CMPSCI 691AD – General Purpose Computation on the GPU

Spring 2009

Lecture 3: CUDA Programming Basics I

Rui Wang

Misc. from last time

- printf in device emu mode
- CUDA architecture has removed vector processors, only scalar processors (SP).
  - 2 special function units (SFUs) per SM
  - GT200 added double precision processor, 1 per SM
  - Full integer support (arithmetic, bitwise...)
    - 4 clock cycles for integer add and bitwise op.
    - For integer multiplication: 4 cycles → 24-bit integer; 16 cycles → 32-bit.

Misc. from last time

- In SLI modes, multiple GPUs appear as one device.
  - Resources are duplicated.
  - Can enable multiple devices by disabling SLI.
- Calculation of GFLOPS:
  - Take 280 GTX as an example:

    30 SMs x 8 SPs/SM x 1.296 Ghz/SP x 3 = 933 GFLOPS

CUDA Programming Model

- Kernel functions
  - Executed N times in parallel by N different threads
  - Defined using the **__global__** qualifier.

```
// Kernel definition
__global__ void vecAdd(float *v1, float *v2, float *out, int n)
{
    ...
}

void main()
{
    // Kernel invocation
    vecAdd<<<1, N>>>(v1, v2, out, n);
}
```

CUDA Programming Model

- Each thread is given a unique threadID accessible from the built-in variable **threadIdx.**

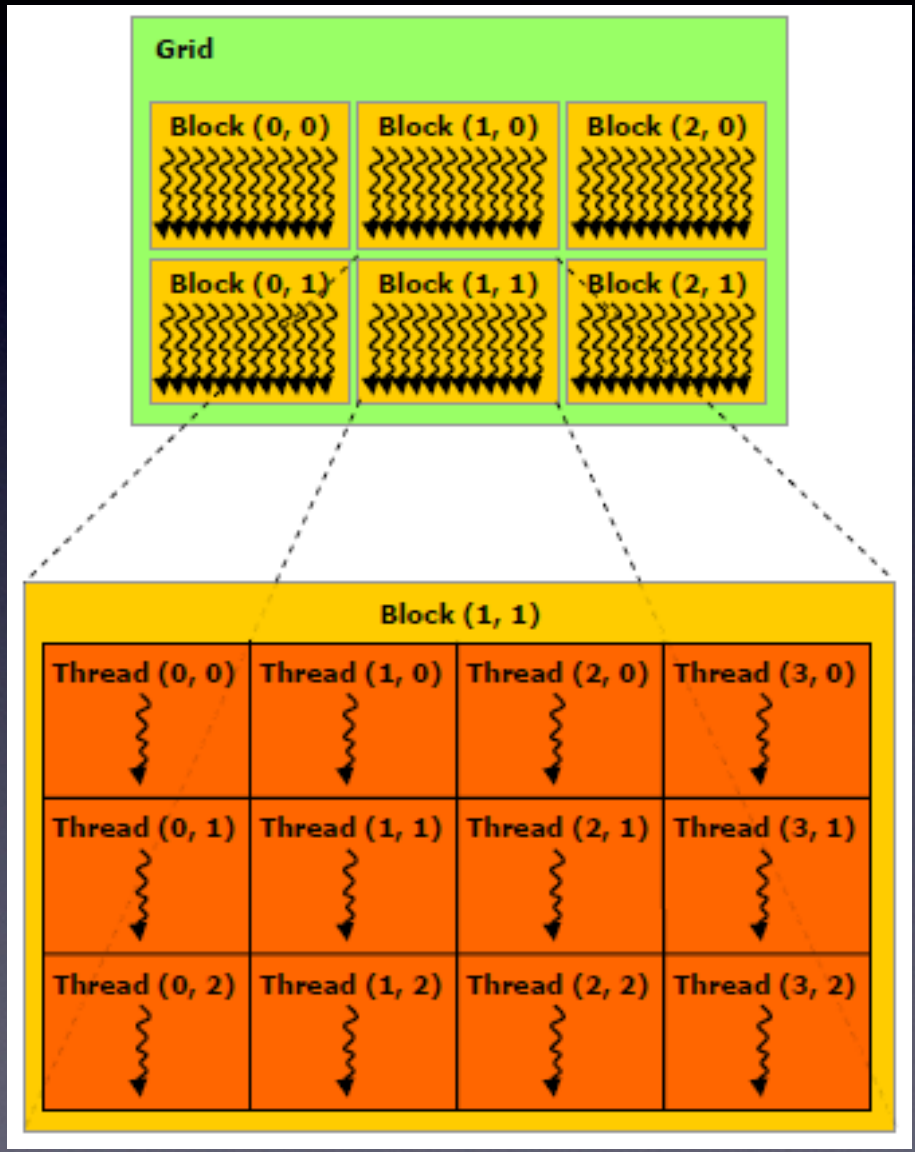
```
// Kernel definition
__global__ void vecAdd(float *v1, float *v2, float *out, int n)
{
    int idx = threadIdx.x;
    out[idx] = v1[idx] + v2[idx];
}

void main()
{
    // Kernel invocation
    vecAdd<<<1, N>>>(v1, v2, out, n);
}
```

Thread Hierarchy

- **threadIdx** is a 3-dimensional variable (type dim3)
  - Natural for 1D, 2D, 3D computation domains.
- Threads are organized into groups defined as thread blocks (TBs)
  - Each TB has the same number of threads.
  - Can be synchronized using barrier: __syncthreads()
- The collection of blocks is called a grid.
  - The layout of blocks in a grid can also be up to 3D
- Why partition into blocks?

Thread Hierarchy

- Blocks are also indexed
  - blockIdx;
- Example:
  - Image processing
  - Volume rendering
  - …



Thread Hierarchy

- Example:
  - Each block has 16x16 = 256 threads

```
__global__ void matAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x,
                 (N + dimBlock.y - 1) / dimBlock.y);
    matAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

Thread Hierarchy

- Example:
  - The actual computation elements may not be a multiple of thread block size.

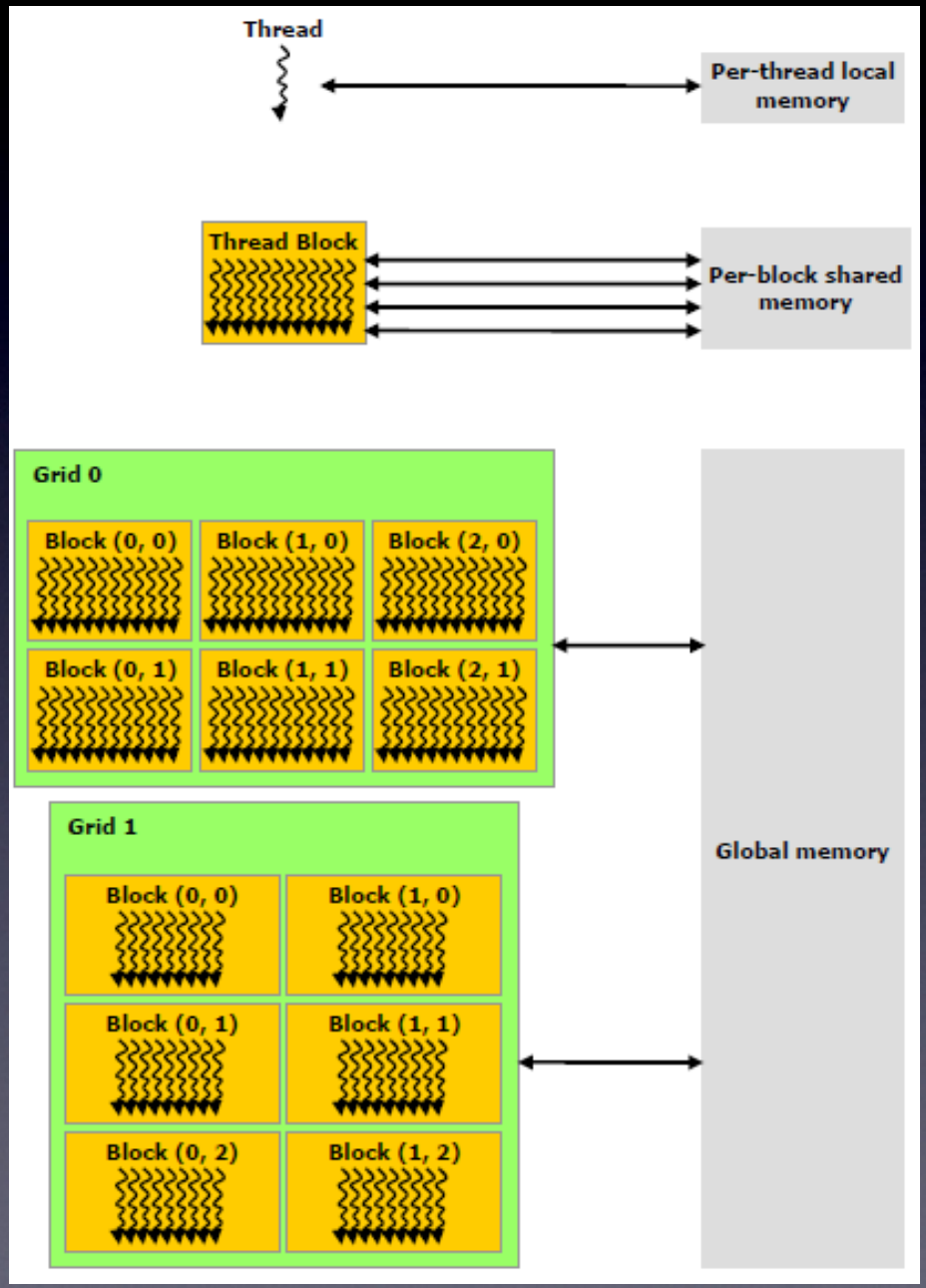
```
__global__ void matAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x,
                 (N + dimBlock.y - 1) / dimBlock.y);
    matAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

Thread Hierarchy

- Again, only threads within a block can be synchronized with __syncthreads()
- Different blocks execute independently, and the order may be arbitrary.
- Sync of all blocks achieved at the end of the kernel function call.
- Threads (including across blocks) can be coordinated with atomic instructions (such as atomAdd ...)

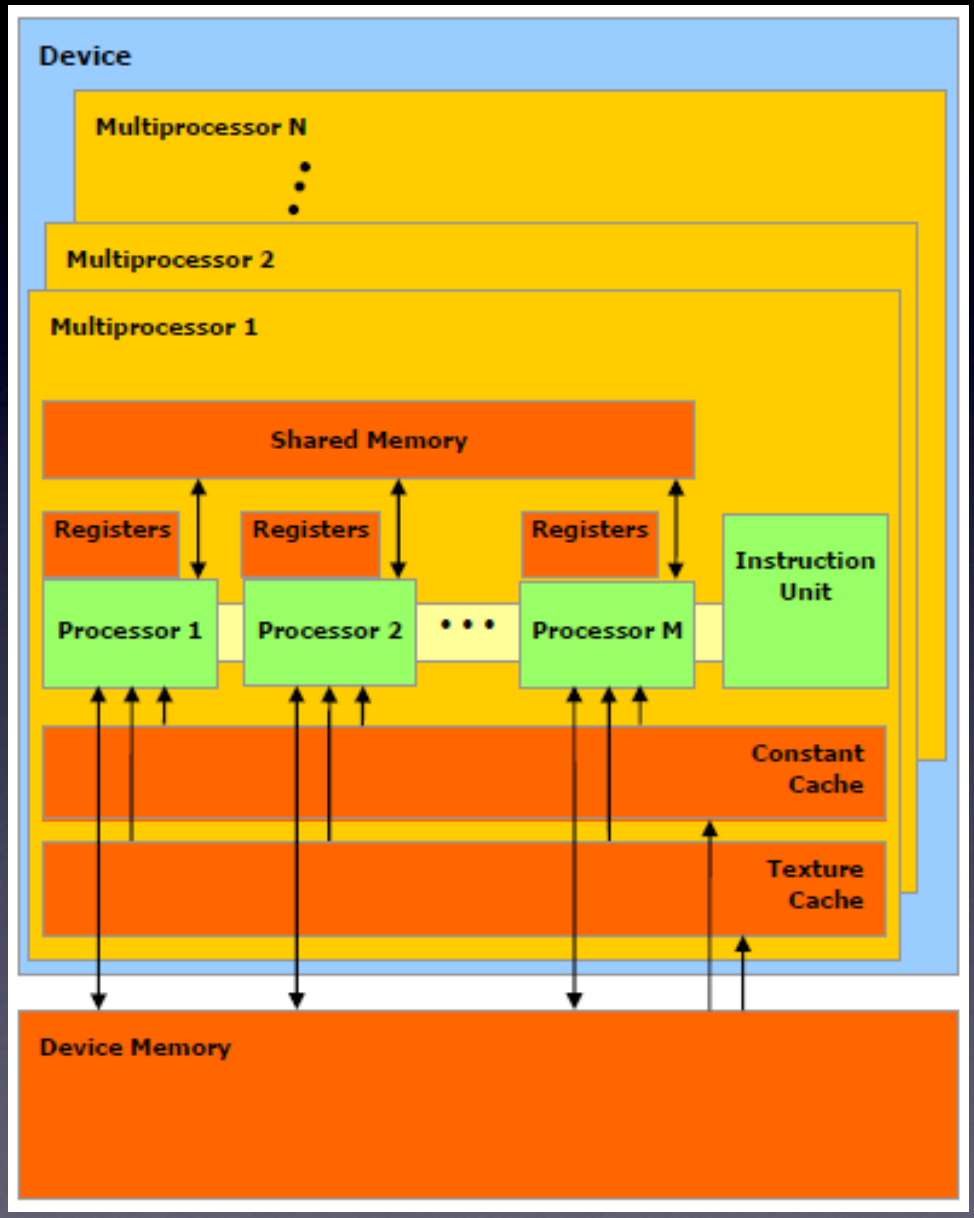
Memory Hierarchy

- Registers
- Per thread local memory
- Per block shared memory
- Global memory
- Read-only texture memory and constant memory
  - Global, texture, and constant memory spaces are persistent in the same application.



Memory Hierarchy

- Registers
- Per thread local memory
- Per block shared memory
- Global memory
- Read-only texture memory and constant memory
  - Global, texture, and constant memory spaces are persistent in the same application.

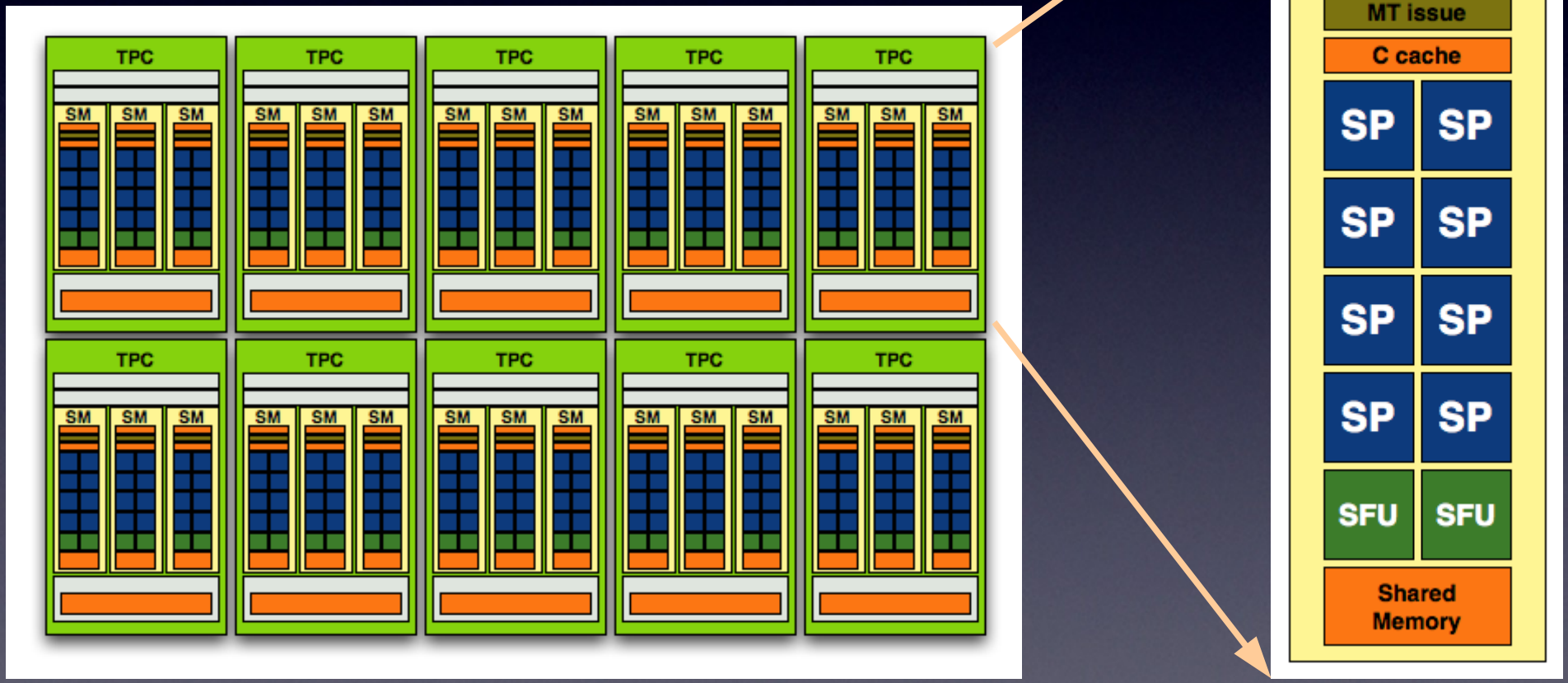


Memory Hierarchy

- Some numbers:
- CUDA 1.0 Compute Capability
  - Registers: 8192 per SM (doubled in CC1.2)
  - Shared memory: 16 kB per SM
  - Constant memory: 64 KB
  - Constant memory cache: 8 KB per SM
  - Texture memory cache: 6-8 KB per SM
  - Global memory: 512 MB to several GB

Distributing Threads to SMs

GT 200 Series



Distributing Threads to SMs

- Each thread is mapped to one scalar processor.
- Each block is carried out on one SM.
- Each SM creates, schedules, and executes threads in groups of 32 threads called **warps**.
  - Half-warp: either first or second half of a warp.
- A block is split into warps for execution.
  - Example: 256 threads per block → 8 warps
- Overhead for scheduling threads is very small.
  - Lightweight, zero scheduling overhead.

Distributing Threads to SMs

- On a single SM, at each instruction issue time:
  - One warp that's ready to execute is selected
  - 8 threads of a warp are allocated to 8 SPs.
  - A warp executes one common instruction at a time
  - Full efficiency if execution path is agreed
  - Divergent paths are serialized
    - Executes each branch serially, by disabling inactive threads
  - Different warps can diverge independently.

Distributing Threads to SMs

- Each SM takes 4 cycles to execute one warp
  - 32 warps / 8 SPs = 4
- Each SM can schedule multiple blocks as long as there are available resources
  - Why maintain multiple blocks?
  - Registers and shared memory space are split among all active threads.
  - Each SM can manage up to 8 blocks concurrently.

Distributing Threads to SMs

- Rules of thumb
  - Block size (#threads/block)?
    - Small block size results in more blocks, which is good to keep each SM busy, and hide memory access latency
    - Block size should be a multiple of warp size: 32, so ideally at least 32.
    - Synchronization is achieved only at block level.
    - Limited by the maximum number of blocks per SM.

Distributing Threads to SMs

- Some numbers:
- CUDA 1.0 Compute Capability
  - Maximum number of threads per block: 512
  - Warp size: 32
  - Maximum blocks per SM: 8
  - Maximum warps per SM: 24
  - Maximum threads per SM: 768 (32 x 24)

Distributing Threads to SMs

- Example:
- Choice between block size of 8x8, 16x16, 32x32
  - For 8x8, we have 64 threads per block. Since each SM can take up to 768 threads, there are 12 blocks. However, each SM can only take up to 8 blocks, so only 512 threads per SM.
  - For 16x16 → 256 threads per block → 3 blocks per SM → Good.
  - For 32x32 → 1024 threads per block, cannot fit!