Out-of-Core Shared Memory Parallel Delaunay Mesh Refinement

Andriy Kot*, Andrey Chernikov* and Nikos Chrisochoides*†
*Department of Computer Science, The College of William & Mary, 118 Mclolthlin-Street Hall, Williamsburg, VA, U.S.A. †Department of Mechanical Engineering, MIT and Department of Radiology, Harvard Medical School, Boston, MA, U.S.A.
e-mail: {kot, ancher, nikos}@cs.wm.edu

Abstract -
We present an out-of-core algorithm for Parallel Delaunay Refinement (PDR) for shared memory machines. Our performance data on a small 4-way shared memory machine indicate that the out-of-core PDR algorithm is 6.5 times faster than the in-core PDR implementation that relies on operating system virtual memory for meshes larger than 30 million elements. However, as it was expected, it is about twice slower than the in-core PDR implementation for meshes smaller than 20 million elements.

Keywords— Out-of-core, Parallel, Delaunay, Refinement, shared-memory.

I. INTRODUCTION

Clusters of Workstations (CoWs) and distributed memory supercomputers provide large aggregate memory and computing power. However, many applications like mesh generation do not require both types of resources in equal proportions. Moreover, it is not unusual for memory-intensive applications to use hundreds of nodes in order to utilize their memory rather than their CPUs. Considering the long wait-in-queue times associated with the usage of parallel computers, there is a need for solving the memory deficit problem.

One approach is to use operating system (OS) supported virtual memory (VM). While VM is easy to employ it has a number of limitations. First, the amount of VM is limited to 4GB for a single process on 32 bits architectures (only 2GB for Windows and Linux since a half is reserved for the OS). Second, since the OS-supported VM is optimized for system throughput, it usually cannot exploit access patterns of irregular and adaptive applications. In our tests increase in problem size from 23.8 million elements (fully in-core) to 58.8 million elements (using double the amount of physical memory) resulted in increase of the execution time from 418 seconds to more than 3 hours.

An alternative approach is to develop algorithm-specific out-of-core methods. This approach has been very effective in linear algebra parallel computations [TOL 96], [D’A 00]. Out-of-core linear algebra libraries use different mapping layouts (depending on the underlying I/O and algorithm specifics) to store out-of-core matrices and employs vendor supplied libraries for asynchronous disk I/O. They rely on high performance in-core subroutines of BLAS [DON 88], LAPACK [DEM 87] and ScaLAPACK [CHO 92] and a simple non-recursive (in most cases) pipeline to hide latencies associated with disk accesses.

Also, Salmon et al. describe an out-of-core N-body parallel method [SAL 97] which is an irregular but not adaptive i.e., there is no creation of new bodies during the execution, unlike the parallel mesh refinement computations we focus on in this paper. Their method extends the virtual memory scheme to store out-of-core pages on the disk. They use an algorithm-specific space-filling curve to arrange data within the memory pages. A problem-independent feature [SAL 97] is the page replacement algorithm which is based on the last recently used (LRU) replacement policy. The same policy is used as a basic virtual memory policy for many platforms (e.g., Linux). However, the authors extend it by introducing priorities, different aging speeds for different data types, and explicit page locking.

An out-of-core algorithm-specific approach for sequential mesh generation is Etree [TU 04]. The novelty of Etree is in the use of a spatial database to store and operate on large octree meshes. Each octant is assigned a unique key using the linear quadtree technique which is stored as a B-tree. There are three steps to generate a mesh with Etree: (1) create an unbalanced octree on disk, (2) balance the etree by decomposing further the octants that violate the 2-to-1 constraint, and (3) store the element-node relations and node coordinates in two separate databases. Subsequently, all the mesh operations are performed by querying the databases using Etree calls. This method targets octree meshes and it is exceptionally fast, especially after recent new improvements using a two-level bucket sort algorithm [TU 05]. However, it targets octree-based meshes and is not parallel.

In [KOT 05] we presented an out-of-core parallel Constrained Delaunay Mesh generation method for distributed memory machines. However, for very large mesh size problems its actual execution time is almost quadratic and it becomes prohibitively large. In order to address this problem, we developed a multi-layered approach similar to the HTMT Petaop design [GAO 96] that can mask disk latencies.

In the rest of the paper we present the Parallel Delaunay Refinement (PDR) method [CHE 04b] in Section II and
The Parallel Delaunay Refinement (PDR) algorithm (see Appendix A) is based on a theoretical framework for constructing guaranteed quality Delaunay meshes in parallel [CHE 04b]. Sequential guaranteed quality Delaunay refinement algorithms insert points at the circumsenters of triangles of poor quality or of unacceptable size. Two points are called Delaunay-independent [CHE 04a] iff they can be inserted concurrently without destroying the conformity and Delaunay properties of the mesh. In [CHE 04b] we provide 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 4r, where r is an upper bound on triangle circumradius in the initial mesh. Evaluating the above criterion for every of newly inserted points is very expensive. In [CHE 04b] we presented an efficient implementation which relies on the use of a coarse auxiliary lattice. The lattice is imposed over the triangulation domain in such a way that the circumsenters 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 [CHE 04b] is implemented by simultaneously shifting cells between processors. 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 (see Fig. 1).

The PDR algorithm is partially coupled [CHR 05] with bulk communication and very simple and inexpensive data decompositions. Thus is suitable for out-of-core parallel Delaunay mesh refinement.

III. SHARED MEMORY PDR

The Shared Memory PDR (SPDR) method is similar to the distributed (MPI) version, except for the following. First, 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. Second, the communication operations (i.e., "shifts") are eliminated and thus the packing/unpacking and merging of the submeshes. Our evaluation on a small shared memory machine shows (see Table I) that performance of the SPDR method using OpenMP is very close to the message passing (MPI) implementation of the PDR.

<table>
<thead>
<tr>
<th>Mesh size, # of elements</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3 \times 10^6</td>
<td>4.4</td>
</tr>
<tr>
<td>1.5 \times 10^6</td>
<td>20.6</td>
</tr>
<tr>
<td>3.0 \times 10^6</td>
<td>43.2</td>
</tr>
<tr>
<td>15.7 \times 10^6</td>
<td>255.5</td>
</tr>
<tr>
<td>23.8 \times 10^6</td>
<td>487.2</td>
</tr>
</tbody>
</table>

IV. OUT-OF-CORE SHARED MEMORY PDR

The Out-of-core SPDR (OSPDR) algorithm is designed to create very large meshes with relatively small memory multiprocessors. On shared memory multiprocessors we assume that: (1) all processors have access to any part of the mesh stored on disk and the access time is the same, (2) only a small fraction of the mesh can be loaded into the system memory due to limited amount of the latter, and (3) disk access has the largest latency. Therefore our goal in SPDR is to minimize the number of accesses and overlap them with computation whenever possible.

The mesh is stored on disk as a collection of subdomains. The subdomains are generated from the block decomposition (using the auxiliary lattice) we used for the PDR method. The OSPDR uses different assignment of the cells to processors, but relies on the SPDR (in-core) Delaunay meshing and refinement code.

There are four refinement steps in the PDR method with data mappings in between (see Fig. 1). We still call these data mappings shifts, for consistency with our previous work. There are two distinct types of in-core shifts: diagonal and horizontal/vertical. We will describe each shift type regardless of the direction, in particular a horizontal shift to the right and diagonal shift to the right and down.

Not all of the buffer cells shift simultaneously because the majority of the cells are residing on the disk (out-of-core). The OSPDR method uses two levels of shifts: (1) from the disk to the memory which we call top-level shift and (2) an in-core assignment of a cell from one processor to another which we simply call a shift, as before. A top-level horizontal shift to the right is performed in the following steps (see Fig. 2, left):

1) for a given row of blocks, load P consecutive horizontal blocks into the memory, where P is the number of processors.

2) each block is refined in parallel; the right-most cells of the last block are stored into buffer cells (in-core), unless
they correspond to the last column of cells in the fine-grain lattice, no data are stored in this case,

3) update the left-most cells of the newly loaded P blocks (see Fig. 2, left and middle row) using the in-core buffer cells;

4) step 1 to 3 are repeated until we complete the refinement of all blocks in that row,

5) steps 1 to 4 are repeated for all rows in the mesh (the order is not important)

A top-level diagonal shift to the right and down involves vertical and diagonal shifts in addition to horizontal ones we performed above: all right-most buffer cells shift to the right and all bottom fine-grain cells shift down and the right-most bottom cell shifts diagonally to the right and down. The top-level shift is performed in the following steps (see Fig. 2, right):

1) for a given row of blocks, load P consecutive horizontal blocks into the memory, where P is the number of processors,

2) each block is refined in parallel; the right-most cells of the last block are stored into buffer cells (in-core), unless they correspond to the last column of cells in the fine-grain lattice, then no data are stored. In addition buffer cells are used to store (in-core) the bottom fine-grain cells, unless they correspond to the last row of cells in the fine-grain lattice, no data are stored in this,

3) update the left-most cells of the newly loaded P blocks (see Fig. 2, right and middle row) using the in-core buffer cells;

4) step 1 to 3 are repeated until we complete the refinement of all blocks in that row,

5) update the top fine-grain cells of the newly loaded P
blocks (see Fig. 2, right and bottom row) using the in-core buffer cells;
6) steps 1 to 5 are repeated for all rows in the mesh using the direction of vertical component (i.e., down) of the diagonal top-level shift.

The out-of-core shared memory PDR algorithm (see Fig. 3) bellow puts all these ideas together.

V. PERFORMANCE EVALUATION

The evaluation of the SPDR and OSPDR algorithms was performed on SciClone\footnote{http://www.compsci.wm.edu/SciClone/index.html}. We used 4-way SMP nodes of subcluster “Hurricane” (4 quad-cpu Sun Enterprise 420R servers @ 450 MHz w/ 4 GB memory and 18.2 GB local disk). Also, we compare the performance of the OSPDR method with the performance of the PDR on a CoWs whose aggregate memory can be large enough to generate the same size meshes. This comparison took place on the subcluster “Whirlwind” (single-cpu Sun Fire V120 servers @ 650 MHz w/ 1 GB memory and 36.4 GB local disk. All algorithms, PDR, SPDR and OSPDR are independent of the geometry of the domain, however, for our performance evaluation we used a square geometry to eliminate other parameters like work load imbalance.

Table II shows the performance of the PDR, the SPDR and the OSPDR using Hurricane. Since we have only four 4-way machines in this subcluster we cannot generate very large meshes i.e., more than 109 million elements. However, using Whirlwind we can generate larger meshes using both PDR and OSPDR; Table III shows these performance data.

<table>
<thead>
<tr>
<th>Mesh size, # of elements</th>
<th>PDR proc</th>
<th>PDR time</th>
<th>SPDR proc</th>
<th>SPDR time</th>
<th>OSPDR proc</th>
<th>OSPDR time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$23.8 \times 10^6$</td>
<td>4</td>
<td>487</td>
<td>481</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$58.8 \times 10^6$</td>
<td>9</td>
<td>490</td>
<td>10905</td>
<td>1653</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$109.3 \times 10^6$</td>
<td>16</td>
<td>553</td>
<td>n/a</td>
<td>2413</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mesh size, # of elements</th>
<th>PDR proc</th>
<th>PDR time</th>
<th>SPDR proc</th>
<th>SPDR time</th>
<th>OSPDR proc</th>
<th>OSPDR time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$175.4 \times 10^9$</td>
<td>25</td>
<td>297</td>
<td>n/a</td>
<td>4304</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VI. SUMMARY

The explicit out-of-core shared memory Delaunay mesh refinement algorithm we present in this paper allows the generation of very large meshes on small k-way SMP desktops. Moreover it allows the generation of even larger meshes on small CoWs. This can save long wait-in-queue times on very large CoWs.

We are working on a new out-of-core implementation of the PDR method for distributed memory machines. This will permit to generate extremely large meshes in the case that some or all of the nodes can use a disk to store data. Finally, we are developing an out-of-core framework that allows a relatively easy adaptation of existing (in-core) mesh generation codes to use disk memory as a supplement for physical memory.

VII. ACKNOWLEDGMENTS

The experimental work was performed using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the NSF, and Virginia’s Commonwealth Technology Research Fund.

REFERENCES


[CHE 04b] Chernikov A. N., Chrisochoides N. P., Practical and efficient point insertion scheduling method for parallel
OUTOFCOREPARALLELDELAUNAYMESHING($\mathcal{X}$, $\Delta$, $\bar{\rho}$, $P$, $p$, $S$)

**Input:** $\mathcal{X}$ is a planar straight line graph which defines the domain $\Omega$  
$\Delta$ and $\bar{\rho}$ are desired upper bounds on triangle area  
and circumradius-to-shortest edge ratio, respectively  
$P$ is the total number of processors  
$p$ is the index of the current processor, $0 \leq p \leq P - 1$  
$S$ is the total number of subdomains (we assume $\sqrt{S}$ is integer and $P \leq S$)

**Output:** a Delaunay mesh $\mathcal{M}$ which conforms to $\mathcal{X}$ and respects $\Delta$ and $\bar{\rho}$

1. Let $l$ be the longest dimension of the bounding box of $\Omega$
2. $\bar{r} \leftarrow l/(4\sqrt{\Delta})$
3. if $p = 0$
4. Construct $\mathcal{M}_1$, a constrained Delaunay mesh of $\Omega$
5. $\Delta_1 \leftarrow \bar{r}^2/(4\bar{\rho}^3)$
6. $\mathcal{M}_1 \leftarrow $ PARAMETERIZEDDELAUNAYREFINEMENT($\mathcal{X}$, $\mathcal{M}_1$, $\Delta_1$, $\bar{\rho}$, true)
7. Assign triangles in $\mathcal{M}_1$, to $\mathcal{M}$, to buffer $B$ based on the coordinates of their circumcenters
8. for each $i \in \{0, \ldots, S - 1\}$
9. Calculate $\text{row}(i)$ and $\text{column}(i)$ of subdomain $D(i)$  
10. if $0 \leq \text{row}(i), \text{column}(i) \leq \sqrt{S} - 1$, $0 \leq i \leq S - 1$
11. Assign cells $\{d_{m,n} \mid 4\text{row}(i) - 1 \leq m \leq 4(\text{row}(i) + 1), (\text{column}(i) - 1 \leq n \leq 4(\text{column}(i) + 1))\}$ to subdomain $D(i)$ and store it on disk
12. endfor

13. for each $k \in \{p, p + P, p + 2P, \ldots, S - 1\}$
14. Calculate $\text{row}(k)$ and $\text{column}(k)$ of subdomain $D(k)$
15. Load subdomain $D(k)$
16. Assign cells $\{d_{4,j} \mid 1 \leq j \leq 3\}$ to buffer $B(\text{row}(k) + 1, \text{column}(k))$
17. Assign cells $\{d_{4,4} \mid 1 \leq i \leq 3\}$ to buffer $B(\text{row}(k), \text{column}(k) + 1)$
18. Assign cells $B(k)$ to buffer $B(\text{row}(k) + 1, \text{column}(k) + 1)$
19. if $B(k) \neq \emptyset$
20. Assign cells $B(k)$ to $\{d_{0,j} \mid 1 \leq j \leq 3\}$ from subdomain $(\text{row}(k) - 1, \text{column}(k))$
21. Assign cells $B(k)$ to $\{d_{i,0} \mid 1 \leq i \leq 3\}$ from subdomain $(\text{row}(k), \text{column}(k) - 1)$
22. Assign cells $B(k)$ to $d_{0,0}$ from subdomain $(\text{row}(k) - 1, \text{column}(k) - 1)$
23. endif

24. Let $(x_0, y_0)$ be the upper left coordinate of local cell $d_{0,0}$
25. $\sigma(q) \leftarrow (q \in [x_0 + 2\bar{r}, y_0 + 2\bar{r}] \times [x_0 + 14\bar{r}, y_0 + 14\bar{r}])$
26. $\mathcal{M} \leftarrow $ PARAMETERIZEDDELAUNAYREFINEMENT($\mathcal{X}$, $\mathcal{M}$, $\Delta$, $\bar{\rho}$, $\sigma()$)
27. Store subdomain $D(k)$ to disk
28. endfor

// Phases (1-2) are over
// Phases (3)-(10) are performed by analogy (see Figs. 1 and 2)
29. return $\mathcal{M}$

Fig. 3. An out-of-core version of parallel shared memory Delaunay refinement algorithm. The Procedure ParameterizedDelaunayRefinement is defined Appendix A.


VIII. APPENDIX A.

The parallel Delaunay Refinement algorithm [CHE04a].

**ParallelDelaunayMeshing**(*X*, *Δ*, *̄p*, *P*, *p*)

**Input:** *X* is a planar straight line graph which defines the domain *Ω*, *Δ* and *̄p* are desired upper bounds on triangle area and circumradius-to-shortest edge ratio, respectively

*P* is the total number of processors (we assume *P* is integer)

*p* is the index of the current processor, 0 ≤ *p* ≤ *P* − 1

**Output:** a Delaunay mesh *M* which conforms to *X* and respects *Δ* and *̄p*.

1. Calculate row(*p*) and column(*p*) of the current processor

   // 0 ≤ row(*i*), column(*i*) ≤ √*P* − 1, 0 ≤ *i* ≤ *P* − 1

2. Let *l* be the longest dimension of the bounding box of *Ω*

3. *r* ← *l*/(4√*P*)

4. *if* *p* = 0

5. Construct *M*1, a constrained Delaunay mesh of *Ω*

6. *Δ*1 ← *r*2/(4̄*p*2)

7. *M*1 ← **ParameterizedDelaunayRefinement**(*X*, *M*1, *Δ*1, *̄p*, true)