Exam 2 Overview

ECE 6397 (GPU Programming) – David Mayerich
Occupancy Limitations

• How do registers and shared memory limit occupancy?
If a program requires some number of shared resources (registers, shared memory, etc) per thread, sufficient resources have to be available to the SM for all threads in the block. For example, there may be sufficient processing units available to support the math required within a thread, but insufficient shared memory to allow execution of sufficient threads to use those processing units.

• How does thread divergence limit occupancy?
Thread divergence occurs when threads within a SIMD process require different instructions. Since only one instruction can be issued at a time, threads that do not execute an instruction are stalled and don’t use system resources, thereby reducing occupancy.
Shared Memory

• You are processing an $N \times M$ output image using square blocks on a system that can support up to 1024 threads per block. Processing for each pixel requires 12 independent bytes of shared memory. What configuration parameters would you use to launch the kernel?

```c
<<< dim3( N/32+1, M/32+1 ), dim3(32, 32), 1024*12 >>>
```
Shared Memory

• If you’ve allocated 1024 bytes of shared memory per block in your grid configuration, how do you declare a pointer in the kernel to access this memory as an array of 64-bit floating point values?

```c
extern __shared__ double sharedPtr[];
```

• How would you specify two 32-bit floating point arrays (A and B) of equal size from the same allocated shared block?

```c
extern __shared__ char sharedPtr[];
float* array0 = (float*) sharedPtr;
float* array1 = (float*) (sharedPtr + 512);
```
Function Declarations

• What function declarations would you use to make the following function callable from a host function and a kernel?

```c
float legendre( float x, int N){
  if(N == 0) return 1.0f;
  if(N == 1) return x;
  float p[3]; p[0] = 1.0f; p[1] = x;
  for(int n = 2; n < N+1; n++){
    p[2] = 0.5 * (3 * x * p[1] + p[0]);
    p[0] = p[1];  p[1] = p[2];
  }
  return p[2];
}

__host__ __device__
```
Synchronization

• Briefly describe the thread divergence for the following launch configuration that occurs in the following kernel (using the Legendre polynomial function).

```c
__host__
__device__
float legendre( float x, int N){
    if(N == 0) return 1.0f;
    if(N == 1) return x;
    float p[3]; p[0] = 1.0f; p[1] = x;
    for(int n = 2; n < N+1; n++){
        p[2] = 0.5 * (3 * x * p[1] + p[0]);
        p[0] = p[1];  p[1] = p[2];
    }
    return p[2];
}
```

```c
__global__
void kernel(float* out, float x){
    size_t i = blockIdx.x * blockDim.x + threadIdx.x;
    int tid = threadIdx.x;
    out[i] = legendre(x, tid);
    __syncthreads();
}
```

• The kernel is launched with the following call:

```c
ekernel<<<N, 1024>>>(result, x);
```
CUDA Memory Model

• Assume that a kernel is executing the following loop that accesses the global memory pointer `float* C`:
  - `float b = 0;`
  - `for(size_t n = 0; n < N; n++)`
  - `b += b * C[n * M + i];`

If $N$ is large, what is the approximate floating point performance (in FLOPS) that can be achieved on a Titan X GPU, which has a maximum device memory bandwidth of 336 GB/s?

Since 2 floating point operations are required for every 4-byte fetch, the maximum occupancy would allow 168 GFLOPS
Data Parallelism

• SIMD – single instruction multiple data
  – one instruction is executed at a time simultaneously on different data

• SPMD – single program multiple data
  – the same program is executed in parallel on different data
  – programs don’t necessarily start or end at the same time

• Which of these is implemented on a GPU and how?
  – SPMD
    • streaming multiprocessors execute the same kernel on multiple data sets
    • kernel execution can start or end at different times (not all threads necessarily start or
      end simultaneously)
  – SIMD
    • code is executed simultaneously within a warp (usually 32 threads)
Data Parallelism

• **occupancy** – percentage of computational resources devoted to solving a problem

• **thread divergence** – when threads in a SIMD model that have to execute different lines of code
  – the SIMD system can only execute one instruction at a time
  – when some threads require instruction $A$ all others stall while $A$ is executed
  – this reduces occupancy since no resources are being used by stalled threads

• **causes of thread divergence:**
  – conditionals (if...else)
  – loops of different lengths (for, while, do, etc.)
  – unaligned global memory instructions
  – returning at different times

• **memory speed:** register file (fastest), shared memory, global memory
CUDA API

• The following pointer:
  
  ```c
  float* matrix;
  ```

  indicates a memory location on the host that contains data for an \( N \times M \) matrix. Write the code necessary to create a pointer `gpu_matrix` to the same data on the GPU.

  ```c
  float* gpu_matrix;
  cudaMalloc(&gpu_matrix, N*M*sizeof(float));
  cudaMemcpy(gpu_matrix, matrix, N*M*sizeof(float), cudaMemcpyHostToDevice);
  ```
CUDA API

• What API function would you use to get the number of CUDA capable devices on a system?
  
  ```c
cudaGetDeviceCount(int* count)
```

  ```c
  int d;
cudaGetDeviceCount(&d);
```

• What is the data type and value returned by a CUDA API function that executed correctly?
  
  ```c
cudaError_t error = cudaSuccess;
```
CUDA Devices

```python
props.major = 3
props.maxGridSize = {65535, 65535, 65535}
props.maxThreadsDim = {1024, 64, 64}
props.maxThreadsPerBlock = 1024
props.minor = 5
props.regsPerBlock = 65536
props.sharedMemPerBlock = 49152
props.totalConstMem = 65536
props.totalGlobalMem = 3.2205e+09
props.warpSize = 32
```

- What is the compute capability of this device? 3.5
- What is the maximum number of threads that you can launch for one kernel? \(1024 \times 65535^3\)
- What is global memory?
  - a block of memory available to all processors on a GPU
- What is constant memory?
  - a block of global memory that is read-only and has improved caching
- What is shared memory?
  - a small block of memory (usually 48Kb) available to all threads in a block (or all cores on an SM)
CUDA Devices

1) You are performing a matrix multiplication $C = AB$ where $A \in \mathbb{R}^{5000 \times 4200}$ and $A \in \mathbb{R}^{4200 \times 5000}$ by assigning each thread in a grid to an output element of $C$.

- If you want to maximize the number of threads per block, and minimize the number of threads that don’t do work, what configuration parameters would you use?
  
  $\langle\langle \text{dim}3(157, 157), \text{dim}3(32, 32)\rangle\rangle$

- How many warps will have stalled threads due to control divergence?
  
  5000
What function declarations are required for any function calling the following code?

```c
__device__ sign(float x){
    if(x == 0) return 0;
    if(x < 0) return -1;
    return 1;
}
```

Either `__global__` or `__device__`

What are three (3) similarities and differences between the following two variable declarations?

```c
__constant__ float TAU = 3.14159 * 2;
__device__ float TAU = 3.14159 * 2;
```

- they are both stored in device memory
- the `__constant__` value is read-only
- they are both global in scope
- they are both only accessible to device functions
CUDA Programming Language

• Describe the grid launched by the following configuration parameters (in terms of the variables used) [5 points]:

   \[
   \text{kernelFunc}<<<\text{dim3}(N, M), \text{dim3}(a, b, c)>>>(\text{gpu}_A, \text{gpu}_B, N, M);
   \]

   A 2D grid is composed of \( N \times M \) blocks.
   Each block contains a 3D group of \( a \times b \times c \) threads
CUDA Memory Model

- How would you align the following structure (fill in the blank) to guarantee that it can be retrieved by a single global memory instruction?

```c
__device__ __align(          )__
struct BadStruct{
    float a;
    char a;
    char b;
};
```

- This structure is 6 bytes in size, so `__align__(8)` will align the structure to 8 bytes

- How many device memory transactions would be issued by a warp if each thread accessed this aligned structure using the following code:

```c
int i = blockIdx.x * blockDim.x + threadIdx.x;
BadStruct b = globalStructs[i];
```

- A warp is composed of 32 threads. Since each structure is aligned to 8 bytes, $8 \times 32 = 256$ bytes. Therefore, $2 \times 128$-byte memory transactions are required
CUDA Memory Model

• How many device memory transactions would be issued by a warp accessing a global memory array of char values with the following code:

```c
int i = blockIdx.x * blockDim.x + threadIdx.x;
char r = globalChar[i];
```

  – Since each fetch is 1 byte, $1 \times 32 = 32$ bytes. Therefore 1 32-byte memory transaction will be required.