Homework Assignment 2 - GPU computing.

The objective of this assignment is to explore the potential and limitations of parallel processing on graphical processing units (GPUs), in the context of scientific computing. The assignment consists of two parts. In Part 1 you will explore the potential of the GPU in molecular dynamics (MD) simulations. See *e.g.* the wiki page or the book chapter I sent to class on what that is, in general. You do not have to know the details of any of the algorithms for this homework. The simulations use the highly optimized, professionally written AMBER MD code to solve $6N$ simultaneous first order differential equations to calculate the motion of N atoms representing a protein. In your case, the protein has $N \sim 2000$. These types of calculations involve computing the interaction between all pairs of atoms in the protein, which, in general, is an $O(N^2)$ calculation per each step. In other words, the problem is extremely computationally intensive.

In Part 2 you will see some of the limitations of the GPU by implementing and running a very simple vector addition program. There, the computations are much less intense.

1 What to submit

Each group submits just one set; make sure you indicate who did what, along with a rough % effort for each group member.

2 Part I. Molecular Dynamics simulation

Before you log into the suprcomputer, make sure to familiarize yourselves with the basics of UINX shells. In addition, you need to know how to copy, move a file, how to create a directory, how to run a code, set permissions, etc.

Before starting on this task on kuprin supercompter, that is at the beginning of each shell session, read README.

Then, make sure the key environment variables (AMBEHOME, CUDA_HOME, etc. see below) are set correctly. You can do *e.g.* `printenv | grep "CUDA_HOME"` to see what CUDA_HOME is set to. PATH may contain other elements, you just need to make sure it contains all of the ones listed below. For example, it must contain the path to CUDA_HOME, which is "/usr/local/cuda-5.5"

If something is missing, run the following Linux commands to setup your environment (on kuprin supercomputer):

```
export AMBERHOME=/home/shared_utilities/amber14/
export CUDA_HOME=/usr/local/cuda-5.5/
```

```
export MPI_HOME=/usr/lib64/openmpi/
PATH=$PATH:$AMBERHOME/bin:$CUDA_HOME/bin:$MPI_HOME/bin/
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CUDA_HOME/lib64/
export PATH=$PATH:/usr/lib64/openmpi/bin/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PATH:/usr/lib64/openmpi/lib/
```

Once done with setting up your shell, create a HW2 subdirectory in your team directory and copy the following files to it :

```
examples/cuda_md.sh
examples/mpi_md.sh
examples/single_md.sh
examples/README
examples/2trx.top
examples/2trx.crd
examples/mdin
```

Change the device number in the following line in cuda_md.sh to the device number that is assigned to your group or the one that is free, for example:

```
export CUDA_VISIBLE_DEVICES=2
```

Allowed values for the above variable are 2 and 3 ONLY.

2.1 (20 points) Run and benchmark AMBER MD simulations

Use the Linux "time" command to execute and time the cuda_md.sh, mpi_md.sh, and single_md.sh scripts. These shell scripts run 1000 steps of Molecular Dynamics for the thioredoxin protein, using the GPU, 4-CPU processors, and a single CPU processor, respectively.

The "Real" time output by the "time" command represents the total execution time. Compare the execution times for the three runs. Explain the differences. Follow the benchmarking guidelines in the lecture notes. Figure out how to obtain the info for your machine and report it.

3 Part II. Vector addition

Important note: Use cudaSetDevice(2) as the first statement in the GPU program, to execute your code on device 2. Or cudaSetDevice(3) for the other half of the class.

3.1 (10 points) CPU implementation

Code, in C, the vector addition program discussed in class, such that all execution is done on the CPU. The vector length n should be a command line input argument (argv). Use the random number generator (rand) to create the input vectors A and B. Define A, B, and $C = A + B$ as single precision (float) arrays. Output the sum s of all elements in C, $s = \sum_{i=1}^{n} C_i$. You may use as a starting point, the code in the CUDA programmers guide http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

3.2 (10 points) GPU implementation

Modify the above code so that the computation $C = A+B$ is done on the GPU. Setting up the input vectors A and B, and the computation of s should still be done on the CPU as before. You may use, the code in the CUDA programmers guide http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, as a guide.

3.3 (10 points) CPU vs GPU comparison

Use the Linux "time" command to execute the CPU and GPU code for $n = 10^6, 5 \times 10^6, 10^7, 5 \times 10^7$. Verify that the values of s are the same for the CPU and GPU runs. Plot execution time t as a function of n, for the CPU and GPU runs. Fit a line $t = c_1 + nc_2$ to the data and show it on the plot.

3.4 (10 points) Conclusions

Assume that the total computation time is the sum of (1) setup time, (2) computation time, and in the case of the GPU, (3) data transfer time between the CPU and GPU. What do c_1 and c_2, calculated above, represent for the CPU? For the GPU? Now, also assume that the GPU is able to parallelize the computation such that computation time is approximately 0, and that the setup time is the same for the CPU and GPU. Explain the difference in run times between the CPU and GPU. (Hint: c_1 and and c_2 represent two different constants for the CPU and GPU. Refer to class notes on the limitations of the GPU and look at the differences in the CPU and GPU code.)

4 Deliverables

Please submit the following:

1. C code for CPU and GPU.
2. Report in pdf format including multiple plots (see lecture notes) and explanation of the results, *i.e.* comparisons of CPU vs GPU. What is your over-all conclusion as to when one would and would not want to use the GPU?