

**Andrei Alhadef Monteiro**

## **Many-core Fragmentation Simulation**

### **DISSERTAÇÃO DE MESTRADO**

Dissertation presented to the Postgraduate Program in Informatics, of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Waldemar Celes Filho

Rio de Janeiro  
August 2011



**Andrei Alhadeff Monteiro**

## **Many-core Fragmentation Simulation**

Dissertation presented to the Postgraduate Program in Informatics, of the Departamento de Informática do Centro Técnico Científico da PUC-Rio, as partial fulfillment of the requirements for the degree of Mestre.

**Prof. Waldemar Celes Filho**

Advisor

Departamento de Informática — PUC-Rio

**Prof. Marcelo Gattass**

Departamento de Informática — PUC-Rio

**Prof. Diego Fernandes Nehab**

Instituto Nacional de Matemática Pura e Aplicada (IMPA)

**Prof. Ivan Fabio Mota de Menezes**

Tecgraf — PUC-Rio

**Prof. José Eugenio Leal**

Coordinator of the Centro Técnico Científico — PUC-Rio

Rio de Janeiro, August 17, 2011

All rights reserved.

### **Andrei Alhadeff Monteiro**

Andrei Alhadeff Monteiro graduated from Pontifícia Universidade Católica do Rio de Janeiro in Computer Engineering. He then obtained a Master degree at Pontifícia Universidade Católica do Rio de Janeiro in Computer Science, acting in the areas of physics animation and engineering simulation together with GPU programming. While doing his Masters, he worked as a researcher at Tecgraf PUC-Rio with reservoir simulation and rendering.

#### Bibliographic data

Monteiro, Andrei Alhadeff

Many-core Fragmentation Simulation / Andrei Alhadeff Monteiro ; advisor: Waldemar Celes Filho. — 2011.

59 f. : il. ; 30 cm

Dissertação (Mestrado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2011.

Inclui bibliografia

1. Informática – Dissertação. 2. Simulação de fragmentação. 3. Múltiplos processadores. 4. CUDA. 5. Método dos elementos finitos. 6. Elementos coesivos. I. Filho, Waldemar Celes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

## Acknowledgments

To my family, for all the support they have given throughout my life. To my father, Ivan de Castro Monteiro, my mother, Myriam Alhadef Monteiro, and my sister, Camila Alhadef Monteiro.

To my adviser, Waldemar Celes, without whom the research would not be possible. Thank you for motivating me throughout these whole years as my adviser and teacher.

To Tecgraf/PUC-Rio laboratory, for giving me opportunity to face such challenges and learning with them.

To all my friends for their support and friendship.

## Resumo

Monteiro, Andrei Alhadef; Filho, Waldemar Celes. **Implementação de simulação de fragmentação em arquitetura de multi-processadores**. Rio de Janeiro, 2011. 59p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Apresentamos um método computacional na GPU que lida com eventos de fragmentação dinâmica, simulados por meio de elementos de zona coesiva. O trabalho é dividido em duas partes. Na primeira parte, tratamos o pré-processamento de informações e a verificação de corretude e eficácia da inserção dinâmica de elementos coesivos em malhas grandes. Para tal, apresentamos uma simples estrutura de dados topológica composta de triângulos. Na segunda parte, o código explícito de dinâmica é apresentado, que implementa a formulação extrínica de zona coesiva, onde os elementos são inseridos dinamicamente quando e onde forem necessários. O principal desafio da implementação na GPU, usando a formulação de zona coesiva extrínica, é ser capaz de adaptar dinamicamente a malha de uma forma consistente, inserindo elementos coesivos nas facetas fraturadas. Para isso, a estrutura de dados convencional usada no código de elementos finitos (baseado na incidência de elementos) é estendida, armazenando, para cada elemento, referências para elementos adjacentes. Para evitar concorrência ao acessar entidades compartilhadas, uma estratégia convencional de coloração de grafos é adotada. Na fase de pré-processamento, cada nó do grafo (elementos na malha) é associado a uma cor diferente das cores de seus nós adjacentes. Desta maneira, elementos da mesma cor podem ser processados em paralelo sem concorrência. Todos os procedimentos necessários para a inserção de elementos coesivos nas facetas fraturadas e para computar propriedades de nós são feitas por threads associados a triângulos, invocando um kernel por cor. Computações em elementos coesivos existentes também são feitas baseadas nos elementos adjacentes.

## Palavras-chave

Simulação de fragmentação; Múltiplos processadores; CUDA; Método dos elementos finitos; Elementos coesivos;

## Abstract

Monteiro, Andrei Alhadef; Waldemar Celes (Advisor). **Many-core Fragmentation Simulation**. Rio de Janeiro, 2011. 59p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A GPU-based computational framework is presented to deal with dynamic failure events simulated by means of cohesive zone elements. The work is divided into two parts. In the first part, we deal with pre-processing of the information and verify the effectiveness of dynamic insertion of cohesive elements in large meshes. To this effect, we employ a simplified topological data structured specialized for triangles. In the second part, we present an explicit dynamics code that implements an extrinsic cohesive zone formulation where the elements are inserted on-the-fly, when needed and where needed. The main challenge for implementing a GPU-based computational framework using extrinsic cohesive zone formulation resides on being able to dynamically adapt the mesh in a consistent way, inserting cohesive elements on fractured facets. In order to handle that, we extend the conventional data structure used in finite element code (based on element incidence) and store, for each element, references to the adjacent elements. To avoid concurrency on accessing shared entities, we employ the conventional strategy of graph coloring. In a pre-processing phase, each node of the dual graph (bulk element of the mesh) is assigned a color different to the colors assigned to adjacent nodes. In that way, elements of a same color can be processed in parallel without concurrency. All the procedures needed for the insertion of cohesive elements along fracture facets and for computing node properties are performed by threads assigned to triangles, invoking one kernel per color. Computations on existing cohesive elements are also performed based on adjacent bulk elements.

## Keywords

Fragmentation simulation; Many-core; CUDA; Finite element method; Cohesive elements;

# Contents

1	Introduction	<b>12</b>
2	Related Work	<b>14</b>
3	CUDA and GPU Concepts	<b>15</b>
3.1	GPU Architecture	15
3.2	Optimization	16
4	Fragmentation Simulation	<b>18</b>
4.1	Simulation definitions	18
4.2	Pre-processing and updating	19
4.3	Stresses	20
4.4	Insertion of cohesive elements	21
4.5	Internal and cohesive forces	22
4.6	Node and element update	23
5	Data Structure	<b>25</b>
5.1	Retrieving adjacency relationship	27
6	Parallel Implementation	<b>29</b>
6.1	Coloring model	29
6.2	Pre-processing and update	31
6.3	Stresses	33
6.4	Insertion of cohesive elements	35
6.5	Internal Forces	37
6.6	Cohesive forces and simulation outcome	39
6.7	Overview	41
7	Experimental Results	<b>42</b>
7.1	Insertion of cohesive elements	42
7.2	Fragmentation simulation	44
8	Conclusion	<b>49</b>
9	Bibliography	<b>52</b>
A	Optimized insertion of cohesive elements	<b>55</b>

## List of Figures

3.1	Diagram of a G80 architecture with 16 SMs and 128 SPs, based on the figures presented in (17).	16
3.2	CUDA memory hierarchy, based on the figures presented in (17).	16
4.1	T3 mesh attributes belonging to the simulation.	18
4.2	Cohesive element insertion algorithm on a T3 mesh. (1) Mesh with initial facets that need to be fractured. Elements belonging to each node are traversed and cohesive element is inserted but no node is duplicated. (2, 3) The other fractured facet is checked for node duplication, the cohesive element is inserted and the node is marked as needing duplication. (4) Node is duplicated by traversing through the elements and updating the node index of the node belonging to them.	22
4.3	Node update algorithms. Incident elements traversal, or gather (1) and element sweep, or scatter (2).	24
5.1	Mesh parameters data structure of a T6 mesh.	26
5.2	Simulation parameters data structure diagram of FEM model. Global memory is used for attributes that change throughout the simulation. Texture memory is used for attributes that are constant during the entire simulation, but occupy too much memory space. Constant memory is used for attributes that are constant during the entire simulation, but are common to all elements and node, therefore requiring few memory space.	27
5.3	Traversal algorithm from a given element node using the proposed data structure.	28
6.1	(1) Bulk elements are re-arranged in color groups (preferable balanced) and the same kernel per color group is called to avoid writing conflicts. (2) Example of a colored T6 structured mesh (3) and using the colored mesh to update nodal masses of the group of elements in the current color in parallel.	31
6.2	Fracture and fragmentation simulation loop.	33
6.3	Splitting the kernel that computes stress and strain into simpler kernels.	34
6.4	To accumulate the stresses and strains on the nodes, we launch 12 threads per element, where each thread will accumulate part of the stress and strain matrices by fetching from the element shape functions and from the stress and strain at the Gauss points.	35



6.5	Cohesive elements insertion on a T3 mesh. (1) Mesh with initial cracks and facets that need to be fractured. Coloring is used to avoid duplicating nodes of elements that share nodes in parallel. (2) From each facet node belonging to the element in the current color group, the algorithm traverses through its incident elements. (3) Nodes that need duplication. (4) T3 mesh with final node duplications and new cracks and cohesive elements. The fractured facets from the next color group are checked for cohesive elements insertion.	36
6.6	When computing internal forces, a thread per stiffness matrix line is launched using the color model. In this example, two elements per block is used.	39
6.7	Splitting the kernel that computes cohesive forces into simpler kernels.	40
7.1	T6 disc mesh used to test insertion of cohesive element decoupled from analysis code (left). T6 bar mesh used to test the fracture and fragmentation simulation (right).	42
7.2	Time for cohesive elements insertion of a T6 mesh.	43
7.3	Speedup for cohesive elements insertion of a T6 mesh.	44
7.4	2D model of a rectangular specimen with initial notch of 2 mm. Initial strain is 0.015, with node thickness of 1 mm. Model dimensions are 16mm per 4mm.	45
7.5	T6 FEM mesh with 36,864 bulk elements at the end of the fragmentation simulation.	46
7.6	Refined T6 FEM mesh with 147,456 bulk elements at the end of the fragmentation simulation.	46
7.7	Principal stress evolution with crack propagation.	47
7.8	Execution time for each kernel relative to the entire simulation time for a T6 mesh with 36,864 bulk elements.	48
7.9	Kernels' average time for the simulation for a T6 mesh with 36,864 bulk elements.	48
8.1	3D view of fragmented 2-dimensional bar with 74,257 nodes and 36,864 bulk elements.	50
8.2	Crack propagation on a 2-dimensional bar with 74,257 nodes and 36,864 bulk elements.	51
A.1	Getting part of the new node index for each thread node counter offset inside the block. This value is added to the current node counters from each block.	56
A.2	Getting part of the new node index from current block node counters.	57
A.3	Cohesive elements insertion time for T6 meshes using atomic functions in global or shared memory.	59
A.4	Cohesive elements insertion speedup for T6 meshes using atomic functions in global or shared memory.	59

## List of Tables

4.1	Fragmentation algorithm	19
6.1	Kernel subroutine call algorithm using mesh coloring	30
6.2	Parallel Fracture Algorithm	32
6.3	Parallel Node Duplication Algorithm	37
7.1	Results for insertion of cohesive elements decoupled from analysis code.	43
7.2	Simulation and mesh parameters for a T6 mesh and its refined version.	45
7.3	Simulation and mesh parameters and results (GPU speedup and GPU and CPU time) for a T6 mesh and its refined version.	46
A.1	Node index retrieving and appending using shared memory when inserting cohesive elements.	58
A.2	Mesh attributes performance results for T6 disc mesh [7.1] and its refined versions.	58

*If music be the food of love, play on,  
Give me excess of it; that surfeiting,  
The appetite may sicken, and so die.*

**William Shakespeare, *Twelfth Night Act 1.***