Lecture 9

Performance Programming with CUDA
**Announcements**

- Lincoln Accounts
- A3
  - Implement the iterative solver with MPI or CUDA
- **CSE 260 Symposium**
  - Week 10
  - 14 project presentations
  - Need to add a 2 hour session
    (Makeup for 11/19)
  - Proposal: Weds, Dec 2nd 11AM to 1PM
Performance Issues

- Using shared memory
- Occupancy
- Coalesced memory access
- Divergent threads
NVIDIA GeForce GTX 280

- Each cluster contains 3 streaming multiprocessors
- Each multiprocessor contains 8 cores that share a local memory
- 3 flops/core/cyc * 8 cores/SM * 3 SM/cluster * 10 clusters = 720 flops/cyc
- @ 1.296 Ghz: 933 GFLOPS/EC
- Compute capability 1.3 (see Appendix A.1 of *Programming Guide*)
Memory Hierarchy

<table>
<thead>
<tr>
<th>Name</th>
<th>Latency (cycles)</th>
<th>Cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>DRAM – 100s</td>
<td>No</td>
</tr>
<tr>
<td>Local</td>
<td>DRAM – 100s</td>
<td>No</td>
</tr>
<tr>
<td>Constant</td>
<td>1s – 10s – 100s</td>
<td>Yes</td>
</tr>
<tr>
<td>Texture</td>
<td>1s – 10s – 100s</td>
<td>Yes</td>
</tr>
<tr>
<td>Shared</td>
<td>1</td>
<td>--</td>
</tr>
<tr>
<td>Register</td>
<td>1</td>
<td>--</td>
</tr>
</tbody>
</table>

Courtesy David Kirk/NVIDIA and Wen-mei Hwu/UIUC
LANGUAGE EXTENSIONS

- Superset of C - .cu files compile to C files for both the device and the host
- New keywords (__global__, __device__, __shared__, __syncthreads())
- New types (dim3, uint[1..4], float[1..4], …)
- New built-in variables (gridDim, blockDim, blockIdx, threadIdx)
- __global__ and __device__ functions have limitations since they run on the device
  - No recursion
  - No closures (static function variables)
  - No varargs
Handling 2D thread hierarchies

const int N=32;
_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() {
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N+dimBlock.x-1) / dimBlock.x, 
                  (N + dimBlock.y-1)/ dimBlock.y);
    MatAdd<<<gridDim, dimBlock>>>(A, B, C);
}
Hierarchical Thread Organization

- Grid > Block > Thread
- Thread Blocks
  - Cooperate, synchronize, with access fast on-chip shared memory
  - Threads in different blocks communicate only through slow global memory
  - Blocks within a grid are virtualized, too
  - May configure number of threads in a block and the number of blocks
- Threads assigned to SM in units of blocks, up to 8 for each SM
- Compiler re-arranges loads to hide latencies

KernelA<<<2,3>,<3,5>>>
Thread scheduling

- Blocks are divided into warps of 32 threads (schedulable unit)
  - SM schedules ready warps, scoreboard, prioritized, zero overhead
  - All threads in a warp execute the same instruction: all branches followed: serialization, instructions may be disabled, divergence
- An SM may be assigned up to 8 thread blocks over 32 warps, max 1024 threads
• A minimum number of warps needed to hide memory latency
• Consider blocked matrix multiply
• 8x8: 64 threads/block, 12 blocks, only 512 threads/SM
• 16x16, 256 threads, 3 blocks, may not be able to hide latency
• Consider another application that uses 32 registers (floats) per thread
• Each SM gets 512 registers, 64 per core, how many threads/SM?
Thread Divergence

• All the threads in a warp execute the same instruction
• Different control paths are serialized
• Divergence when predicates are a function of the threadId
  
  ```
  if (threadId < 2) {}
  ```

• Avoids divergence, everyone execute the same path
  
  ```
  if (threadId / WARP_SIZE < 2) {}
  ```
Bank Conflicts

- To improve bandwidth, we organize memory into banks
- Each bank responds once per cycle
- Accesses are *coalesced* if accesses from a half warp are conflict-free
- We have a *bank conflict* if there is more than 1 access to the same bank in a cycle

```c
int idx = blockIdx.x*blockDim.x + threadIdx.x;
a[idx] = a[idx]+1.f;
```
Avoiding Conflicts

- 16 banks currently = 1/2 warp
- If 2 or more instructions in a 1/2 warp access different banks, we have a conflict
- No conflict if all access the same bank (broadcast)
Identifying conflicts

- Consider
  ```
  __shared__ float shared[256];
  float foo = shared[base + s * threadIdx.x];
  ```

- If s has no common factors with number of banks (16), then there are no conflicts (s is odd)
- Traditional wisdom for exploiting cache locality can result in bank conflicts
- What if a thread loads 2 consecutive array elements?
  ```
  int tid = threadIdx.x;
  shared[2*tid] = global[2*tid];
  shared[2*tid+1] = global[2*tid+1];
  ```

- To avoid conflicts
  ```
  shared[tid] = global[tid];
  shared[tid + blockDim.x] = global[tid + blockDim.x];
  ```
## Bank Conflicts in Reduction

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 2</th>
<th>Thread 4</th>
<th>Thread 6</th>
<th>Thread 8</th>
<th>Thread 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0+1</td>
<td>2+3</td>
<td>4+5</td>
<td>6+7</td>
<td>8+9</td>
<td>10+11</td>
</tr>
<tr>
<td>0..3</td>
<td>4..7</td>
<td>8..11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0..7</td>
<td>8..15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

David Kirk/NVIDIA & Wen-mei Hwu/UIUC
Avoiding bank conflicts

Thread 0
The Code

__global__ void reduce(int *input, unsigned int N, int *total) 
{
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ int x[blocksize]
    x[tid] = (i<N) ? input[i] : 0;
    __syncthreads();

    for (int s = blockDim.x/2; s=0; s /= 2) {
        if (tid < s) 
            x[tid] += x[tid + s];
        __syncthreads();
    }

    if (tid == 0) atomicAdd(total,x[tid]);
Measuring timings

- Kernel invocations are asynchronous
- To take timings, we need to synchronize

```c
cudaThreadSynchronize();
double t_device_compute = -getTime();
for (int r=0; r < 100; r++)
    incrementArray<<< nBlocks, blockSize >>> (a_d, N);
cudaThreadSynchronize();
t_device_compute += getTime();
```
Performance

N= 8388480, block size = 128

100 iterations, without synchronization
Host times: Total (1.344159), Compute (1.315612)
Device times: Total (3.036168), Compute (0.000568)

1000 iterations, without synchronization
Host times: Total (13.113861), Compute (13.085682)
Device times: Total (3.812472), Compute (0.004233)

With Synchronization. 100 and 1000 iterations
Host: Total (1.351412), Compute (1.322542)
Device times: Total (3.039179), Compute (0.086578)
Host times: Total (13.247272), Compute (13.218687)
Device times: Total (3.818402), Compute (0.864388)