

# Iowa High Performance Computing Summer School 2012

## GPU Computing Using CUDA C Exercises

Thursday, June 7, 2012

NOTE: Because CUDA C is an extension of the C programming language, all of the exercises below must be written in C.

### 1. Vector Addition Using CUDA C kernel

Here we will write a code that is the “Hello World” equivalent for CUDA C programming, a code that adds two vectors  $A$  and  $B$  of length  $N$ , to obtain a third vector  $C = A + B$ , element by element.

- (a) First write a serial C program that defines three `float` arrays  $A$ ,  $B$ , and  $C$  of size  $N$ . Fill arrays  $A$  and  $B$  with some non-trivial initial values. Then call a function, passing in the vectors  $A$  and  $B$ , that computes  $C = A + B$  element by element.
- (b) Choose  $N$  large enough that this operation takes a minimum time  $T$  of 10 seconds, but not more than 60 seconds. Due to memory size limitations, you may need to add an outer loop to the code to repeat the calculation  $n$  times, choosing an appropriate value of  $n$  to obtain  $10 \text{ s} \leq T \leq 60 \text{ s}$ . Measure the time it takes to perform this function using the UNIX `time` command, `time vecadd_serial.e`
- (c) Create a CUDA C version of the code (saving the serial C version) that employs a CUDA C kernel to perform the vector addition on the GPU device. Don’t forget to allocate memory on the device for  $A$ ,  $B$ , and  $C$ , copy data from host to device and back as necessary, and free the device memory allocation when you are done.
- (d) Time the execution of the GPU code using the UNIX `time` command, as above. Consider whether or not you want the memory allocation and free calls on the device to fall within the outer  $n$  loop. Try it both ways to see if including the memory allocation calls leads to a noticeably longer execution time.

### 2. Monte Carlo Determination of the Value of $\pi$ Using the GPU

Here we will use the GPU to compute the value of  $\pi$  using the Monte Carlo method.

- (a) Start with the serial C version of the Monte Carlo  $\pi$  code that you wrote for the MPI Programming Exercises. If you wrote that code in Fortran, you will have to translate it to C for this exercise.
- (b) If you have not written the serial C version in this manner, modify it so that the value of  $\pi$  is computed in a function call, with the only arguments passed being the number of Monte Carlo points  $N$  and the computed value of  $\pi$  (should be type `double`).
- (c) Time the execution of the code using the UNIX `time` command, as above, with the likely need to repeat the calculation  $n$  times in an outer loop to obtain  $10 \text{ s} \leq T \leq 60 \text{ s}$ .
- (d) Create a CUDA C version of the code (saving the serial C version) that employs a CUDA C kernel to compute the value of  $\pi$  using the Monte Carlo method. Note that generating random numbers on the GPU device (in the CUDA C kernel) is a challenging task. Also note that only `__device__` functions can be called from within the kernel, so the usual intrinsic random number generator functions in C (which are `__host__` functions) cannot be used to generate random numbers. You will need to define your own `__device__` function that generates random numbers—you can use the `MersenneTwister` CUDA C code from the NVIDIA GPU Computing SDK as a blueprint for writing this `__device__` function to generate random numbers.
- (e) Time the execution of the GPU accelerated code using the UNIX `time` command, as above.