High Performance Computing: Homework 4

Paul Beckman

1 Matrix-vector operations on a GPU

Computing the matrix vector product between $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ and $\boldsymbol{b} \in \mathbb{R}^N$ for $N = 2^{13}$ gives the following timings using `OMP_NUM_THREADS = 8` for both machines.

On `cuda2` with `Intel Xeon E5-2660 (2.60 GHz)` CPU and `GeForce RTX 2080 Ti` GPU:

```
CPU: 0.292824 s
CPU Bandwidth = 5.500280 GB/s

GPU: 0.007551 s (0.091952 s total)
GPU Bandwidth = 17.515802 GB/s
```

and on `cuda5` with `Intel Xeon E5-2650 (2.60 GHz)` CPU and `GeForce GTX TITAN Z` GPU:

```
CPU: 0.431621 s
CPU Bandwidth = 3.731541 GB/s

GPU: 0.006431 s (0.195043 s total)
GPU Bandwidth = 8.257744 GB/s
```

where the total time includes memory transfer to and from the GPU. We see in both cases that the GPU gives significant speedups, and that the data transfer totally dominates the runtime.

2 2D Jacobi method on a GPU

Again using `cuda2` with `Intel Xeon E5-2660 (2.60 GHz)` CPU and `GeForce RTX 2080 Ti` GPU and `OMP_NUM_THREADS = 8`, we observe the following timings for 100 iterations of Jacobi

N	CPU time	GPU time	GPU total
128	1.020e-02	4.675e-01	4.677e-01
256	2.839e-02	4.746e-01	4.751e-01
512	8.995e-02	4.929e-01	4.940e-01
1024	2.770e-01	5.299e-01	5.335e-01
2048	1.104e+00	6.118e-01	6.256e-01
4096	4.228e+00	8.003e-01	8.567e-01
8192	1.751e+01	1.942e+00	2.172e+00

We note that the CPU obtains superior performance for small N, but for $N > 1024$ the GPU is faster. Memory transfer is a much smaller fraction of the total computation time here than in the matvec above, likely because of the need to synchronize the GPU within each iteration, which slows down the body of the computation significantly.

3 Update on final project

We have implemented a serial version of the Barnes-Hut multipole algorithm built on a tree data structure. Some debugging remains, but it is almost complete. We are realizing that this tree-based implementation may prove difficult to parallelize due to its reliance on recursive function calls and pointer linkages. So after we finish a basic OpenMP implementation of our tree-based, we may experiment with developing an alternative implementation based on array data structures for more efficient parallelization.