1
Introduction

Fracture, branching, and fragmentation simulations of large-scale finite element meshes have a broad range of engineering applications. To achieve realistic and more accurate results, there is a need to employ highly discretized models, thus requiring a large amount of computational resources. In order to accelerate finite element analysis, current parallel environments are based on distributed memory architectures. In this scenario, each processor (or small group of processors) of a computing node has private access to a region of the global system memory. Processors on different nodes communicate among themselves by sending messages over a network. Different parallel finite element systems with support for distributed mesh representation have been proposed (25, 11). Parallel fracture, microbranching, and fragmentation simulation using the extrinsic cohesive zone model (6) requires that cohesive elements are adaptively inserted along the simulation. This increases the challenge for parallelization, since mesh consistency must be ensured among partitions (7).

In this work, we focus on the use of many-core architectures, such as the one provided by modern graphics processor units (GPU), for accelerating fracture, microbranching, and fragmentation simulations based on the extrinsic cohesive zone model (as far as we know, this is the first proposal of a complete adaptive finite element analysis running on the GPU). During the last years, general purpose computing on graphics processor units (GPGPU) has proved to be an efficient and powerful mean to accelerate expensive numeric simulations and algorithms that require large amount of input data. GPUs are massively multithreaded many-core chips and they are suited for excessive numeric computations with high arithmetic intensity, which is the case of finite element analysis. However, mapping the CPU version of an extrinsic cohesive fragmentation simulation to the GPU is not immediate or trivial. Several challenges emerge, such as algorithm parallelization, high-performance memory access, concurrency, and device architecture dependent factors.

We investigate and describe mapping and parallelization techniques for two-dimensional fracture, branching, and fragmentation simulation of finite element meshes on a GPU using NVIDIA's CUDA (Compute Unified Device Architecture) framework . We propose a simple but effective data structure for performing all data-parallel computations and algorithms for 2D models using triangle meshes (but an extension to tetrahedron elements is straightforward). The previously established coloring method for FEM meshes is also used to minimize concurrency. Parallel techniques are presented for the numeric analysis code and for updating the FEM mesh when cohesive elements are inserted during the course of the simulation. As a result, we are able to speedup the simulation by a factor close to 30, when compared to the serial code running on a single CPU processor. In our experiments, we have employed a two-dimensional microbraching analysis, but the created framework for parallel simulation can support other separation phenomena simulations.

The research is organized as follows. Chapter 2 reviews related work. Chapter 3 discusses the CUDA and GPU architecture, and how we can take advantage of many-core devices to improve performance. Chapter 4 briefly reviews simulations based on the extrinsic cohesive zone model, discussing the requirements for adaptive insertion of cohesive elements. Chapter 5 presents the proposed (simple) topological data structure to support mesh modification on the GPU. Chapter 6, conceptually the main section of this paper, discusses the parallelization of the fragmentation simulation itself, including the parallel algorithm for inserting cohesive elements. Kernel optimizations are also presented, bearing in mind memory and flow optimizations that lead to performance boosts. We also discuss the use of mesh coloring and its impact on the simulation's parallelization performance and concurrency issues. Chapter 7 presents the achieved results, and, finally, concluding remarks and directions of future work are drawn in Chapter 8.