9
Bibliography

[1] CECKA, C.; LEW, A. J. ; DARVE, E. **International Journal for Numerical Methods in Engineering**. Assembly of finite element methods on graphics processors, journal, 2010.

[2] CELES, W.; PAULINO, G. H. ; ESPINHA, R. **Journal of Computing and Information Science in Engineering**. Efficient handling of implicit entities in reduced mesh representations, journal, v.5, n.4, p. 348–359, 2005.

[3] CELES W, PAULINO GH, E. R. **Int J Numer Methods Eng**. A compact adjacency-based topological data structure for finite element mesh representation, journal, v.64, p. 15291565, 2005.

[4] CIRAK, F.; ORTIZ, M. ; PANDOLFI, A. **Computer Methods in Applied Mechanics and Engineering**. A cohesive approach to thin-shell fracture and fragmentation, journal, v.194, n.21-24, p. 2604 – 2618, 2005. Computational Methods for Shells.

[5] DE SOUZA LIMA ESPINHA, R. **Suporte topológico em paralelo para malhas de elementos finitos em análises dinâmicas de fratura e fragmentação**. 2011. PhD thesis - PUC-Rio.

[6] DOOLEY, I.; MANGALA, S.; KALE, L. ; GEUBELLE, P. **J. Sci. Comput.** Parallel simulations of dynamic fracture using extrinsic cohesive elements, journal, v.39, p. 144–165, April 2009.

[7] ESPINHA, R.; CELES, W.; RODRIGUEZ, N. ; PAULINO, G. **Engineering with Computers**. Partops: compact topological framework for parallel fragmentation simulations, journal, v.25, p. 345–365, 2009. 10.1007/s00366-009-0129-2.

[8] GEBREMEDHIN, A. H.; MANNE, F.; MANNE, G. F. ; OPENMP, P. **Scalable parallel graph coloring algorithms**, 2000.

[9] GÖDDEKE, D.; STRZODKA, R. ; TUREK, S. **Accelerating double precision FEM simulations with GPUs**. In: PROCEEDINGS OF ASIM 2005 - 18TH SYMPOSIUM ON SIMULATION TECHNIQUE, Sep 2005.

[10] G.T., C.; M., O. **International Journal of Solids and Structures**. Computational modelling of impact damage in brittle materials, journal, v.33, n.20, p. 2899–2938, 1996.

[11] KIRK, B. S.; PETERSON, J. W.; STOGNER, R. H. ; CAREY, G. F. **Eng. with Comput.** libmesh: a c++ library for parallel adaptive mesh refinement/coarsening simulations, journal, v.22, p. 237–254, December 2006.

[12] KIRK, D. B.; HWU, W.-M. W. **Programming Massively Parallel Processors: A Hands-on Approach**. 1st. ed., San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[13] KLEIN, P. A.; FOULK, J. W.; CHEN, E. P.; WIMMER, S. A. ; GAO, H. J. **Theoretical and Applied Fracture Mechanics**. Physics-based modeling of brittle fracture: cohesive formulations and the application of meshfree methods, journal, v.37, n.1-3, p. 99 – 166, 2001.

[14] KOMATITSCH, D.; MICHÉA, D. ; ERLEBACHER, G. **J. Parallel Distrib. Comput.** Porting a high-order finite-element earthquake modeling application to nvidia graphics cards using cuda, journal, v.69, p. 451–460, May 2009.

[15] LAWLOR, O.; CHAKRAVORTY, S.; WILMARTH, T.; CHOUDHURY, N.; DOOLEY, I.; ZHENG, G. ; KALÉ, L. **Engineering with Computers**. Parfum: a parallel framework for unstructured meshes for scalable dynamic physics applications, journal, v.22, p. 215–235, 2006. 10.1007/s00366-006-0039-5.

[16] MICIKEVICIUS, P. **3D finite difference computation on GPUs using CUDA**. In: Kaeli, D. R.; Leeser, M., editors, PROCEEDINGS OF 2ND WORKSHOP ON GENERAL PURPOSE PROCESSING ON GRAPHICS PROCESSING UNITS, ACM International Conference Proceeding Series, p. 79–84. ACM, 2009.

[17] **CUDA C Programming Guide 3.2**, Nov. 2010.

[18] ORTIZ, M.; PANDOLFI, A. **International Journal for Numerical Methods in Engineering**. Finite-deformation irreversible cohesive elements for three-dimensional crack-propagation analysis, journal, v.44, n.9, p. 1267–1282, 1999.

[19] PANDOLFI, A.; ORTIZ, M. **Engineering with Computers**. Solid modeling aspects of three-dimensional fragmentation, journal, v.14, p. 287–308, 1998. 10.1007/BF01201761.

[20] PARK, K.; PAULINO, G. H. ; ROESLER, J. R. **Journal of the Mechanics and Physics of Solids**. A unified potential-based cohesive model of mixed-mode fracture, journal, v.57, n.6, p. 891 – 908, 2009.

[21] PAULINO, G. H.; CELES, W.; ESPINHA, R. ; ZHANG, Z. J. **Eng. with Comput.** A general topology-based framework for adaptive insertion of cohesive elements in finite element meshes, journal, v.24, p. 59–78, January 2008.

[22] RADOVITZKY, R.; SEAGRAVES, A.; TUPEK, M. ; NOELS, L. **Computer Methods in Applied Mechanics and Engineering**. A scalable 3d fracture and fragmentation algorithm based on a hybrid, discontinuous galerkin, cohesive element method, journal, v.200, n.1-4, p. 326 – 344, 2011.

[23] RODRIGUEZ-NAVARRO, J.; SUSIN, A. **Eurographics**. Non structured meshes for cloth gpu simulation using fem, journal, p. 1–7, 2006.

[24] SANDERS, J.; KANDROT, E. **CUDA by Example: An Introduction to General-Purpose GPU Programming**. 1. ed., Addison-Wesley Professional, July 2010.

[25] SEOL, E. S.; SHEPHARD, M. S. **Engineering With Computers**. Efficient distributed mesh data structure for parallel automated adaptive analysis, journal, v.22, p. 197–213, 2006.

[26] WELSH, D. J. A.; POWELL, M. B. **The Computer Journal**. An upper bound for the chromatic number of a graph and its application to timetabling problems, journal, v.10, n.1, p. 85–86, 1967.

[27] ZHANG, Z.; PAULINO, G. H. ; CELES, W. **International Journal for Numerical Methods in Engineering**. Extrinsic cohesive modelling of dynamic fracture and microbranching instability in brittle materials, journal, v.72, p. 893–923, 2007.

[28] ZHOU, F.; MOLINARI, J. F. **International Journal for Numerical Methods in Engineering**. Dynamic crack propagation with cohesive elements: a methodology to address mesh dependency, journal, v.59, n.1, p. 1–24, 2004.

A
Optimized insertion of cohesive elements

To optimize the cohesive elements insertion and node duplication, and also avoid writing on the same global memory address using the atomic operation, we tested a new strategy where each thread block will have its own node counter. Each thread within the block is responsible for updating the node counter that resides in shared memory, and only one thread in the block (not necessarily the first) will be responsible for accumulating the shared counter on the global counter residing in global memory. The advantages of this strategy are that writing in shared memory is much faster and fewer number of threads will be updating the same global memory address simultaneously as well as few threads updating the same node counter in shared memory. Retrieving the new node index is done by using the atomic functions return values. For each thread, when adding one to the block counter in shared memory, we retrieve the number of block's duplicated nodes until that moment, representing the node index offset within the block. Threads are then synchronized. One thread in each block adds its block counter to the global node counter and the atomic function returns the number of nodes immediately before the sum. This result represents the current number of nodes for all of the other blocks. Threads are synchronized, and adding the atomic intrinsic result from global counter with the shared counter index will give the current new node index for this thread. This strategy is best taken use for when many nodes are duplicated (such as test case reported in Section 7.1). In actual simulation, however, cohesive element insertion is checked at a number of steps in which few nodes are duplicated, so the increase in performance is unremarkable. Figure A.1 illustrates the algorithm for retrieving the node index inside the current block. Using the atomic function to accumulate the shared memory counter when accumulating the number of nodes inside the current block gives us the node offset inside the block. Figure A.2 illustrates retrieving the node index offset from all other blocks. Using the atomic function to add the current number of nodes in the global counter with the current number of nodes inside the block (stored in the shared node counter) gives us the current node offset for this block. Adding it with the current node offset inside the block gives the new node index. Table A.1 presents the algorithm.

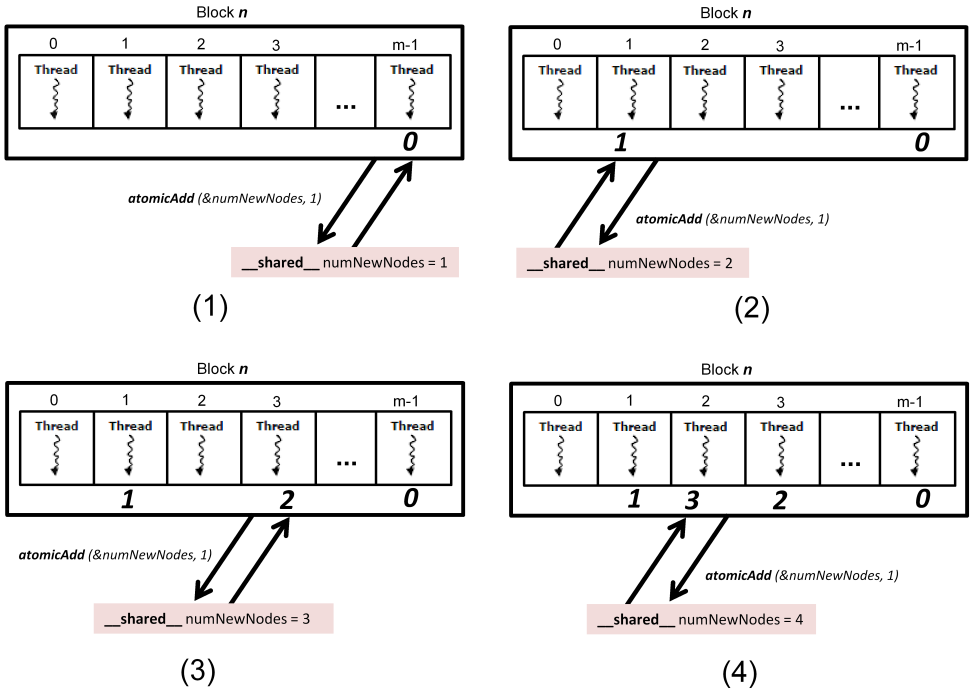

Figure A.1: Getting part of the new node index for each thread node counter offset inside the block. This value is added to the current node counters from each block.

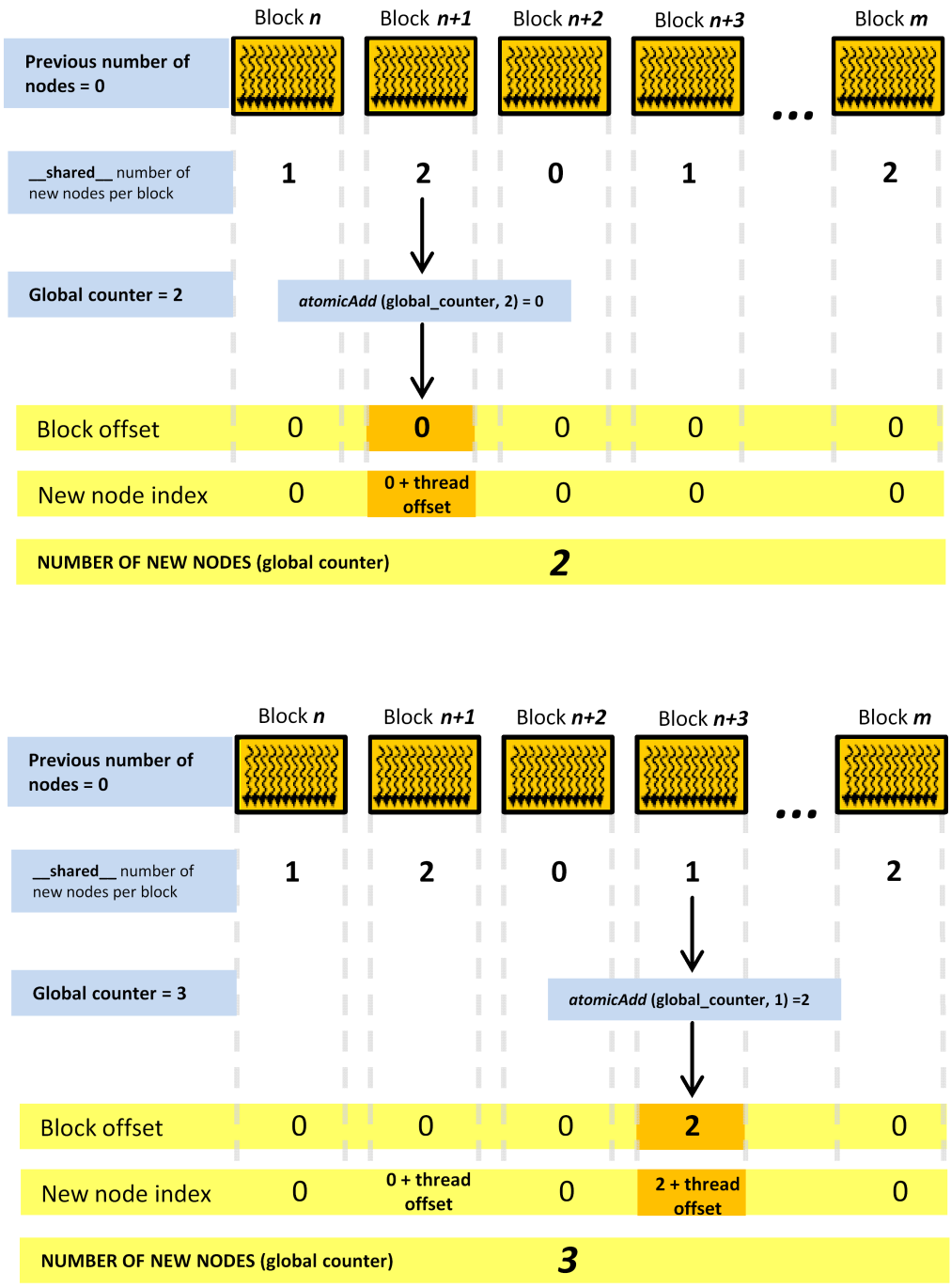

Figure A.2: Getting part of the new node index from current block node counters.

```
1: n ← Node to duplicate
2: if first thread then
3:     Number of New Nodes Per Block ← 0
4:     Block Offset ← 0
5: end if
6: syncthreads
7: Node Thread Offset ← atomicAdd(Number of New Nodes Per Block, 1)
8: syncthreads
9: if first thread then
10:    Block Offset ← atomicAdd(Global Node Counter, Number of New Nodes
       Per Block)
11: end if
12: syncthreads
13: New Node Index ← Block Offset + Node Thread Offset
14: NodeArray[New Node Index] = n
```

Table A.1: Node index retrieving and appending using shared memory when inserting cohesive elements.

To test the optimized insertion of cohesive elements, we fractured 5 to 5 percent of the facets (as discussed in Section 7.1) and launched a kernel for each color. To color the mesh, we used the Welsh Powell algorithm (26). It is important to highlight the performance boost when using a node counter in shared memory for each block. The results below show that the speedup rose, indicating that thousands to millions of threads updating the same memory address simultaneously is a bottleneck, and that shared memory's fast access as well as few threads updating the same address can be a useful technique when duplicating nodes. Table A.2 shows the GPU results for T6 disc mesh and its refined versions compared to the CPU results, as well as mesh attributes before and after the simulation. Graphs A.3 and A.4 compare the GPU speedup and time of using and not using atomic functions in shared memory when inserting cohesive elements in T6 disc mesh and its refined versions.

Bulk elements	Initial nodes	Final nodes	CZ elements	CPU Time (s)	GPU Time (s)	Speedup
240,000	481,200	1,440,000	359,400	9.29	0.0363	255.9
960,000	1,922,400	5,760,000	1,438,800	36.946	0.0778	474.9
2,160,000	4,323,600	12,960,000	3,238,200	84.94	0.1161	731.6
3,840,000	7,684,800	23,040,000	5,757,600	150.04	0.1761	852.0

Table A.2: Mesh attributes performance results for T6 disc mesh [7.1] and its refined versions.

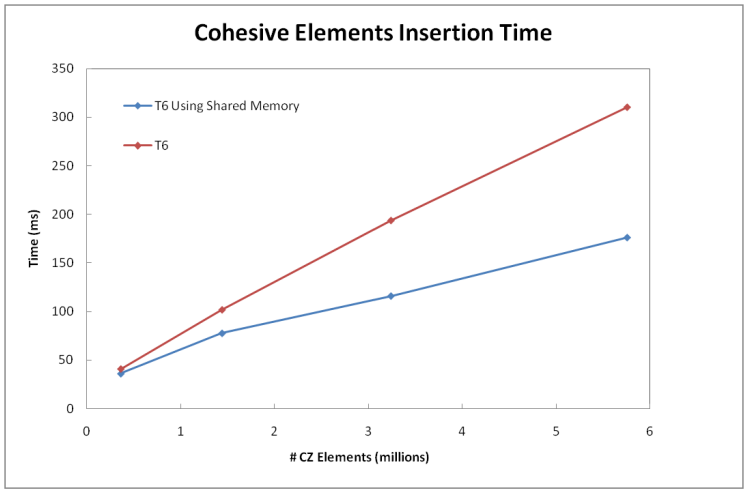

Figure A.3: Cohesive elements insertion time for T6 meshes using atomic functions in global or shared memory.

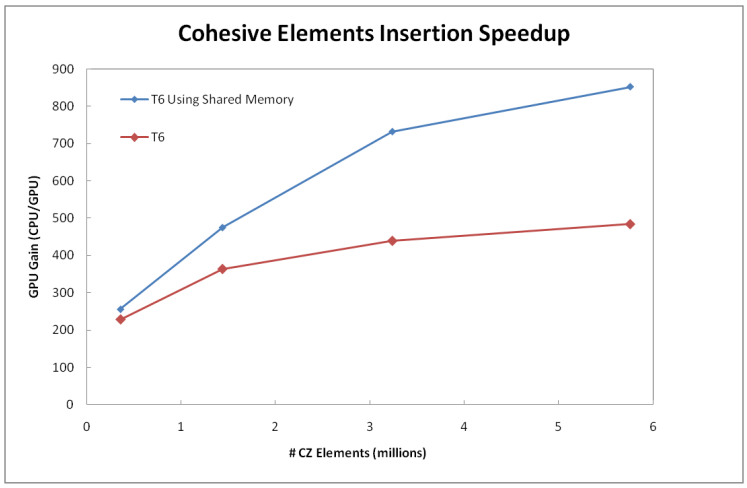

Figure A.4: Cohesive elements insertion speedup for T6 meshes using atomic functions in global or shared memory.