Programming with CUDA

Andy Loomis
GPU Computing

CPU Design Philosophy

◆ Make a single thread very fast
◆ Low latency through caching
◆ Predictive

GPU Design Philosophy

◆ Focus on high throughput—single threads do not matter
◆ Hide latency through parallelism
◆ Let programmer deal with data hierarchy
CUDA and OpenCL

CUDA

◆ Compute Unified Device Architecture
◆ Runs on Nvidia GPUs
◆ Compile using nvcc

OpenCL

◆ Open Computing Language
◆ Runs on heterogeneous platforms, both CPUs and GPUs
◆ Link your program to the OpenCL library
GPU Execution Model

Two-tiered Parallelism
- Grid
- Block

Threads in the same Block can communicate with each other.

Grids and Blocks replace the outer loops in an algorithm.
Example 1: Kernel Function

```c
__global__ void square_array(float* a, int n)
{
    // Get the unique index of this thread
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    // If the index refers to a valid member of the array, then square it
    if (x < n) {
        a[x] *= a[x];
    }
}
```
Example 1: Calling the Kernel Function

```c
int n;
float* a;

// Initialize n and a
...

// Run the kernel

int block_size = 256;
int grid_size = (n + block_size - 1) / block_size;
square_array <<< grid_size, block_size >>> (a, n);
```
Example 1: Compiling and Running

To compile and run:

```
$ module load cuda
$ nvcc –o example1 example1.cu
$ ./example1
```
Example 2: Kernel Function

// /users/aloomis/pub/CUDA_examples/example2.cu

// This function takes in a matrix of size w * h and returns its transpose.

__global__ void transpose_naive(float* in, float* out, int w, int h)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    if (i < h && j < w) {
        out[w*i+j] = in[h*j+i];
    }
}
Measuring Performance

What is going to be the limiting factor?

- Floating point throughput
- Memory bandwidth

Benchmark the limiting factor

Evaluate

- Know your peak throughputs.
- Are you getting close?
- Are you tracking the right limiting factor
Memory Model

Registers/Local Memory
◆ Only accessible by a single thread
◆ Limited space, very fast

Shared Memory
◆ Accessible by all the threads in a block
◆ Slower than registers, but faster than device memory
◆ Watch out for bank conflicts

Texture Memory
◆ Accessible by all threads, but is read only
◆ High latency, but spatially cached

Device Memory
◆ High bandwidth, high latency
Thread Synchronization

◆ With CUDA you can synchronize all of the threads in a single block using the `__syncthreads()` function.

◆ This is especially useful when you are writing and reading to shared memory.
Back to the Matrix Transpose

- Global device memory dislikes non-unit strides
- Shared memory doesn’t mind


- Read untransposed block from global memory and write it to shared
- Read transposed block from shared and write to global
Example 2

See:/users/aloomis/pub/CUDA_examples/example2.cu
Example 3

```c
__global__ void mystery_kernel(float* in, float* out, int n)
{
    extern __shared__ float sdata[];

    int i = blockDim.x * blockIdx.x + threadIdx.x;

    sdata[threadIdx.x] = (i < n) ? in[i] : 0.0f;
__syncthreads();

    for(int s = blockDim.x/2; s > 0; s >>= 1) {
        if (threadIdx.x < s) {
            sdata[threadIdx.x] += sdata[threadIdx.x + s];
        }
__syncthreads();
    }

    if (threadIdx.x == 0) {
        out[blockIdx.x] = sdata[0];
    }
}
```
Example 3

See: /users/aloomis/pub/CUDA_examples/example3.cu
Helpful CUDA Functions

cudaError_t cudaGetDeviceCount(int* num_of_devices)
cudaError_t cudaGetDevice(int* device_id)
cudaError_t cudaSetDevice(int device_id)
cudaError_t cudaGetDeviceProperties(
    cudaDeviceProp* properties, int device_id)
cudaError_t cudaChooseDevice(
    int* device_id, cudaDeviceProp* properties);
Error Checking

```c
// cudaMalloc((void**)&a_device, num_bytes);

cudaError_t cuda_error_code = cudaMalloc((void**)&a_device, num_bytes);
if (cuda_error_code != cudaSuccess)
{
    printf("cudaMalloc failed with error code %d\n", cuda_error_code);
    abort();
}
```

Typical Errors

- Out of memory
- Invalid parameters
- Kernel runs for too long
- Hardware errors
Other Resources

CUDA Zone: http://developer.nvidia.com/category/zone/cuda-zone

CUDA by Example: An Introduction to General-Purpose GPU Programming

CUDA C Programming Guide

CUDA Best Practices Guide

CUDA Reference Manual