VSCSE Summer School

Accelerators for Science and Engineering Applications: GPUs and Multi-cores

Understanding the labs

A typical CUDA program

```
void CUDA_interface (…){
   //allocate memory space in global device memory for input data
   cudaMalloc(…);
   //copy input data from host to the allocated device space
   cudaMemcpy(…);
   //allocate memory space in global device memory for the output
   cudaMalloc(…);

   //define block and grid size for the kernel;
   dim3 grid (x,y);
   dim3 block (x,y,z);

   // launch kernel
   CUDA_kernel<<<grid,block>>>(…);

   //copy output data from device memory to the host
   cudaMemcpy(…);

   //free all device allocated memory (inputs and outputs)
   cudaFree(…);
}
```

A typical CUDA program

```
void CUDA_kernel (…){
  //declare a shared memory array (optional)
  __shared__ array_s[…];
  //figure out index into different arrays in terms of
  blockIdx, threadIdx, and block_size
  int index = …;

  //bring in data from global memory (into registers, or
  shared memory)
  …
  //Do the computation
  …
  //Copy data back to global memory (from registers or
  global memory)
  …
}
```

Lab 1.1

- Objective: perform a matrix-matrix multiplication

  $$M*N = P$$

- Assumptions/Requirements:
  - There is no use of shared memory.
  - We operate on data in global memory and keep a running sum in a register. Every thread is ony responsible for computing its element.
- Difficulty levels
  - DL1: All the lines are given to you, with some function parameters missing, as well as some values of declared variables
  - DL2: Some lines are completely omitted
- Functions to modify:
  - Interface function runTest(…) in "matrixmul.cu"
  - Kernel function matrixMul(…) in "matrixmul_kernel.cu"

Lab 1.2

- Objective: perform a parallel reduction on an array to compute the total sum.
- Assumptions/Requirements:
  - There is only one tile/block
  - The array has exactly 512 elements in it
- Difficulty levels
  - DL1: All function calls are given to you with missing parameters. Reduction code inside the kernel has been omitted.
  - DL2: Some function calls have been omitted. Entire body of the kernel function has been omitted
- Functions to modify:
  - Interface function computeOnDevice(…) in vector_reduction.cu
  - Kernel function reduction(…) in vector_reduction_kernel.cu

Lab 2.1

- Objective: perform a matrix-matrix multiplication

  M*N = P

- Assumptions/Requirements:
  - We use shared memory to load in intput data tiles
  - Every thread is reponsible for loading data from global to shared memory, and computing the value of 1 output element.
- Difficulty levels
  - DL1: All the lines are given to you, with some array indeces missing in the kernel function.
  - DL2: All lines are given to you, with some some array indeces missing, as well as the initial values of some variables.
- Functions to modify:
  - Kernel function matrixMul(…) in “matrixmul_kernel.cu”

Lab 2.2

- Objective: perform a parallel reduction on an array to compute the total sum.
- Assumptions:
  - The array can be of any size.
  - The code should be able to handle sizes larger than 1 tile size
- Difficulty levels
  - DL1: Timer and kernel synchronization omitted in interface function. Kernel code given works for 1 tile of 512 elements.
  - DL2: Timer and kernel synchronization omitted in interface function. Kernel code removed.
- Functions to modify:
  - Interface function runTest(…) in reduction_largearray.cu
  - Kernel function reductionArray(…) in reduction_largearray_kernel.cu

Lab 3.1

- Objective: tune performance of matrix-matrix multiplication

  M*N = P
- Assumptions/Requirements:
  - Tune the performance of the program, using predefined macros.
- Difficulty levels
  - N/A.
- Functions to modify:
  - Parameters in “marixmul.h”
- Additional objectives:
  - Use the CUDA profiler to profile your program.

Lab 3.2

- Objective: optimize the performance of an MRI application.
- Assumptions:
  - N/A
- Difficulty levels
  - DL1: Using predefined macros in “computeQ.h”, tune the application and observe its performance.
  - DL2: Modify the unoptimized kernel in “computeQ.cu” to improve performance.
- Functions to modify:
  - See Difficulty levels above.

Questions?