Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 4 (due Apr. 15, 2019)

Terrence Alsup

1. **Matrix-vector operations on a GPU.**

   The CUDA file `reduction.cu` contains a parallelized dot product and matrix-vector product. The table below shows the memory bandwidth on different GPUs (cuda{1-5}.cims.nyu.edu) and a CPU. Interestingly, the cuda3 compute server was by far the fastest and we obtained significant speed-up for the matrix-vector product. cuda4 and cuda5 compute servers were significantly slower. For the dot product, the problem was small enough that it was more costly to transfer memory between the device and the host than actually performing all of the floating point operations. This is why we do not see any speed-up for this problem.

Processor	Dot Product	Matrix-Vector Product
CPU	1.02	29.47
cuda1	0.10	51.45
cuda2	0.26	95.99
cuda3	0.22	184.26
cuda4	0.05	5.08
cuda5	0.01	3.38

Table 1: The memory bandwidth GB/s of the dot product and matrix-vector product for the CPU and different GPUs. Here $N = 2^{10}$. The CPU reference values were taken when using the cuda1 compute server.

2. **2D Jacobi method on a GPU.**

   The CUDA file `jacobi2D.cu` contains an implementation of the 2D Jacobi method to solve Poisson's equation from Homework 2. The table below shows the timings on different GPUs as well as the CPU result from the OpenMP implementation from Homework 2. We see significant speedup for cuda1, cuda2, and cuda3 compute servers with cuda3 again being the fastest. Also as before, cuda4 and cuda5 are much slower and perform about the same as the CPU with OpenMP.

Processor	Wall-clock time (s)
CPU	17.62
cuda1	0.50
cuda2	0.35
cuda3	0.33
cuda4	14.56
cuda5	21.19

Table 2: Wall-clock time in (s) to perform 10^4 iterations of Jacobi's method when $N = 2^7 = 128$. Note that for the CPU result we actually used $N = 100$. For the CPU with OpenMP implementation 4 threads were used.

3. **Pitch your final project.**

Anya and I will work on parallelizing code from an old research project that implements Kinetic Monte Carlo (KMC) in serial. KMC is an algorithm for modelling the dynamics of a continuous time jump Markov process. The process we will consider is atom movement on the surface of a crystal. In 2D we consider a lattice $\Omega = \{1, \dots, L\} \times \{1, \dots, L\}$. At each site $\vec{\ell} \in \Omega$ on the lattice we have atoms stacked on top of each other. The number of atoms at each site $\vec{\ell}$ will be denoted by the height of the stack $h_{\vec{\ell}}$. The surface of the crystal is then just the atoms at the top of each stack. KMC simulates a Markov jump process describing the evolution of the vector $(h_{\vec{\ell}}(t))_{\vec{\ell} \in \Omega}$ by randomly choosing one lattice site, at which the topmost atom will jump to the top of one of the neighboring stacks. The rates of this process are governed by the height difference between two neighboring stacks. In the serial implementation of KMC, only one jump occurs at a time, and all other atoms, even those far away, must wait before getting a chance to move.

To parallelize KMC we will divide the lattice into sections and then independently run Markov jump processes on each section. After every process has run for a while we will need to synchronize the sections since the jump times are random. We will also need to handle potential conflicts on the boundaries of the sections where atoms may be jumping across. This could potentially be handled with a layer of ghost lattice sites. We plan to transfer the computation to a GPU for the parallelization.

For large values of L, this Markov process can be seen as a discretized version of continuous dynamics (described by a PDE) governing the evolution of a smooth surface $h(x, t)$. This PDE is known and solutions to it can be computed numerically. To test our implementation, we can evolve a continuous height profile forward for a certain amount of time using the PDE, evolve a discretization of that profile for the same amount of time using parallel KMC, and compare the two resulting profiles.