Out-of-Core Parallel Delaunay Mesh Generation [*]

Extended Abstract

Andriy Kot, Andrey N. Chernikov and Nikos P. Chrisochoides
Computer Science Department
College of William and Mary
Williamsburg, VA 23185

Abstract *We present an extension of an existing Parallel Delaunay Refinement (PDR) method to be used for out-of-core computing, thus allowing it to generate large meshes using limited computing resources. We base our work on a shared memory implementation of the PDR method. Also, we are designing a distributed out-of-core algorithm which would allow to compute very large meshes using supercomputers and COWs with limited resources.*

Introduction

COWs and contemporary supercomputers provide large aggregate memory and computing power. However, many applications do not require both types of resources in equal proportions. Moreover, it is not unusual for memory-intensive applications (e.g., mesh generation and refinement) to use hundreds of nodes to utilize their memory rather than CPUs. We see the solution in an out-of-core approach which already demonstrated its effectiveness for wide variety of applications [1, 2, 3, 4, 5], including out-of-core mesh generation [6, 7].

Here, we propose an extension to recently developed Parallel Delaunay Refinement (PDR) method to allow it to be used for generating large out-of-core meshes.

Parallel Delaunay Refinement

The Parallel Delaunay Refinement method is based on a theoretical framework for constructing guaranteed quality Delaunay meshes in parallel [8]. The sequential Delaunay refinement algorithms insert points at the circumcenters of triangles of poor quality or unacceptable size. Two points are called Delaunay-independent [9] iff they can be inserted concurrently without destroying the conformity and Delaunay properties of the mesh. The method provides a sufficient condition of Delaunay-independence, which is based on the distance between points, i.e. two points are Delaunay-independent if the distance between them is no less than $4\bar{r}$, where $\bar{r}$ is an upper bound on triangle circumradius in the initial mesh. This condition allows to avoid using expensive coloring techniques. Its efficient implementation is based on the use of a coarse auxiliary lattice, which is imposed over the triangulation domain in such a way that the circumcenters in non-adjacent cells are a-priori Delaunay-independent. Processors are logically arranged into a two-dimensional grid, and each processor is assigned some subset of cells for refinement. The parallel meshing in [8] is implemented by simultaneously shifting the refinement focus by all processors, so that specific buffer cells serve to separate the refinement zones. After every refinement iteration, the triangles in buffer cells are exchanged between neighboring processors and are used in subsequent refinement steps. Data exchange is organized by shifting cells along vertical, horizontal, and diagonal directions.

# of processors	Time, sec. MPI	Time, sec. OpenMP	# of elements, $\times 10^6$
4	220.3	214.1	14.6

Table 1: Pipe cross-section, distributed (MPI) and shared (OpenMP) memory implementations.

Shared Memory PDR

Multiprocessing (including support for multiple hardware threads) is becoming very popular. Even small COWs and laptops more and more often are capable of shared memory computing which calls for development and use of parallel shared memory algorithms. Here, we describe the Shared Memory PDR (SPDR) method. It is similar to the original, except for the following: (1) since all of the cells reside in the same memory there is no need for exchanging buffer cells, instead they are referenced by different processors (2) consequently, no operations associated with communication, such as packing and merging of submeshes, are necessary in contrast to the original algorithm. Evaluation showed (see Tab. 1) that performance of the SPDR is very close to the original implementation.

Out-of-Core PDR

The Out-of-Core SPDR (OSPDR) algorithm addresses constructing large meshes with limited resources, this also applies to using smaller amount of supercomputers' resources to avoid high wait-in-queue times.

While designing the OSPDR several assumptions were used, including those derived from the original PDR method:

1. synchronization (between neighbors) is necessary between the computational phases
2. all processors have access to any part of the mesh stored on disk and the access time is the same
3. only a small fraction of the mesh can be loaded into the system memory due to limited amount of the latter
4. disk access has the largest latency, therefore the aim of this design is to minimize the number of accesses and overlap them with computation whenever possible

*This work was supported by NSF Career Award #CCR-0049086, ITR #ACI-0085969, NGS #ANI-0203974 and ITR #CNS-0312980.

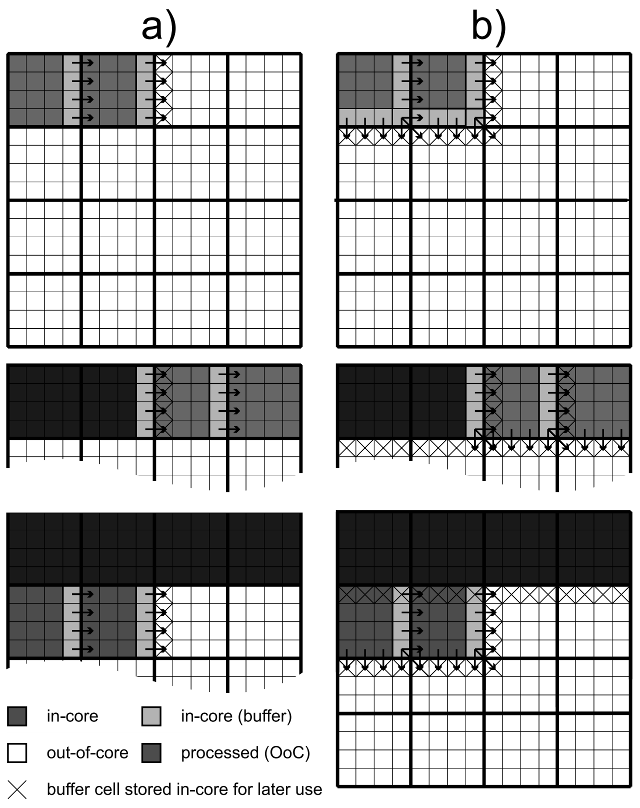

Figure 1: Out-of-core schemes of top-level shifts (2 processors, shared memory and disk)

The mesh is stored on disk in squares derived from imposing an auxiliary lattice found in the original PDR implementation. Other than the organization of shift the OSPDR is not different from the original PDR.

There are four refinement steps in the PDR method with data exchanges (shifts) in between. There are two distinct types of shifts: diagonal and horizontal/vertical. We will focus on each type regardless of direction, in particular horizontal shift to the right and diagonal shift to the right and down.

Not all of the buffer cells shift simultaneously due to majority of them residing out-of-core. We call a shift of all cells in a certain phase a top-level shift while keep the name "shift" for only those cells that currently are in-core. A top-level horizontal shift to the right is performed in the following steps (see Fig. 1.a):

- a consecutive horizontal strip of squares is loaded into the memory
- refinement is performed followed by a shift; buffer cells from the right-most square are stored in-core (unless it is also the last square in the row, then no data is stored)
- the above steps are repeated for the rest of the row, applying the stored buffer cells as exchange data for the left-most cells
- the above steps are repeated for all rows in the mesh (order is not important)

A top-level diagonal shift to the right and down involves vertical and diagonal shifts in addition to horizontal ones: all right-most buffer cells shift to the right, all bottom-most cells shift down and the right-most bottom-most cell shifts diagonally to the right and down. The top-level shift is performed in the following steps (see Fig. 1.b):

- a consecutive horizontal strip of squares is loaded into the memory
- refinement is performed followed by a shift; buffer cells from the right-most square are stored in-core (unless it is also the last square in the row, then no data is stored); buffer cells from bottom-most squares (from all cells if it is possible to load only one strip or less)
- the above steps are repeated for the rest of the row, applying the stored buffer cells as exchange data for the left-most cells
- when the next strip is loaded the buffer cells stored for the vertical part of the previous shift(s) are used with the top-most cells during shift
- the above steps are repeated for all rows in the direction of vertical component (here down) of top-level shift

Conclusion and Work in Progress

Our developments allow for greater flexibility in computing large meshes, namely it is virtually possible to generate large meshes on desktop machines and even larger meshes using small COWs. We plan to finish the implementation and evaluation of OSPDR as well as finish our work on distributed memory version of out-of-core PDR method that allows for even greater flexibility.

References

[1] J.Salmon and M. Warren. Parallel Out-of-core Methods for N-body Simulation. *8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997

[2] M. Kandemir, A. Choudhary, J. Ramanujam and M. Kandaswamy. A Unified Compiler Algorithm for Optimizing Locality, Parallelism and Communication in Out-of-core Computations. *"I/O in Parallel and Distributed Systems*, 1997

[3] M. Kandemir, A. Choudhary, J. Ramanujam and R. Bordawekar. "Compilation techniques for out-of-core parallel computations. *Parallel Computing*, 1998

[4] S. Toledo and F. Gustavson. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations. *4th Annual Workshop on I/O in Parallel and Distributed Systems*, 1996

[5] M. Hall, P. Kogge, J. Koller et al. Mapping Irregular Applications to DIVA, A PIM-based Data-intensive Architecture, 1999

[6] Tiankai Tu, D. O'Hallaron and J. López. Etree — A Database-Oriented Method for Generating Large Octree Meshes, 2003

[7] Tiankai Tu and D. O'Hallaron. Extracting Hexahedral Mesh Structures from Balanced Linear Octrees. *13th INternational Meshing Roundtable*, 2005

[8] A. N. Chernikov and N. P. Chrisochoides. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. *Proc. 18th Intl. Conf. on Supercomputing*, 48–57. ACM Press, 2004.

[9] A. Chernikov, N. Chrisocoides. Parallel Guaranteed Quality Planar Delaunay Mesh Generation by Concurrent Point Insertion. *CGW*, to appear 2005