IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications
5-7 September 2005, Sofia, Bulgaria

Parallel Out-of-Core Constrained Delaunay Mesh Generation

Andriy Kot [1], Andrey Chernikov [2], Nikos Chrisochoides [3]
The College of William and Mary, PO Box 8795, Williamsburg, VA 23187
1) kot@cs.wm.edu, www.cs.wm.edu/~kot
2) ancher@cs.wm.edu, www.cs.wm.edu/~ancher
3) nikos@cs.wm.edu, www.cs.wm.edu/~nikos

Abstract - In this paper we present two approaches for parallel out-of-core mesh generation. The first approach is based on a traditional prioritized page replacement algorithm using prioritized version of accepted LRU replacement scheme proposed by Salmon et al. for n-body calculations. The second approach is based on the percolation model proposed for the HTMT petaflops design. We evaluate both approaches using the parallel constrained Delaunay mesh generation method. Our preliminary data suggest that for problem sizes up to half a billion element meshes the traditional approach is very effective. However for larger problem sizes (in the order of billions of elements) the traditional approach becomes prohibitively expensive, but it appears from our preliminary data that the non-traditional percolation approach is a good alternative.

Keywords - Parallel, Mesh Generation, Delaunay, Out-of-Core, Distributed

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. Considering the long wait-in-queue times associated with usage of parallel computers, there is a need for an alternative solution, namely an effective mechanism to employ a disk memory as a supplement for the RAM.

One approach is to use OS-provided 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.

Another approach is to use algorithm-specific out-of-core algorithms to bypass both the size and performance limitations of the VM. Although complexities of such algorithms are quite high, they usually achieve very good performance. For example, dense matrix operations are well-known for much better performance when performing out-of-core with specific algorithms [25] rather than VM. Although, it should be noted that these algorithms are labor intensive and once they are developed cannot be applied to a different application.

In this paper we present two out-of-core approaches for parallel mesh generation and we evaluate them with Parallel Constrained Delaunay Mesh (PCDM) generation [4]. The Out-of-core PCDM (OPCDM) is not limited in problem size and its performance is comparable to the original PCDM for small to medium mesh sizes. The OPCDM presents a good and effective alternative to PCDM, for very large meshes which require large number of processors for their aggregate physical memory. In this case the difference in waiting for larger number of processors can often be quite substantial and it can be much higher than the additional overhead cost we pay for running PCDM in an out-of-core mode.

However, our own performance data from OPCDM suggest that 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 are developing yet another approach: a multi-layered approach similar to the HTMT Petaflop design [25, 10] that masks disk latencies. Moreover we designed and implemented a multi-layered run-time system (MRTS) that handles the accesses to out-of-core memory. MRTS provides clear and simple interface to an application. MRTS targets commercial of-the-self COWS with no dedicated hardware.

The following are the research goals of the MRTS system:

- efficient handling of out-of-core requests and minimizing or completely hiding latencies associated with them

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

- modular design of the runtime system to allow for an easy plug-and-play replacement of scheduling policies and disk management schemes
- simple and clear interface for an application developer to allow easy porting from in-core to out-of-core environment[1]

In this paper we evaluate the performance of the PCDM with problem sizes that fit in physical as well as virtual memory and compare it OPCDM in terms of problem size and speed of execution. Finally, we describe the first design and implementation of the MRTS system and discuss some of its drawbacks we will be improving in the future. Our preliminary data suggest that there is much space for improvement. These improvements along with the fact that 3-dimensional mesh generation is much more demanding in memory and more computational intensive suggest that our MRTS approach can be useful for more reasonable size meshes. With the current implementation and for 2-dimensional meshes we see the benefits of the MRTS approach to appear for problem sizes larger than three billion elements (see Fig. 6). Our goal is to improve MRTS' performance so that we can see its benefits, for 3-dimensional meshes, at a size around of one billion. The meshes of such sizes are necessary for many engineering applications today.

II. RELATER WORK

To the best of our knowledge, the only out-of-core algorithm-specific approach for sequential mesh generation is Etree [27]. 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) construct, creates an unbalanced octree on disk, (2) balance, octants violating the 2-to-1 constraint are decomposed, and (3) transform, element-node relations and node coordinates are stored in two separate databases. Then, the operations on the mesh are performed by querying the database via provided Etree calls. Though limited to unstructured octree meshes, this approach provides exceptional performance where the requirements for mesh quality and conformity allow it; especially, considering the recently introduced Two-Level Bucket Sort algorithm[26] that reduces the size and time requirements for extract operations. Etree is different from our work in two ways: (1) Etree is not parallel; (2) it is specific to octree-based mesh generation.

Salmon's et al. method for N-body simulation [21] (computation with irregular access patterns like mesh generation) uses an extended virtual memory scheme to store out-of-core pages on the disk and algorithm-specific space-filling curves to arrange data within the pages. One non algorithm-specific feature [21] 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, authors extend it by introducing priorities, different aging speeds for different data types, and explicit page locking. The N-body calculations are irregular, but not adaptive, and we can not use fully this approach, although we borrow some ideas.

An approach that target different applications without limiting a developer to specific algorithm(s) involves built-in support for out-of-core directly from the compiler [15, 14]). There are two methods behind this approach: (1) conservative analysis of the code, similar to compiler cache optimization, (2) user specified directives that guide compiler on how to optimize the out-of-core operations. The first method works very well with regular applications, and is completely transparent to the programmer. The second method is suitable to handle irregular access patterns, but it requires programmer's input. Unfortunately, it is often hard or sometimes impossible for the programmer to predict the access pattern, especially for adaptive applications. Also, not enough theoretical work has been done so far for combining adaptive and out-of-core algorithms. Our work in this paper is less general than the compiler approach but addresses adaptivity in the context of a class of applications parallel mesh generation and refinement.

Finally, there are many traditional application-specific approaches for parallel dense matrix operations. For example, SOLAR [25] is a library for scalable out-of-core linear algebra computations. SOLAR uses 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. Its efficiency is achieved through the use of high performance in-core subroutines of BLAS [8], LAPACK [6] and ScaLAPACK [5] and a simple non-recursive (in most cases) pipeline to hide latencies associated with disk accesses. While limited to algorithms with regular access pattern, SOLAR performance is almost as good as in-core implementations of the same problems. Other out-of-core approaches for parallel linear algebra appear in [7, 28, 20, 19, and 17].

A. Non-traditional Out-of-core

The HTMT [24] design can be considered as an out-of-core system. One of the features of the HTMT is a Processing in Memory (PIM) [18] technology. Basically, the memory chips are combined with processing logic thus relieving the CPU(s) and caches from some of their

1 Given the original parallel application was developed using the concept of Berkeley Active Messages[29].

work. Being around for quite some time, it was generally used for regular applications (i.e., dense-arrays computations on large amounts of data). However, research [13] showed that it is possible to map irregular application to PIM-based architectures quite effectively. The main idea is to introduce some processing power close to the lower-level memory for performing memory-intensive but not computationally-intensive operations. It is very useful when dealing with huge parallel irregular computations which require gather and scatter operations of data.

The HTMT program execution model is based on the percolation model [10, 11] which relies on fine-grained threads to hide long latency events. Under this percolation model long latencies are never encountered because the right data are percolated (moved) to the right processor at the right time. Such a model relies on the ability to decompose a program into a sufficient number of fine grained automatically executed threads (tasks).

The HTMT runtime system consists of components or modules that responsible for migration of percolating threads, known as parcels, through the parallel machine.

In this paper, we borrow the percolation execution model in order to hide long latencies for traditional out-of-core parallel mesh generation applications.

III. PARALLEL CONSTRAINED DELAUNAY MESHING (PCDM)

The mesh generation procedure starts with constructing an initial mesh which conforms to the input vertices and segments, and then refines this mesh until the constraints on triangle quality and size are met. The general idea of the Delaunay refinement is to insert points in the circumcenters of triangles that violate the required bounds, until there are no such triangles left. To update the triangulation, we use the Bowyer/Watson algorithm[2, 30], which is based on deleting the triangles that are no longer Delaunay and inserting new triangles that satisfy the Delaunay property.

The set of triangles in the mesh whose circumcircles include the newly inserted point p_i is called a *cavity* [12], and we will denote it as $C(p_i)$. Also, we will use the symbol $\partial C(p_i)$ to stand for the set of edges which belong to only one triangle in $C(p_i)$, i.e., external edges.

In the absence of external boundaries, the algorithm maintains a Delaunay mesh M; at any iteration it performs the following steps:

- Select a triangle from the queue of unsatisfactory triangles.
- Compute the circumcenter p_i of this triangle.
- Find $C(p_i)$ and $\partial C(p_i)$.
- Delete all triangles in $C(p_i)$ from M.
- Add triangles obtained by connecting p_i with every edge in $\partial C(p_i)$ to M.

The case when the new point happens to be close to a constrained edge is treated separately. Following Shewchuk [22], we use diametral lenses to detect if a segment is encroached upon. The *diametral lenses* of a segment is the intersection of two disks, whose centers lie on the opposite sides of the segment on each other's boundaries, and whose boundaries intersect in the endpoints of the segment. A segment is said to be *encroached upon* by point p_i if p_i lies inside its diametral lenses. When a point selected for insertion is found out to encroach upon a segment, another point is inserted in the middle of the segment instead.

To refine the mesh in parallel, we use coarse grained domain decomposition. First, an initial conforming coarse mesh of the domain is created. The use of the available highly optimized and reliable codes (e.g. Triangle [23]) allows accomplishing this step very efficiently. Second, a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is constructed, such that (i) every triangle t_i in the coarse mesh is represented by a vertex $v_i \in V$, and the weight of v_i is set equal to the area of t_i; (ii) every edge $p_k p_1$ which is shared by two triangles t_i and t_j in the mesh is represented by an edge $e_{ij} \in E$, and the weight of e_{ij} is set equal to the length of $p_k p_1$. This graph is partitioned using the Metis library [16] into $N >> P$ vertex sets such that the total weight of the vertexes in all sets is approximately equal, and the total weight of the edges which connect vertexes in different sets is fairly small. Finally, one more graph partitioning problem is solved. Now, the goal is to distribute the subdomains among the processors, so that the sum of the weights of the subdomains on each processor is approximately equal, and the total length of the subdomain boundaries which are shared between processors is minimized. Fig. 1 shows an example of rocket engine pipe domain decomposition. During runtime, the Load Balancing Library [1] maintains the equidistribution and small edgecut conditions by moving the subdomains among the processors in response to dynamically changing work load imbalance.

The domain decomposition procedure described above creates \$N\$ subdomains, each of which is bounded by edges of the initial coarse triangulation. The edges and their endpoints that are shared between two subdomains are duplicated. The boundary edges are treated as constrained segments, and whenever they are split due to encroachment on one processor, an active message[9, 29] is sent to the processor holding the adjacent subdomain, so that the duplicate of the boundary edge is also split, and the mesh is globally consistent (see Fig. 2).

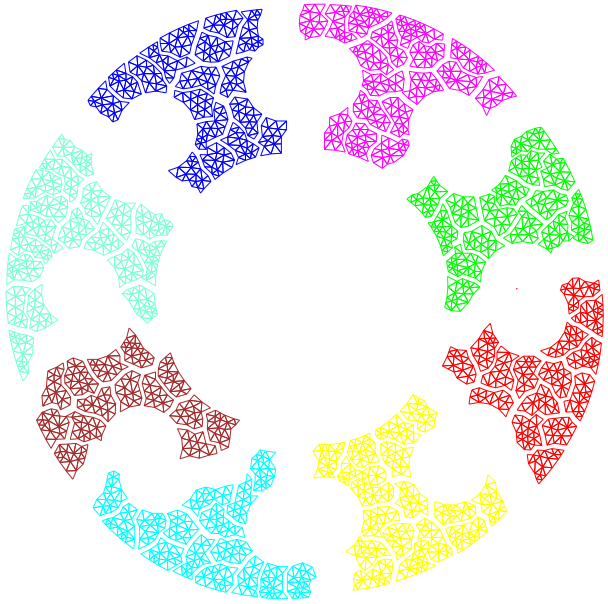

Fig. 1.A decomposition of a pipe cross-section into 128 subdomains, which are mapped into 8 processors.

A. Out-of-core PCDM

In order to generate large finite element meshes on a smaller set of computing nodes we developed a straightforward out-of-core implementation of Parallel Constrained Meshing (OPCDM). As the PCDM is continually upgraded and improved, we tried to make the porting as transparent as possible, sacrificing some performance for code reuse.

In the center of OPCDM is a table containing mapping of in-core (currently present in memory) and out-of-core (currently residing on disk) subdomains. The replacement policy is determined by the end user; for our experiments we used prioritized version of accepted last recently used replacement scheme proposed by Salmon [22]. Out of all subdomains, only a small amount is kept in-core, with the rest residing on hard drive. During refinement, an in-core subdomain is replaced with an out-of-core one if both of the following conditions are met: (1) there is pending work (i.e., outstanding splits of edges in its interfaces due to their refinement in an adjacent subdomain) for an out-of-core subdomain, (2) the amount of available physical memory is not sufficient to load new subdomains.

There are no changes in the algorithm other than the following (these changes do not alter the correctness of the original parallel algorithm):

- when a new out-of-core subdomain is picked for refinement it is put into loading queue (loads immediately if no other work is being performed)
- when a new incoming active message[29] is pending for execution and the target subdomain is out-of-core, the message's data buffer is stored until the subdomain is scheduled for loading; after the subdomain is loaded, the message's handler is executed

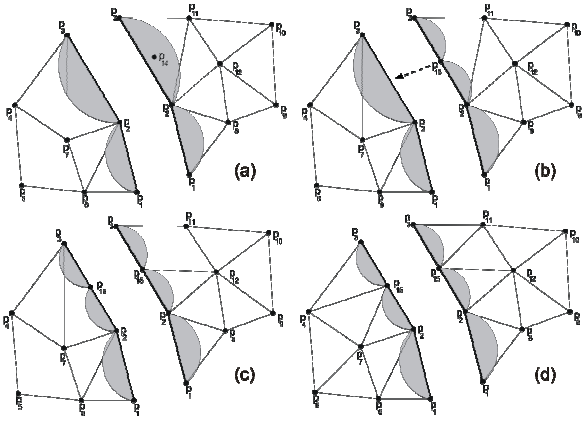

Fig. 2. If the inserted point p_{14} encroaches upon a constrained edge p_2p_3 (a), another point p_{15} is inserted in the middle of p_2p_3 instead and a message is sent to the neighboring processor (b), the local cavity of p_{15} is constructed and triangulated (c), the neighboring processor also constructs and triangulates its local cavity of p_{15} (d).

Also, some aggregation of messages is performed to minimize the amount of replacements.

B. Evaluation

The following section will provide the reader with the results of testing the performance of the PCDM using physical memory only, the PCDM with enabled virtual memory, and the OPCDM which is using disk explicitly.

We will discuss the pros and cons of all three implementations which are used as a motivation for the main contribution of this paper: the design and implementation of a run-time system which implements a percolation model for out-of-core parallel mesh generation methods.

To evaluate the performance of the PCDM implementation we used SciClone[1] computational cluster at the College of William and Mary. In particular, we used between \$32\$ and \$64\$ nodes of subcluster whirlwind (64 single-cpu Sun Fire V120 servers @ 650 MHz with 1 GB memory and 36.4 GB disk space per node) and for \$128\$-processors configuration we used additionally 64 nodes of subcluster twister (32 dual-cpu Sun Fire 280R servers @ 900 MHz with 2 GB memory and 72.8 GB disk space per node, 36.4 GB per processor). For our mesh generation tests we used pipe geometry smaller version of which is presented in Fig. 1.

Tables TABLE I, TABLE II and TABLE III show the speeds of mesh generation for problems of different sizes

[1] http://www.compsci.wm.edu/SciClone/index.html

varying from small to medium to large. Meshes were generated using 1, 32 and 128 processors configurations.

TABLE I
TRIANGULATION SPEEDS (TRIANGLES PER SECOND) FOR SMALL PROBLEM SIZE (13.8×10^6 TRIANGLES)

procs	Triangle	PCDM
1	1.4×10^4	5.9×10^4
32	n/a	8.6×10^4

TABLE II
TRIANGULATION SPEEDS (TRIANGLES PER SECOND) FOR MEDIUM PROBLEM SIZE (5.75×10^8 TRIANGLES)

procs	Triangle	PCDM	OPCDM
1	n/a	n/a	n/a
32	n/a	1.45×10^6	1.13×10^6
128	n/a	4.0×10^3	no need

TABLE III
TRIANGULATION SPEEDS (TRIANGLES PER SECOND) FOR LARGE PROBLEM SIZE (1.15×10^9 TRIANGLES)

procs	Triangle	PCDM	OPCDM
1	n/a	n/a	n/a
32	n/a	n/a	3.2×10^3
128	n/a	4.0×10^3	no need

Since Triangle [23] is not suited for parallel computation we use Parallel Delaunay Refinement (PDR) method which uses Triangle as a subroutine. The PDR method is based on the theoretical framework [3] which allows inserting points at triangle circumcenters concurrently without destroying the conformity and Delaunay properties of the mesh. It uses a coarse-grained mesh partitioning scheme which guarantees that the points in certain regions will be independent a-priori. The PDR algorithm does not rely on domain decomposition, since no explicit boundary construction is required. This method also eliminates the need to restructure the sequential mesh generation kernel and allows plugging in the available serial libraries (e.g. Triangle [23]).

Table TABLE I compares the sequential execution time and speed of the Triangle and PCDM. Due to the parallel overheads, PCDM is in most cases about two times slower. However, as we increase the mesh size up to 13.8 million, Triangle starts thrashing and becomes significantly slower, which can be explained by different choices of data structures. For a comparison, the PDR method on 36 processors creates a 12.7 million mesh in 22 seconds.

Table TABLE II compares the speed of mesh generation between the PCDM and the OPCDM for 32 and 128 processors configurations (the problem size is just too large to run sequentially, even for out-of-core). In the configuration with 32 processors the amount of memory required for computation is about the same as the total amount of available virtual memory. Therefore, the speeds of both the PCDM and the OPCDM are close. In the configuration with 128 processors, the speed of mesh generation with PCDM is even higher. However, the wait-in-queue time can be as high as 40 hours and thus rendering the effective speed to three orders of magnitude lower. We did not test the OPCDM in 128 processors configuration since with 32 processors and OPCDM we can generate half a million elements.

Table TABLE III compares speed of mesh generation of the PCDM and OPCDM for 32 and 128 processors configurations (the problem size is just too large to run sequentially with out-of-core). The problem size is so large that it is not possible to compute it even with 32 nodes with PCDM. Yet again, the speed of mesh generation is rather high on 128 processors configuration, but it is the effective speed which is dominated by wait-in-queue time of approximately 40 hours. We did not test the OPCDM in 128 processors configuration for the same reason as above. Additionally, the OPCDM on 32 processors configuration still has better effective speed. However, it is two orders of magnitude slower than with the medium problem (size is only two times smaller). This suggests that the scalability of the OPCDM is not acceptable for larger problem sizes.

Fig. 3 shows these same data in a form that allows seeing trends better.

Above data suggests that better more scalable solution is necessary to generate even larger meshes. The next section will describe a design and an execution model of which we believe is capable to reduce the disk latencies for very large size meshes (i.e., large than a billion elements).

IV. MULTI-LAYERED OUT-OF-CORE APPROACH

Similar to the HTMT architecture, we propose a run-time system organized into multiple layers. The top (execution) layer contains the fastest processors and the fastest, but very limited in size memory. The bottom (storage) layer contains the slowest processors and the slowest but large in size memory. The execution of an application is divided into multiple tasks that are executed in the top layer, but are stored in the bottom layer. The middle (control) layer is coordinating and balancing the loads of both execution and storage layers.

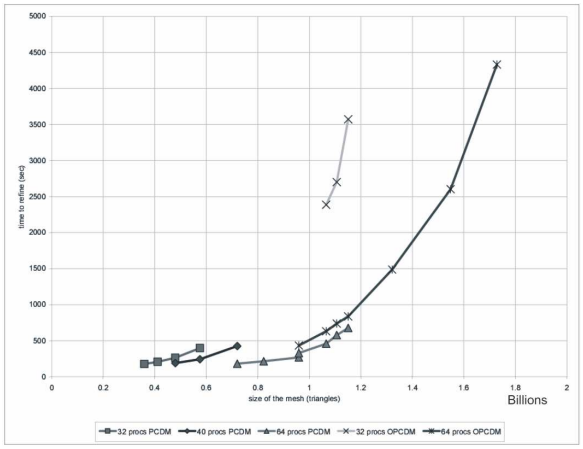

Fig. 3. Time to refine a mesh with PCDM (swap on/off) and OPCDM

The rest of the section describes the architecture of the Multi-layered Run-Time System (MRTS), based on the simplified version of the Percolation model from the HTMT Petaflop design [10, 11]. The MRTS is a software system which organizes multiple nodes of a multiprocessor or COWs in three layers:

- the Computing Engines (CE) layer - provides computing power
- the Data Servers (DS) layer - provides storage
- the Control Unit (CU) - controls system resources and execution of an application.

A. Program Execution Model

The MRTS stores application data in the DS layer and performs computations in the CE layer. The main idea behind the program execution model is a controlled movement of data used and/or produced by computations between the CE and the DS layers. A datum normally resides at the DS layer. However, when it is needed for execution it moves up (percolates) to the CE layer. Eventually, after the execution, when datum is not necessary at the CE anymore, it moves back down to the DS. The CU is responsible for achieving the best possible performance. It controls the execution of each computational block and ensures that the necessary data percolate to the CE just before the execution of the block that uses them. Fig. 4 shows the organization of the MRTS.

The MRTS uses CE nodes exclusively for computation. Therefore, they must have powerful processors. Unfortunately, very fast memory needed for nodes with high-end processors is quite expensive. As a result, the CE nodes have limited memory capacity. It is imperative that they do not stay idle, thus the rest of the system should provide continuous stream of work to the CE.

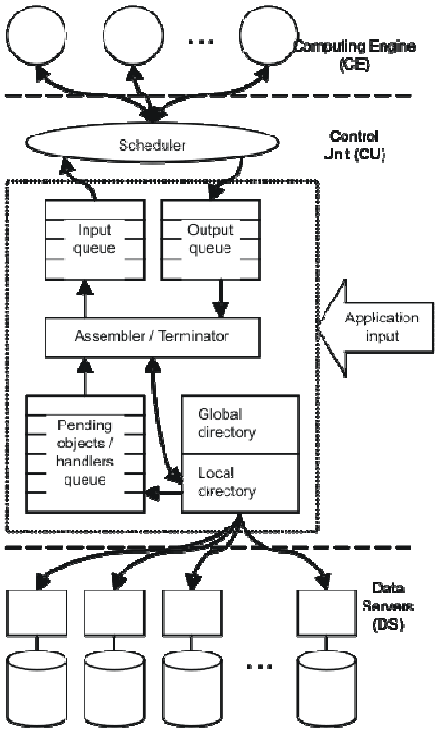

Fig. 4. Organization of the MRTS

The Data Servers provide storage for the MRTS. Therefore, these nodes must have larger memory. While RAM has certain limitations (e.g., max 4GB address space for 32 bits architectures, relatively high cost), disk memory can be an effective alternative. The DS use an out-of-core approach with physical memory acting as a cache. Currently, we employ two out-of-core strategies. The first uses virtual memory provided by the operating system. The second implements an out-of-core mechanism proposed by Salmon [22]. It is similar to OS-provided virtual memory, but allows more control over paging.

The DS layer is independent from the rest of the system, thus allowing for different implementations. In the future, we plan to develop a database storage subsystem that supports an effective mapping between the application data structures and internal system constructs (e.g., memory pages).

The CU controls the execution flow of an application. It schedules percolations of data and executes parallel blocks of computation that hide latencies associated with disk and network accesses. This suggests the CU is a potential performance bottleneck; as a result the MRTS could be hard to scale. To overcome this in large configurations, we plan to use multiple MRTS subsystems connected in one big cluster with their CUs rather than increase the number of the DS and the CE (see Fig. 5).

B. Percolation cycle

Throughout this document, we will call a computational task a handler and a datum necessary for the handler's execution an object. The following is the percolation cycle of an object:

1. after the creation and between percolations, an object resides in one of the DS nodes
2. to proceed with its execution an application posts requests for the execution of different handlers
3. when certain conditions are met, the CU picks an object and orders it to percolate to one of the CE nodes
4. the requested handlers execute soon after the object has been percolated and stored in the CE
5. after the completion of the handler, the object percolates down to one of the DS nodes.

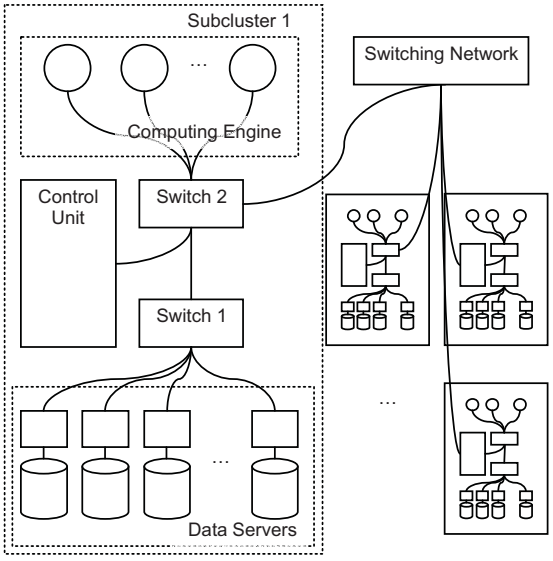

Fig. 5. Scaling for the MRTS multi-cluster

During the execution, a handler might call other handlers and create new objects. The percolation cycle repeats until there are no more handlers to execute, which signals the termination of the application.

We implemented the PCDM with a prototype of the MRTS (MPCDM). Though its performance still needs improvement, we can see a very positive trend (see Fig. 6): the MPCDM scales much better than the OPCDM.

V. SUMMARY

We presented two out-of-core approaches for parallel mesh generation and we used the PCDM application to evaluate their performance for different sizes of problems and processor configurations. Our preliminary data suggest that the traditional approach based on page replacement algorithm using prioritized version of accepted last recently used replacement scheme developed by Salmon et al. is very effective for size problems up to half a billion elements. While the second non-traditional approach based on the HTMT percolation is a good alternative for problem sizes in the order of billions of elements.

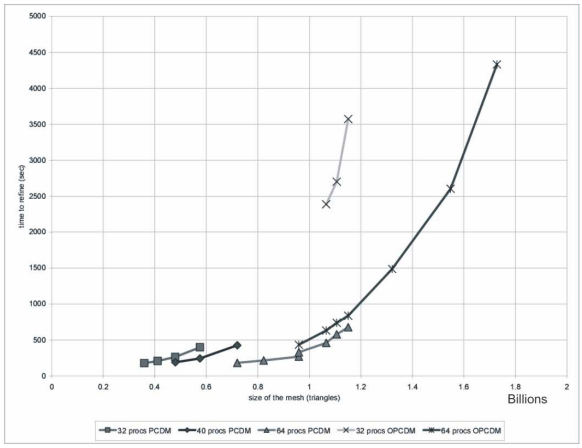

Fig. 6. Prediction of performance for the OPCDM vs the MPCDM for very large problems (billions of triangles).

We plan to improve the performance of the MRTS in order to make this approach attractive for smaller size problems. We will implement and evaluate different schemes for managing system resources on all three levels. For the DS level, we will evaluate different out-of-core strategies, including use of databases and VM on 64 bits architectures. For the CU, we will evaluate different scheduling policies for managing available computational and memory resources focusing on minimizing and completely hiding network/memory-access latencies. Finally, for the CE, we will research the possibilities of exploiting multiple level of parallelism using multi-threaded programming techniques.

Also, we plan to evaluate the use of compression on both application (mesh compression) and system level. There is an efficiency tradeoff between the time it take to compress and the time we save by transmitting/storing smaller amount of data. Therefore it is not clear yet what the pay offs are.

ACKNOWLEDGMENT

This work was performed in part using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund.

REFERENCES

[1] Kevin Barker, Andrey Chernikov, Nikos Chrisochoides, and Keshav Pingali. A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):183.192, February 2004.

[2] Adrian Bowyer. Computing Dirichlet tesselations. *Computer Journal*, 24:162.166, 1981.

[3] Andrey N. Chernikov and Nikos P. Chrisochoides. Practical and ef_cient point insertion scheduling method for parallel guaranteed quality delaunay re_nement. In *Proceedings of the 18th annual international conference on Supercomputing*, pages 48.57. ACM Press, 2004.

[4] L. Paul Chew, Nikos Chrisochoides, and Florian Sukup. Parallel constrained Delaunay meshing. In *ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation*, pages 89.96, Northwestern University, Evanston, IL, 1997.

[5] J. Choi, J. Dongarra, R. Pozo, and D. Walker. Scalapack: A scalable linear algebra for distributed memory concurrent computers. In *In Proceedings of the 4th Symposium on the Frontiers of Massively Parallel Computation*, 1992.

[6] James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, and Danny Sorensen. Prospectus for the development of a linear algebra library for high-performance computers. Technical Report ANL/MCS-TM-97, 9700 South Cass Avenue, Argonne, IL 60439-4801, USA, 1987.

[7] Jack Dongarra, Sven Hammarling, and DavidW.Walker. Key concepts for parallel out-of-core LRU factorization. Technical report, Knoxville, TN 37996, USA, 1996.

[8] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1.17, 1988. 18

[9] Andriy Fedorov and Nikos Chrisochoides. Communication support for dynamic load balancing of irregular adaptive applications. In *2004 International conference on parallel processing workshops (ICPPW'04)*, pages 555.562, 2004.

[10] G. Gao, K. Theobald, A. Marquez, and T. Sterling. The htmt program execution model, 1997.

[11] Guang Gao, Jos´e Nelson Amaral, Andr´es M´arquez, and Kevin Theobald. A re_nement of the htmt program execution model, 1998.

[12] Paul-Louis George and Houman Borouchaki. *Delaunay Triangulation and Meshing. Application to Finite Elements*. HERMES, 1998.

[13] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Apoorv Srivastava, William Athas, Jay Brockman, Vincent Freeh, Joonseok Park, and Jaewook Shin. Mapping irregular applications to DIVA, A PIM-based data-intensive architecture. 1999.

[14] M. Kandemir, A. Choudhary, J. Ramanujam, and R. Bordawekar. Compilation techniques for out-of-core parallel computations. *Parallel Computing*, 24(3.4):597.628, 1998.

[15] Mahmut T. Kandemir, Alok N. Choudhary, J. Ramanujam, and Meenakshi A. Kandaswamy. A uni_ed compiler algorithm for optimizing locality, parallelism and communication in out-of-core computations. In *I/O in Parallel and Distributed Systems*, pages 79.92, 1997.

[16] George Karypis and Vipin Kumar. *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 4.0*. University of Minnesota, September 1998.

[17] Kenneth Klimkowski and Robert A. van de Geijn. Anatomy of a parallel out-of-core dense linear solver. In *ICPP (3)*, pages 29.33, 1995.

[18] P. Kogge, J. Brockman, T. Sterling, and G. Gao. Processing in memory: Chips to peta_ops, 1997.

[19] Z. Li, J. H. Reif, and S. K. S. Gupta. Synthesizing ef_cient out-of-core programs for block recursive algorithms using block-cyclic data distributions. *IEEE Transactions on Parallel and DistributedSystems*, 10(3):297.??, 1999.

[20] Steven Newhouse. A parallel implementation of an out of core dense matrix solver using hidios.

[21] John Salmon and Michael Warren. Parallel out-of-core methods for N-body simulation. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scienti_c Computing*, 1997.

[22] Jonathan Richard Shewchuk. *Delaunay Re_nement Mesh Generation*. PhD thesis, Carnegie Mellon University, 1997.

[23] J.R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Proceedings of the First workshop on Applied Computational Geometry*, pages 123.133, Philadelphia, PA, 1996.

[24] Thomas Sterling. A hybrid technology multithreaded computer architecture for peta_ops computing, 1997.

[25] S. Toledo and F. Gustavson. The design and implementation of solar, a portable library for scalable out-of-core linear algebra computations, 1996.

[26] Tiankai Tu and David R. O'Hallaron. Extracting hexahedral mesh structures from balanced linear octrees. In *Proceedings of the 13th INternational Meshing Roundtable*, 2005.

[27] Tiankai Tu, David R. O'Hallaron, and Julio C. L´opez. Etree . a database-oriented method for generating large octree meshes.

[28] Jeffrey Scott Vitter. External memory algorithms and data structures. In James Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms and Visualization*, pages 1.38. American Mathematical Society Press, Providence, RI, 1999.

[29] T. von Eicken, D.E. Culler, K.E. Schauser, and S.C. Goldstein. Active messages: a mechanism for integrated communication and computation. In *19th annual symposium on computer architecture*, 1992.

[30] David F. Watson. Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes. *Computer Journal*, 24:167.172, 1981.