Assembly of Finite Element Methods on Graphics Processors

C. Cecka*, E. Darve†, A. Lew†

*Computational and Mathematical Engineering
Stanford University
e-mail: ccecka@stanford.edu

†Mechanical Engineering
Stanford University
e-mail: lewa@stanford.edu, darve@stanford.edu

ABSTRACT

Recently, graphics processing units (GPUs) have had great success in accelerating many numerical computations. We present their application to computations on unstructured meshes such as those in finite element methods. Multiple approaches in assembling and solving sparse linear systems with NVIDIA GPUs and the Compute Unified Device Architecture (CUDA) are presented. Multiple strategies for efficient use of global, shared, and local memory, methods to achieve memory coalescing, the optimal choice of parameters, and the appropriate level of parallelism to choose are discussed.

In the basic approach that mimics the serial algorithm, threads compute data for an element and then scatter the results to the matrix and forcing vector of the linear system of equations. We also considered algorithms in which threads are primarily in charge of computing a single non-zero entry or an entire row in the matrix. We discuss the pros and cons of each technique and in particular how they can efficiently make use of the different hardware resources.

The more successful approaches are compared using a two-dimensional benchmark case on an NVIDIA GeForce 8800 GTX, an NVIDIA Tesla C1060, and a single core of an Intel Core 2 Quad CPU Q9450 2.66 GHz. The performance is analyzed as a function of the problem size (number of mesh nodes), and the order of the finite-element method. We find that with appropriate preprocessing and arrangement of support data, the GPU coprocessor achieves speedups of 30 or more in comparison to a well optimized serial implementation. We also find that the optimal assembly strategy depends on the order of polynomials used in the finite-element discretization.

Thus, the assembly, solution, and visualization of a dynamic FEM problem can be performed completely on the GPU, avoiding expensive data transfers between the host device and the GPU coprocessor. This is especially interesting for real-time visualization in graphics, design software, simulation, and gaming.