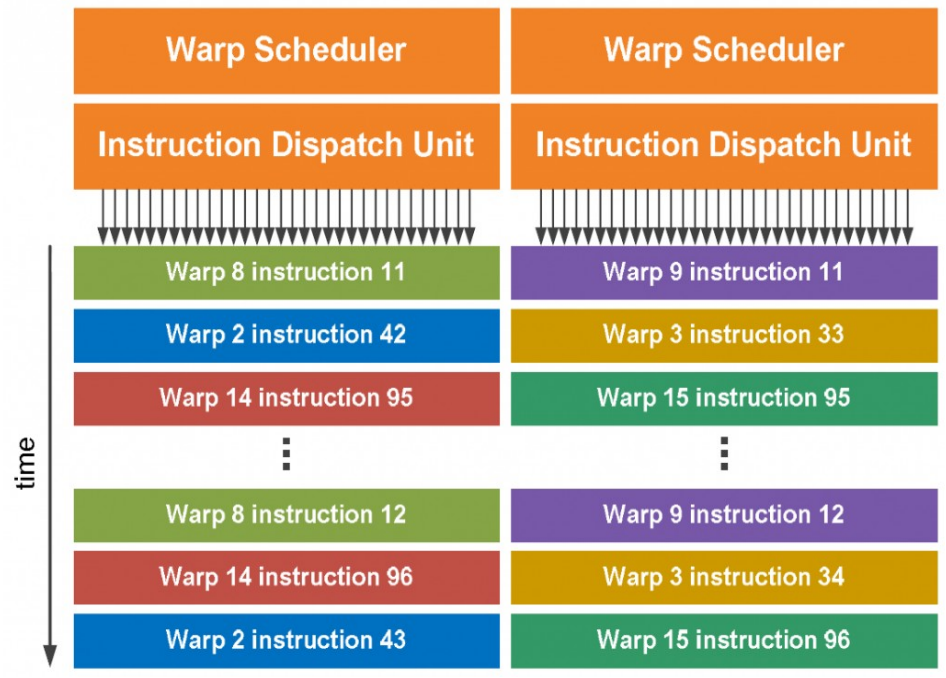


(images source: [6])

1 Fermi GPU contains 16 Streaming Multiprocessors (SM)
1 SM contains 32 CUDA cores

CUDA Warp

32 consecutive threads = 1 warp



(image source: [6])

- Program ~ Sequence of Instructions

(*In this practical, a kernel can be considered as a program*)
  - SPMD: Single Program Multiple Data
  - SIMD: Single Instruction Multiple Data
- Between the threads in the same warp: SIMD
- Between the threads in different warp: SPMD

Helloword

```
./helloworld 1 128
blocksPerGrid = 1, threadsPerBlock = 128
```

```
Hello, I am thread 64, of block 0
Hello, I am thread 65, of block 0
Hello, I am thread 66, of block 0
Hello, I am thread 67, of block 0
Hello, I am thread 68, of block 0
Hello, I am thread 69, of block 0
Hello, I am thread 70, of block 0
Hello, I am thread 71, of block 0
Hello, I am thread 72, of block 0
Hello, I am thread 73, of block 0
Hello, I am thread 74, of block 0
Hello, I am thread 75, of block 0
Hello, I am thread 76, of block 0
Hello, I am thread 77, of block 0
Hello, I am thread 78, of block 0
Hello, I am thread 79, of block 0
Hello, I am thread 80, of block 0
Hello, I am thread 81, of block 0
Hello, I am thread 82, of block 0
Hello, I am thread 83, of block 0
Hello, I am thread 84, of block 0
Hello, I am thread 85, of block 0
Hello, I am thread 86, of block 0
Hello, I am thread 87, of block 0
Hello, I am thread 88, of block 0
Hello, I am thread 89, of block 0
Hello, I am thread 90, of block 0
Hello, I am thread 91, of block 0
Hello, I am thread 92, of block 0
Hello, I am thread 93, of block 0
Hello, I am thread 94, of block 0
Hello, I am thread 95, of block 0
```

warp 2

```
Hello, I am thread 96, of block 0
Hello, I am thread 97, of block 0
Hello, I am thread 98, of block 0
Hello, I am thread 99, of block 0
Hello, I am thread 100, of block 0
Hello, I am thread 101, of block 0
Hello, I am thread 102, of block 0
Hello, I am thread 103, of block 0
Hello, I am thread 104, of block 0
Hello, I am thread 105, of block 0
Hello, I am thread 106, of block 0
Hello, I am thread 107, of block 0
Hello, I am thread 108, of block 0
Hello, I am thread 109, of block 0
Hello, I am thread 110, of block 0
Hello, I am thread 111, of block 0
Hello, I am thread 112, of block 0
Hello, I am thread 113, of block 0
Hello, I am thread 114, of block 0
Hello, I am thread 115, of block 0
Hello, I am thread 116, of block 0
Hello, I am thread 117, of block 0
Hello, I am thread 118, of block 0
Hello, I am thread 119, of block 0
Hello, I am thread 120, of block 0
Hello, I am thread 121, of block 0
Hello, I am thread 122, of block 0
Hello, I am thread 123, of block 0
Hello, I am thread 124, of block 0
Hello, I am thread 125, of block 0
Hello, I am thread 126, of block 0
Hello, I am thread 127, of block 0
```

warp 3

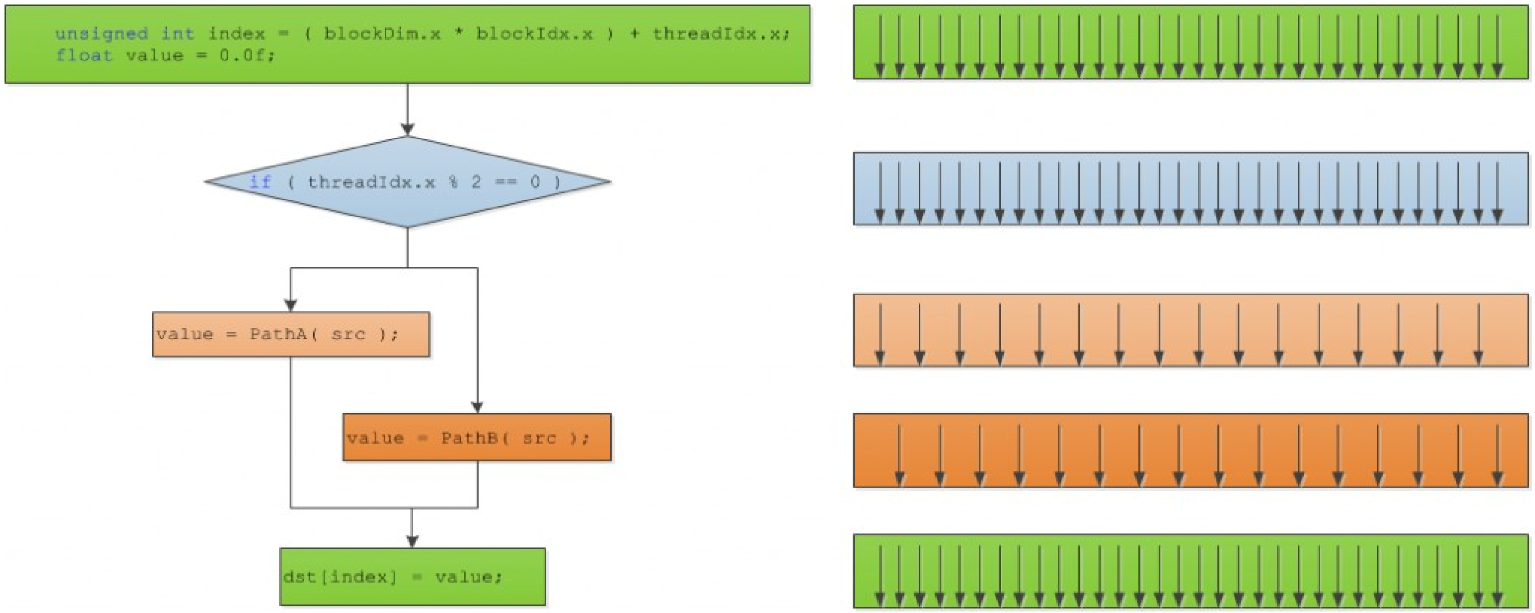
```
Hello, I am thread 0, of block 0
Hello, I am thread 1, of block 0
Hello, I am thread 2, of block 0
Hello, I am thread 3, of block 0
Hello, I am thread 4, of block 0
Hello, I am thread 5, of block 0
Hello, I am thread 6, of block 0
Hello, I am thread 7, of block 0
Hello, I am thread 8, of block 0
Hello, I am thread 9, of block 0
Hello, I am thread 10, of block 0
Hello, I am thread 11, of block 0
Hello, I am thread 12, of block 0
Hello, I am thread 13, of block 0
Hello, I am thread 14, of block 0
Hello, I am thread 15, of block 0
Hello, I am thread 16, of block 0
Hello, I am thread 17, of block 0
Hello, I am thread 18, of block 0
Hello, I am thread 19, of block 0
Hello, I am thread 20, of block 0
Hello, I am thread 21, of block 0
Hello, I am thread 22, of block 0
Hello, I am thread 23, of block 0
Hello, I am thread 24, of block 0
Hello, I am thread 25, of block 0
Hello, I am thread 26, of block 0
Hello, I am thread 27, of block 0
Hello, I am thread 28, of block 0
Hello, I am thread 29, of block 0
Hello, I am thread 30, of block 0
Hello, I am thread 31, of block 0
```

warp 0

```
Hello, I am thread 32, of block 0
Hello, I am thread 33, of block 0
Hello, I am thread 34, of block 0
Hello, I am thread 35, of block 0
Hello, I am thread 36, of block 0
Hello, I am thread 37, of block 0
Hello, I am thread 38, of block 0
Hello, I am thread 39, of block 0
Hello, I am thread 40, of block 0
Hello, I am thread 41, of block 0
Hello, I am thread 42, of block 0
Hello, I am thread 43, of block 0
Hello, I am thread 44, of block 0
Hello, I am thread 45, of block 0
Hello, I am thread 46, of block 0
Hello, I am thread 47, of block 0
Hello, I am thread 48, of block 0
Hello, I am thread 49, of block 0
Hello, I am thread 50, of block 0
Hello, I am thread 51, of block 0
Hello, I am thread 52, of block 0
Hello, I am thread 53, of block 0
Hello, I am thread 54, of block 0
Hello, I am thread 55, of block 0
Hello, I am thread 56, of block 0
Hello, I am thread 57, of block 0
Hello, I am thread 58, of block 0
Hello, I am thread 59, of block 0
Hello, I am thread 60, of block 0
Hello, I am thread 61, of block 0
Hello, I am thread 62, of block 0
Hello, I am thread 63, of block 0
```

warp 1

Problem of Branch Divergence



unsigned int index = (blockDim.x * blockIdx.x) + threadIdx.x;
float value = 0.0f;
if (threadIdx.x % 2 == 0)
value = PathA(src);
value = PathB(src);
dst[index] = value;

(image source: [4])

- Increase the running time of the whole program
- Waste computing resources

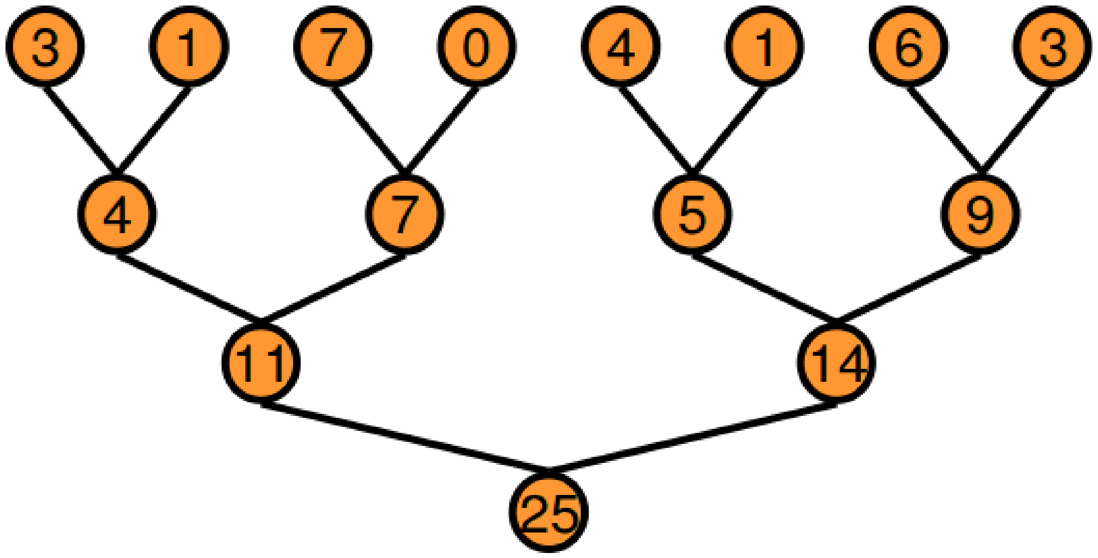
Part 2:

Implementation the reduction algorithm on GPU

Reduction

- Given an array of value => reduce them to a single value.
  - Sum reduction => sum of all values in the array
  - Max reduction => maximal value of the array
  - etc
- In this practical, we work with the Sum Reduction

Parallel reduction algorithm



(image source: [1])

Implementation Manual

- Based on the given skeleton: cudaReduction.cu (or from the source code of the squareArray application in Day 1)
- Using the ideas and the pseudo-codes in:

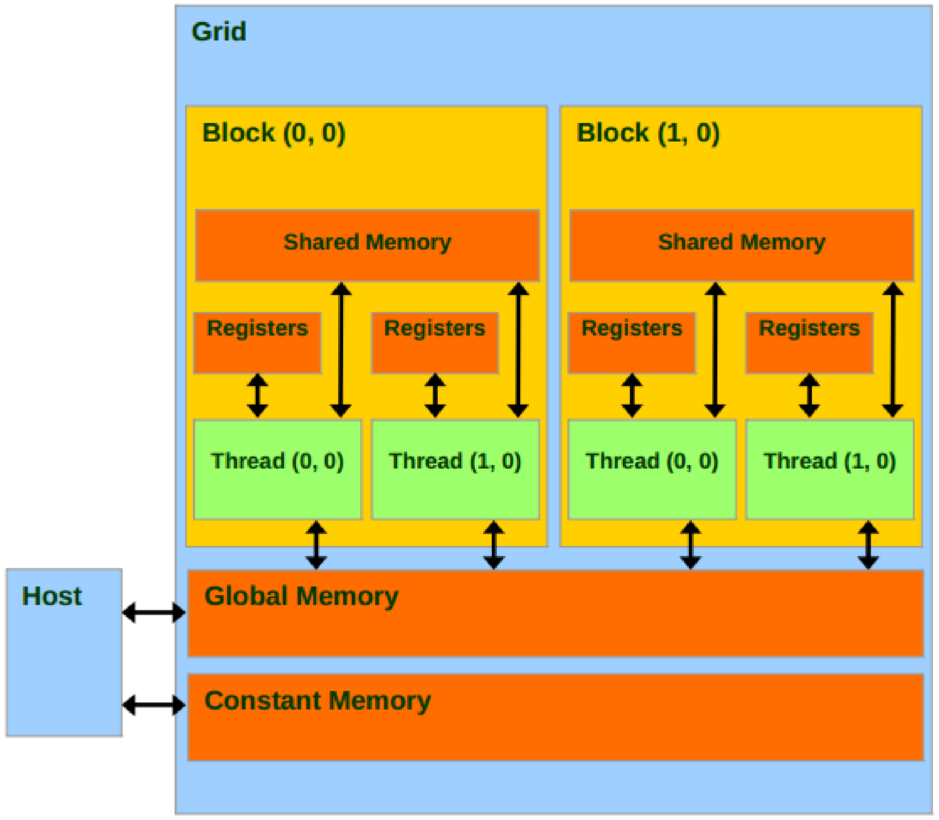
  [1] Mark Harris, Optimizing Parallel Reduction in CUDA,

  http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf

- **Basic implementation**: the tree-based approach with 1 thread block, with small data (up to 1024 elements), by using 2 algorithms:
  - Interleaved addressing with divergent branching [1, page 7]
  - Interleaved addressing without divergent branching [1, page 11]
- **Advanced implementation**: based on the basic implementation for small data, write the full CUDA application to process the large data of any size, by using recursive kernel invocation [1, page 7]. We suppose that the size of the input array is the power of 2, and up to 32M (33554432 elements).
- **Extend implementation**: the tree-based approach with 1 thread block, with small data (up to 1024 elements), by using the other ideas and optimizations in [1],

A brief introduction to shared memory

- Much faster than global memory
- Small (upto 48 KB)
- Shared among the thread in the same block
- *(more details will be discussed in Day 3)*



(image credit: NVIDIA)

Using the shared memory inside the CUDA code

The shared memory can be declared and allocated **inside** or **outside** the kernel:

Inside the kernel, the size must be fixed

```
#define LEN 8
__global__ kernel1(…)
{
    __shared__ int  arr1[LEN];
    __shared__ int  arr2[16];
    ...
}
```

Outside the kernel, the size can be set in the host code,
by using the third configuration argument when invoking the kernel:

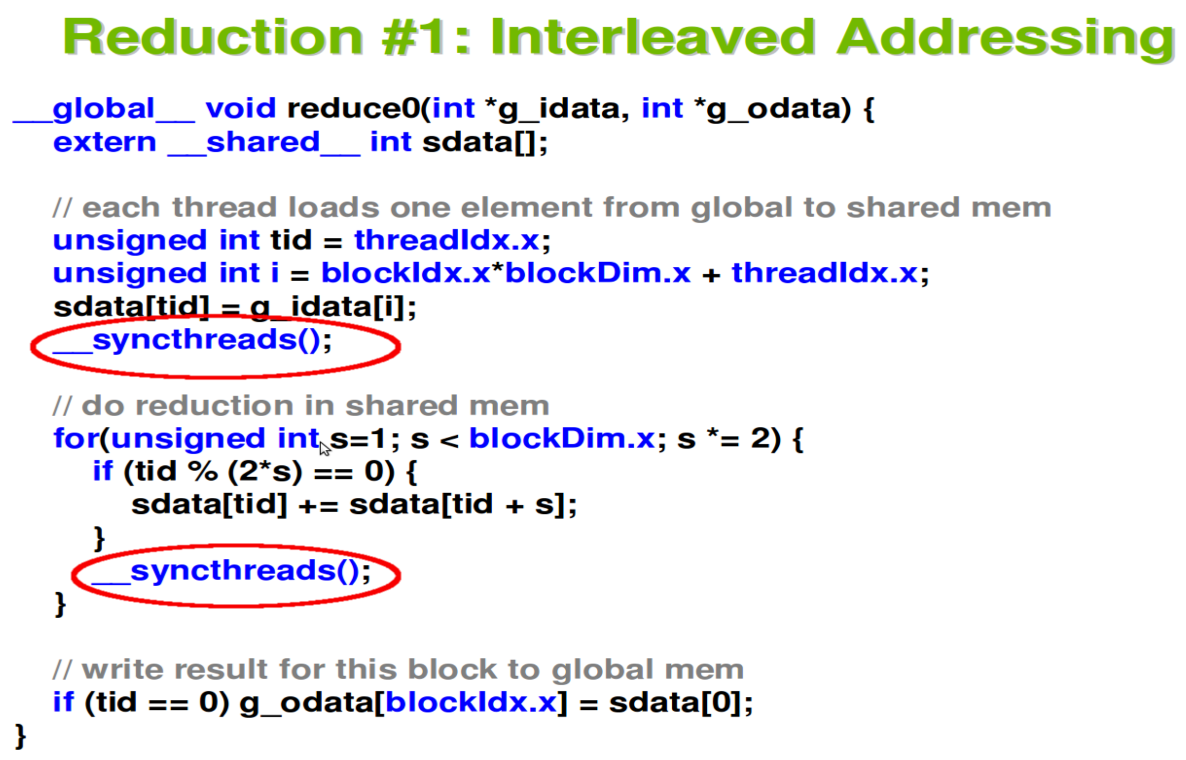
```
#define LEN 8
__global__ kernel1(…)
{
    extern __shared__ int  arr[];
    ...
}
//
int main(int argc, char** argv)
{
    ...
    n = 24;
    sharedSize = n * sizeof(int);
    kernel1<<<nBlock,nThread,sharedSize>>>(...)
    ...
}
```

Declare multiple __shared__ memory segments

```
__global__ kernel1(…,len1,len2)
{
    extern __shared__ int  arr[];
    int* arr1 = arr;
    int* arr2 = arr1 + len1;
    ...
}
//
int main(int argc, char** argv)
{
    ...
    len1 = 8;
    len2 = 16;
    n = len1 + len2;
    sharedSize = n * sizeof(int);
    kernel1<<<nBlock,nThread,sharedSize>>>(...,len1,len2)
    ...
}
```

Thread Synchronization

- Using the command: `__synchthreads()`
- Example (from [1]):



Reduction #1: Interleaved Addressing

```
__global__ void reduce0(int *g_idata, int *g_odata) {
   extern __shared__ int sdata[];

   // each thread loads one element from global to shared mem
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
   sdata[tid] = g_idata[i];
   __syncthreads();

   // do reduction in shared mem
   for(unsigned int s=1; s < blockDim.x; s *= 2) {
      if (tid % (2*s) == 0) {
         sdata[tid] += sdata[tid + s];
      }
      __syncthreads();
   }

   // write result for this block to global mem
   if (tid == 0) g_odata[blockIdx.x] = sdata[0];
}
```

Reference

[1] Mark Harris, Optimizing Parallel Reduction in CUDA,

http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf

[2] Hendrik Lensch and Robert Strzodka, Massively Parallel Computing with Cuda, Control Flow

http://www.mpi-inf.mpg.de/~strzodka/lectures/ParCo08/slides/Par03-ControlFlow.pdf

[3] Shalf, J., Asanovic, K., Patterson, D., Keutzer, K., Mattson, T., and Yelick, K. (2009). The Manycore Revolution: Will HPC Lead or Follow? SciDAC Review, 14.

[4] Jeremiah van Oosten, Optimizing CUDA Application, http://3dgep.com/?p=2081

[5] Semin, A. (2009). Inside Intel Nehalem Microarchitecture. In NOTUR 2009, The 8th Annual Meeting on High Performance Computing and Infrastructure in Norway.

[6] NVIDIA Corp. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. white paper, version 1.1

And some images from the CUDA technical documents of NVIDIA

Question ?