CS267 Assignment 0

Alejandro Francisco Queiruga

February 7, 2013

Bio

I am a second year Ph.D. in Mechanical Engineering. My research is focused on writing simulations of high speed impacts with electromagnetically sensitive ballistic fabrics. In my code, the fabric is modeled as an interconnected network of yarn segments so that a high level detail is obtained to capture the rupture of individual yarns during the impact with a projectile. However, this discretization results in a large amount of degrees of freedom. As a result of the class, I would like to parallelize my model so that I can simulate much larger systems.

Application

In “Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA” by Komatitsch et al., a spectral finite element code is parallelized to a GPU system[1]. While the authors’ original code is already parallelized using OpenMP and MPI, they start from a serial implementation to simplify the development process in CUDA, and to have a benchmark to compare to.

Because the nodes in a finite element mesh are interconnected, neighboring elements share degrees of freedom and thus write to the same parts of memory. While it is natural to write finite element codes as operations on elements, parallelizing this process blindly will result in a race condition during the assembly procedure. The authors solve the problem of multiple threads simultaneous writing to the same region of memory by using a graph-coloring procedure, depicted in Figure 1, reproduced from their paper. The element mesh is colored into disjoint sets, where all elements of one color do not share degrees of freedom with elements of the same color. Instead of performing the parallelism over all of the elements at once, only threads of a single color are operated at a time, with thread syncing occurring between colors. After the sets are initially made, they may contain widely different amounts of elements, so the authors regroup subsets of the colors to form more equally-sized sets to improve the load balancing of the simulation.

An issue that arrises with GPUs is the (relatively) limited amount of memory available, restricting the maximum problem size that can fit on the chip. The authors implemented two versions of their CUDA algorithm, one for the case where the entire problem fits onto the GPU, and the other for the case where the problem needs to be broken up and transferred back-and-forth between main memory and GPU memory

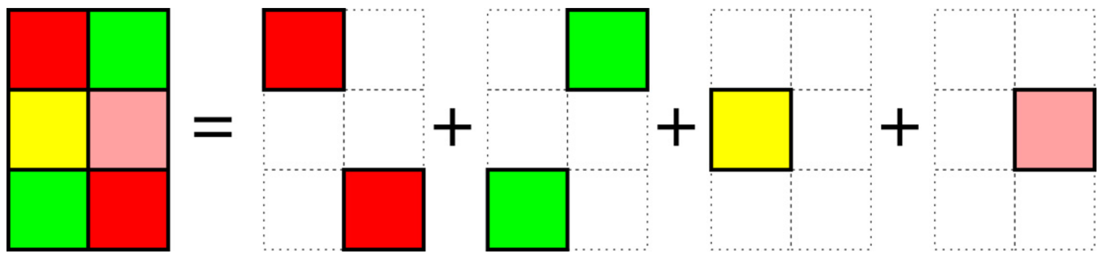

Figure 1: Illustration from [1] illustrating coloring algorithm to divide mesh into sets of disjoint elements to be computed in parallel.

during each iteration. The authors' code achieved an up to 25× speed up over their serial implementation when the problem fit onto the GPU's memory. On the same sized problem, the version that could handle larger problems only achieved a 7× speed up, as they noted at 75% of the time was spent copying data back and forth.

References

[1] D. Komatitsch, D. Michéa, and G. Erlebacher. Porting a high-order finite-element earthquake modeling application to nvidia graphics cards using cuda. *Journal of Parallel and Distributed Computing*, 69(5):451–460, 2009.