PARALLEL GUARANTEED QUALITY DELAUNAY MESH GENERATION AND REFINEMENT: CURRENT STATUS

v1.0 1/18/2003

NIKOS CHRISOCHOIDES[¶]

[¶] Computer Science Department
College of William and Mary
Williamsburg, VA 23185

ABSTRACT: Unstructured mesh generation and refinement poses a number of difficult technical challenges, both in terms of algorithms and software. During the last five years we have directed our research effort on mesh quality and parallel mesh generation and refinement. This paper describes a summary of our results, open problems and future directions for parallel guaranteed quality Delaunay mesh generation and refinement.

KEY WORDS: Parallel, Mesh, Generation, Refinement, Quality, Delaunay.

Introduction

Parallel mesh generation methods decompose the original meshing problem into smaller subproblems that can be solved (i.e., meshed) in parallel. The requirements for the solution of the subproblems on distributed memory computers are: (1) *stability*, distributed meshes should retain the good quality of elements and decomposition properties of the sequentially generated and partitioned meshes, (2) *scalability* (3) *fault-tolerance*, and (4) *code re-use*, in order to leverage from the ever evolving basic sequential meshing techniques. The design and implementation of well tested and fine tuned adaptive mesh refinement codes is a very expensive and time consuming task. The same task becomes orders of magnitude more complex in the case of *stable, scalable and guaranteed-quality mesh generation.*

In order to develop cost-effective (i.e., high code re-use), stable and scalable guaranteed-quality parallel mesh generation codes we are experimenting with three different approaches: *(1) parallelization of existing sequential guaranteed quality Delaunay methods, (2) parallel Constrained Delaunay meshing techniques, and (3) parallel Domain Decoupling approaches.* In the rest of the paper we present a short overview on the current state-of-the-art parallel methods for mesh generation and then we will briefly describe the three approaches we developed at the College of William and Mary in collaboration with our colleagues at Cornell University in the context of crack propagation project [5].

[¶]nikos@cs.wm.edu

1 Parallel Mesh Generation: An Overview

In [15] parallel mesh generation methods, for distributed memory computers are classified in terms of the *way* and the *order* the artificial boundary surfaces (interfaces) of the subproblems are meshed. Specifically, parallel methods are classified in three large classes: (i) the methods that mesh simultaneously the interfaces as they mesh the individual subproblems [14, 7, 12], (ii) methods that first mesh (in parallel or sequentially [22]) the interfaces of the subproblems and then mesh in parallel the individual subproblems, and (iii) the methods that first solve the meshing problem in each of the subproblems in parallel and then mesh the interfaces [16] so that the global mesh is consistent. Other classifications are possible, for example in [19] Lohner et al. classify parallel mesh generation methods in terms of the basic sequential meshing techniques used to mesh the individual subproblems.

In [10] we evaluate and classify parallel mesh generation methods in terms of two fundamental issues: *concurrency and synchronization* that affect the stability, performance, and code re-use of parallel meshing methods. The level (coarse-grain or fine-grain) at which concurrency is explored and the type (local or global) synchronization is required, determine the degree code re-use is possible and vise-versa. For example, high code re-use of inherently tightly coupled methods, it reduces opportunities for concurrency at a fine-grain level [22] or requires frequent global synchronization [19].

Concurrency and synchronization are independent off implementation choices like communication paradigm (shared memory or message passing) and programming model (data-parallel or task-parallel) of parallel and distributed mesh generation codes. And therefore concurrency and synchronization can be used as key attributes to evaluate parallel mesh generation methods. Parallel mesh generation methods can explore concurrency in either a coarse-grain level (i.e., subdomain or submesh level of the geometry to be meshed) or a fine-grain level (node or element level of the mesh) and they synchronize either globally (all processors are involved) or locally (only small set of processors is involved).

2 Parallel Guaranteed Quality Delaunay Mesh Generation

Most of the traditional parallel mesh generation methods explore concurrency at a coarse-grain level using stop-and-repartition methods which are based on global synchronization —a major source of overhead for large-scale parallel machines (see Figure 1). Also, traditional parallel mesh generation methods solve the mesh generation and partitioning problems separately —this leads to an unnecessary and expensive memory access overheads, since parallel mesh generation is NOT a computation intensive application. Moreover the boundary (interfaces) of the subproblems are fixed —for some algorithms this affects the stability of the parallel mesh.

In [13, 20] we presented the first provable 3-dimensional (3D) parallel guaranteed

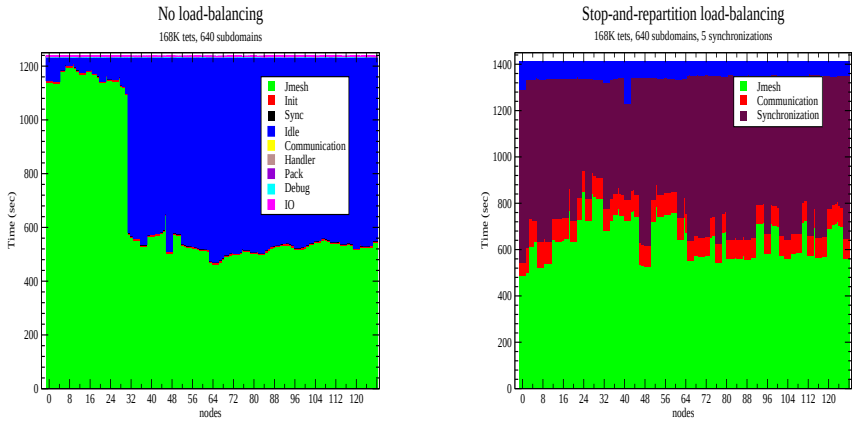

Figure 1: Performance data from parallel advance front technique [21] on 128 processor cluster. In the left graph, the blue color depicts the cycles lost due to work-load imbalance due to refinement. In the right graph, the red and magenta depict the cycles spend in communication (due to data movement) and global synchronization for traditional stop-and-repartition parallel mesh generation methods. The green color in both graphs depicts the actual computation time of the advance front mesh generation.

quality Delaunay mesh (PGQDM) generation and refinement algorithm and software for polyhedral domains. The PGQDM mathematically guarantees the generation of tetrahedra with circumradius to shortest edge ratio less than 2, as long as the angle separating any two incident segments and/or facets is between 90^o and 270^o degrees. The PGQDM kernel is based on existing sequential algorithms [23, 9].

The PGQDM algorithm: (1) explores concurrency at both coarse-grain and fine-grain levels in order to tolerate communication and local synchronization latencies, (2) couples the mesh generation and partitioning problems into a single optimization problem in order to eliminate redundant memory operations (loads/stores) from and to cache, local & remote memory, and discs, (3) allows the submesh interfaces induced by an element–wise partitioning of an initial mesh of the domain to change as the mesh is refined in order to mathematically guarantee the quality of the elements.

Figure 2a depicts an initial decomposition of a coarse mesh which is used for data distribution purposes only. Figure 2b shows that the Bowyer-Watson [4, 25] cavity extend beyond the submesh interfaces in order to guarantee the quality of the mesh. The extension of the cavity beyond the interfaces is a source of intensive communication. However as Figure 2c shows we can tolerate almost

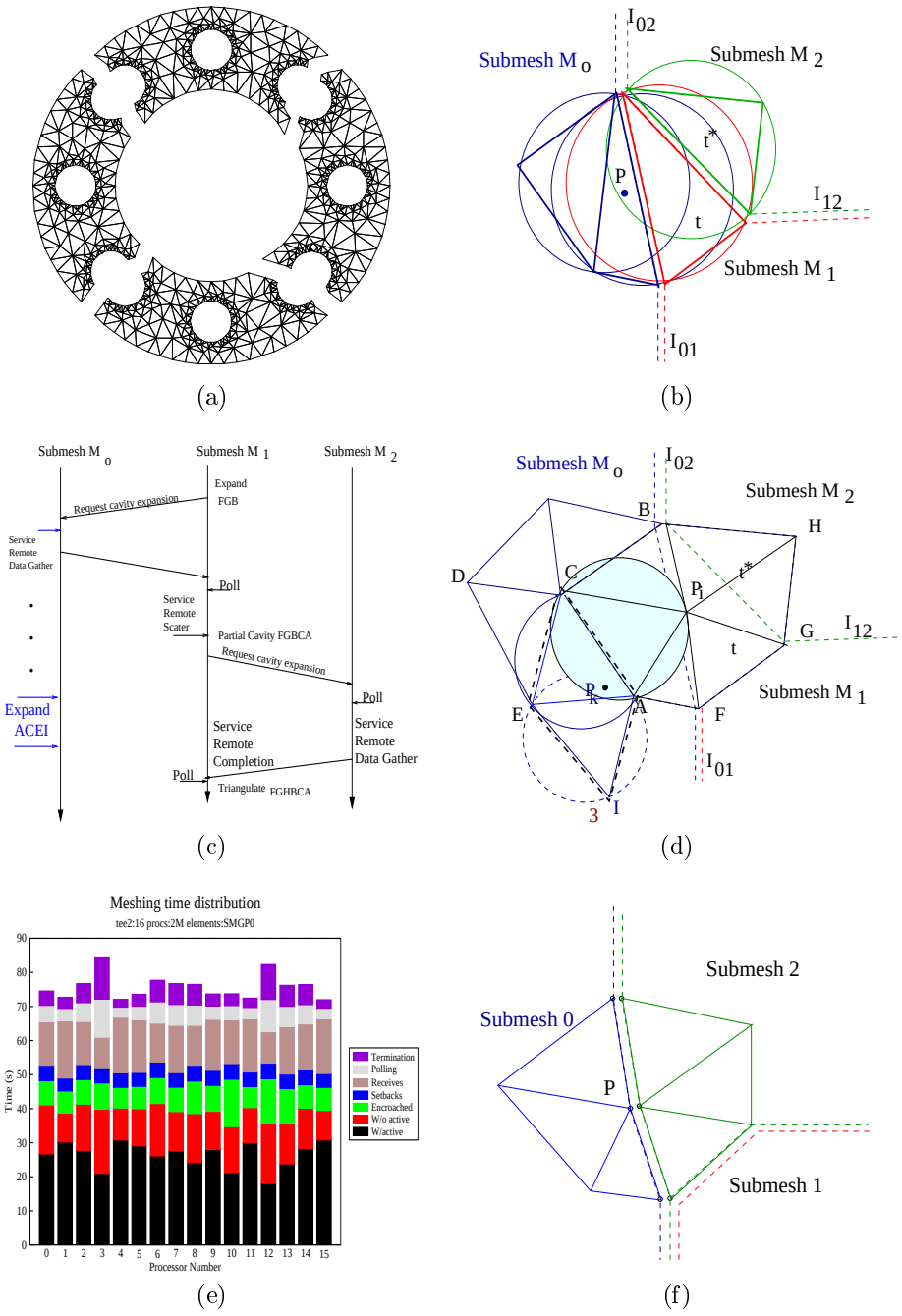

Figure 2: a) Initial data distribution, b) cavity extension beyond the submesh interfaces, c) time diagram with concurrent point insertion for tolerating communication latencies, (d) an example that depicts a source of mesh inconsistency (e) performance data that show setbacks due to elimination of mesh inconsistencies and finally (f) the refinement of a cavity with simultaneous distribution of the newly created elements.

90% of the communication by concurrently refining other regions of the submeshes while we wait for remote data to arrive. Unfortunately, the concurrent refinement can create a number of inconsistencies in the mesh as the Figure 2d shows. These inconsistencies are resolved at the cost of setbacks and frequent message polling shown in Figure 2e. At a communication cost the PGQDM it can afford to be domain decomposition independent and it allows the distribution of the new elements as they are generated (Figure 2f). Both characteristics eliminate the need to solve a challenging partitioning problem before, during or after parallel meshing.

Our performance data from a 16 node SP2 and 32 node Cluster of Sparc Workstations suggest that between 80% and 90% (in some instances even more) of the communication latency can be masked effectively at the cost of increasing communication overhead up to 20% of the total run time. Despite the increase in the communication overhead the latency-tolerant PGQDM can generate elements for parallel finite element PDE solvers many times (it depends on the size of the generated mesh) many times faster than the traditional approach which is based on the sequential generation and parallel partition and placements of elements.

In PGQDM although we mask most of the communication and synchronization latencies, the communication overhead (cycles for pushing and pulling messages from the network) is around 50% of the total runtime. In summary, PGQDM is stable, but it is communication intensive with limited code re-use. Next, I describe a stable method we developed which eliminates local synchronization and minimizes communication as it improves code re-use.

3 Parallel Constrained Delaunay Mesh Generation

Contrary to PGQDM which ignores the interface edges or faces, with the Parallel Constrained Delaunay Mesh (PCDM) generation method [7] each submesh is treated as a mesh defined by external boundary and constrained edges (or faces in 3D) which are the edges of the interfaces between any pair of adjacent submeshes. PCDM simplifies and minimizes the communication required for the parallel generation of unstructured meshes.

Intuitively, the Constrained Delaunay Mesh (CDM) is as close as one can get to the Delaunay triangulation given that one needs to handle certain (constrained) edges. This idea can be described mathematically [8] and leads to Delaunay-based mesh generators that allow internal boundaries. It has be shown [8] that these internal boundaries do not affect the quality of the resulting mesh for 2-dimensional (2D) planar domains. Although an observant user might infer the presence of such boundaries in the resulting mesh by noting the way in which triangle edges are aligned. Using the idea of a CDT, we can introduce in the mesh artificial constrained (interface) edges which decompose the mesh into submeshes that can be meshed independently.

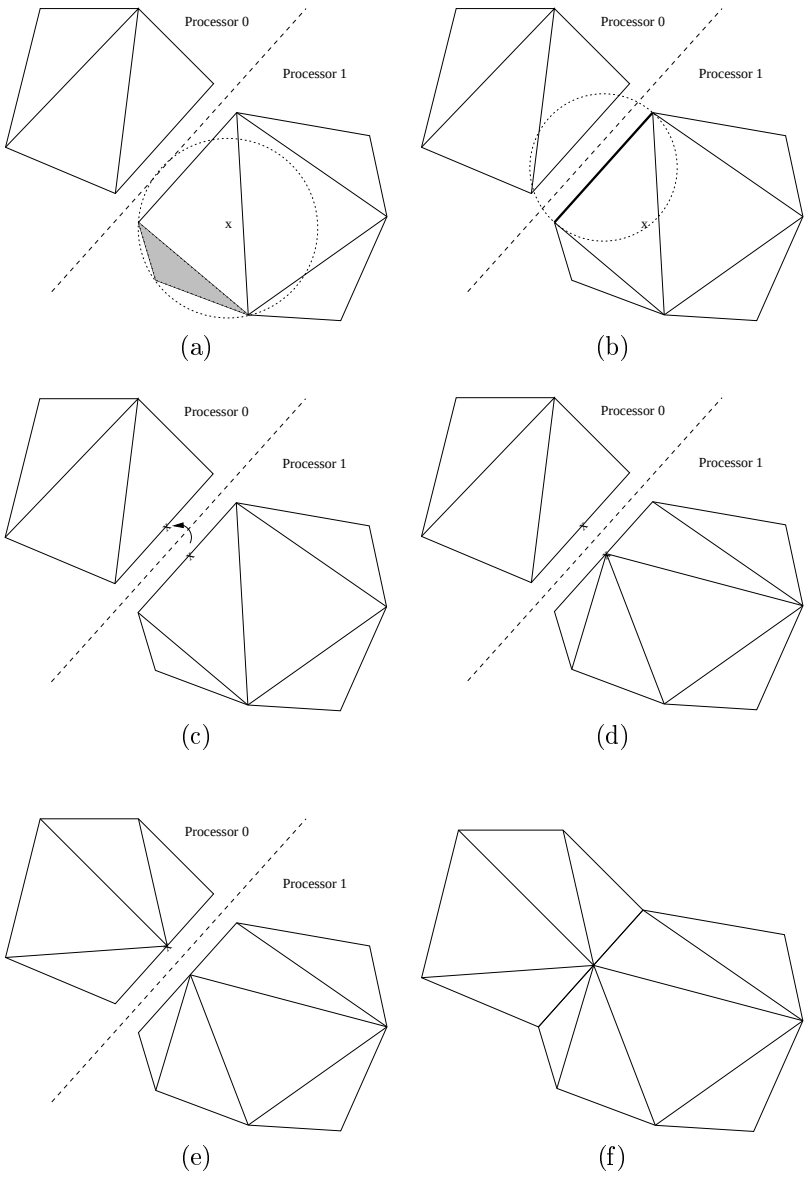

Figure 3: Processor P1 inserts a new point (a) which is encroaching upon an interface edge (b). Then P1 discards the new point and inserts the midpoint of the encroached edge(c) while at the same time it sends a request to split the same interface edge on processor P0. Processor P0 computes the cavity of the midpoint (d). The triangulation of the cavities (d) and (e) of the midpoint of the interface edge result into a new consistent and distributed Delaunay (in the CDT sense) triangulation which guarantees the quality of the elements.

The interfaces between submeshes must be created during a preprocessing (domain decomposition) step. The domain decomposition problem is not trivial since the interfaces must satisfy certain properties:

- The interfaces should not create any new features like segments and angles which are smaller in size than the features of the original geometry.
- The interfaces should minimize surface to volume ratio and connectivity between the submeshes.

The interfaces do not have to be simple line segments. The boundaries can be polylines or even curves, although line segments are probably sufficient for most situations. For planar geometries we can achieve such decomposition using Medial Axis Transform (MAT). Figure 3 depicts the MAT of the cross section of a pipe and the corresponding decomposition (see Figure 3c).
Note that, by the definition of the CDM, points on one side of the interfaces have no effect on triangles on the other side; thus, no synchronization is required during the element creation process. In addition, inter-process communication is tremendously simplified: the only message between adjacent processes is of the form, "Split this interface (i.e., constrained) edge" if a newly inserted point encroaches upon an interface edge (i.e., violates the Delaunay property of a Delaunay edge). Since interface edges always split exactly in half, no additional information needs to be communicated.
Although PCDM eliminates the synchronization and minimizes the communication of parallel guaranteed quality Delaunay mesh generation, the PCDM for 3D domains it is still an open problem. The code re-use for PCDM is not as high as we would like it to be.

4 Parallel Domain Decoupling Delaunay Mesh Generation

In order to maximize code re-use and be able to leverage from the ever evolving and mature technologies in sequential meshing we developed the Parallel Domain Decoupling Delaunay (PD^3) method [18]. PD^3 works for 2D domains and we are working for its extension on 3D domains.
The PD^3 based on the idea of decoupling the individual submeshes so that they can be meshed independently with zero communication and synchronization. In the past similar attempts to parallelize Delaunay triangulations and implement Delaunay based mesh generation presented in [3, 17]. However, in [18] we solve some of the drawbacks and improve upon the previously published methods.
The two-dimensional, divide-and-conquer Delaunay triangulation (DCDT) algorithm and its parallel implementation presented in [3] assumes that all points are known at the beginning of the triangulation. This is not a practical assumption for mesh generation and refinement, because new points are inserted on demand in order to improve element quality, perform mesh refinement, and recovery of boundary edges and faces.

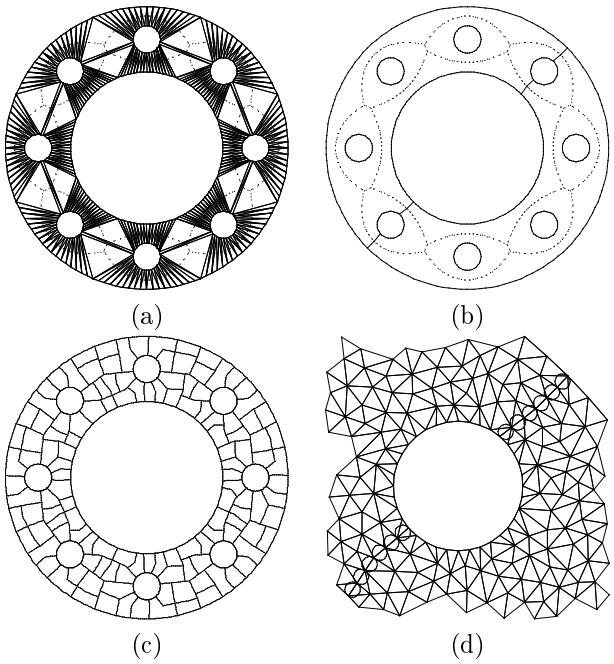

Figure 4: a) Initial boundary conforming mesh used to compute the Medial Axis Transformation (b) which in turn is used to achieve high quality domain decomposition (c). For PD^3 the interfaces of the subdomains are refined (d) in a pre-processing step.

The Parallel Projective Delaunay Meshing (P^2DM) method presented in [17] requires pre-processing of the domain in order to compute sequentially the proper Delaunay separators. This by itself is not a major problem since many methods require some pre-processing before the parallel mesh generation. However the PD^2M method after that pre-processing might suffer setbacks and start all-over again. The setbacks are in the form of discarding the mesh created because the separators do not always guarantee the Delaunay triangulation (i.e., they are not always Delaunay admissible) of the domain as new points are inserted [17]. The second problem is that the dynamic load balancing of the parallel mesh is based on prediction of the computational load of the processors. However, the computation of Delaunay meshing is variable and the workload prediction wihtout major corrections (during the mesh refinement) is very difficult and inaccurate.
In summary both methods have two main problems, proper decomposition of the domain in the context of adaptivity and dynamic load balancing. The construction of decompositions that can decouple the mesh and the proof that these decompositions lead to stable parallel meshes is a challenging problem. We use a domain decomposition method which is based on the partition of a

Table 1: Preliminary data on the number of elements and execution time (in seconds) for generating very large meshes using 1 to 128 processors.

Procs	1	8	16	32	48	64	128
Domains	12	96	192	384	576	768	1200
# elems	19M	150M	300M	600M	934M	1.2B	1.6B
Dec. Time	0.20	0.31	0.35	0.54	0.90	1.27	3.08
Mesh Time	124.18	131.57	132.57	135.04	136.78	135.81	106.31
Total time	124.38	131.88	132.92	135.58	137.68	137.08	109.39

weighted graph and an approximation of the MAT. Before the construction of the graph, the initial mesh is pre-processed in order to prevent the creation of "bad" angles [18] between either the interfaces themselves or between the interfaces and the external boundary of the domain. Then the pre-processed mesh is used to construct a dual weighted graph of the sides of the triangles. This graph is simply connected and its partition provides a decomposition of the domain.

After the decomposition of the domain, the PD^3 constructs a "zone" around the interfaces of the submeshes. In [18] we prove that Delaunay meshers will not insert any new points within this zone i.e., the sequential Delaunay mesh on the individual submeshes it can terminate without inserting any new points on the interfaces and thus eliminate communication and code modifications of the sequential codes. So the problem of parallel meshing is reduced into a "proper" domain decomposition and a discretization of interfaces. Of course one has to show that: (1) the domain is not over-refined because of a predefined discretization and (2) the quality of the elements is maintained. Our preliminary experimental data indicate that over-refined is not a major factor of inefficiency in the geometries we tested our method. Similarly the high quality of the elements is preserved.

At the end of the parallel mesh generation process some communication is required to compute the global topology of the mesh. Finally, the PD^3 achieves 100% code re-use.

5 Parallel Implementation

Providing adequate software support for adaptive mesh refinement codes is a challenging task even for sequential implementations. The program complexity, for parallel mesh generation codes that are scalable and stable, increases by an order of magnitude, due their dynamic and irregular computation, communication and synchronization requirements. In order to handle the software complexity of managing task parallelism we developed a Portable Runtime Environment for Multiprocessor Applications (PREMA).

5.1 Parallel Runtime Environment for Multiprocessor Applications

PREMA's design objective is to assist mesh generation practitioners to develop parallel meshing codes in an evolutionary (incremental) way starting from well tested and fine-tuned sequential codes (preferable written in C or C++ programming languages). PREMA supports a Simple Mesh Partitioning Library (SMPL) that runs on a single processor and gets as input a mesh which is defined in the standard format that most of the public domain Finite Element Analysis codes use. SMPL generates as output the submeshes using the same format, but at the end appends adjacency information at the subdomain level.
The next two layers handle communication [2, 11]. The *Data Movement and Control Substrate* (DMCS) implements one-sided low-latency communication. DMCS is built on top of low-level, vendor-specific communication subsystems that bypass the operating system and access the network interface directly in order to minimize communication startup overheads. Also, it is implemented on widely available libraries like MPI for clusters of workstations and PCs. DMCS adds a small overhead (less than 10%) to the communication operations provided by the underline communication system. In return DMCS provides a flexible and easy to understand application program interface for one-sided communication operations including *put-ops* and *remote procedure calls*. Furthermore, DMCS is designed so that it can be easily ported and maintained by non-experts. The second communication component is the Mobile Object Layer (MOL). MOL supports one-sided communication in the context of data (submesh) or object mobility. Specifically, MOL uses global logical name space and supports automatic message forwarding in order to ease the implementation of dynamic load balancing methods for adaptive applications on distributed memory machines. MOL is implemented on top of DMCS and adds another 10% to 15% overhead. In return the application programmer can perform data (submesh) or object migration, for load balancing, without changing a single line of code.
Finally, the third layer implements the dynamic Load Balancing Library (LBL) on top of DMCS and MOL. LBL supports both explicit and implicit load balancing methods. LBL uses the abstraction of *Schedulable Object* (which sometimes is called Work Unit) to represent application-defined data objects like the submeshes, for the case of discrete DD methods. The Schedulable Object (SO) is the smallest unit of granularity that is managed by the runtime system. Balancing processors workload or memory (in the case of adaptive refinement) results in moving one or more Schedulable Objects. The SOs are migrated automatically by the *Load Migrator* which is responsible for un-installing, moving, and installing SOs using MOL so that messages to them will automatically forwarded. The decision making for which SO and where to migrate is handled by the *Scheduler*. The Scheduler supports a wide spectrum of load-balancing methods and it is described in detail in [1]. In the next version of the LBL tools like *Load Monitor* and *System Monitor* will provide to the scheduler all the systems (processors and network) information it needs for effective dynamic

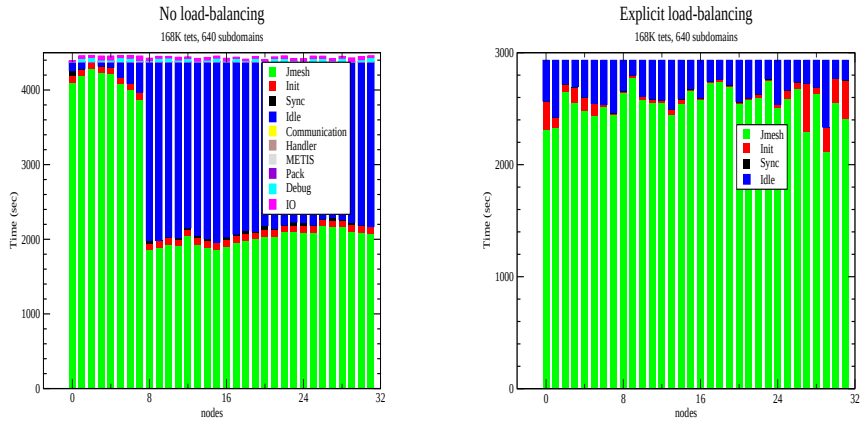

Figure 5: Performance data from parallel advance front technique [21] on 32 processor Sun cluster. The explicit method is based on work-stealing [1] and does not require global synchronization.

load balancing.

By using PREMA parallel mesh practitioners can be assisted in many different levels. First, they can customize to their code requirements one-sided message passing operations like *put_ op*, where *op* is user defined function. We found that this functionality simplifies code complexity substantially, since the application programmer does not have to concern about what to do when a message arrives. Second, they know that at some point they have to implement mesh refinement i.e., they know that they will have to move submeshes (object) among the processors for load balancing purposes, they can use MOL messages which find the object independently off its physical location. Third, the programmer can get load balancing (and in the future out-of-core implementation), for free, by using the LBL's *Scheduler*.

Finally once the parallel meshing code is build and it is ready for production, the mesh practitioners can fine tune the performance by directly using *Load Migrator* and for performing explicit load balancing and swapping of SOs (i.e., submeshes). This incremental or evolutionary approach is described in great detail in [1], for parallel Constrained Delaunay meshing. Figure 5 depicts some initial performance data from balancing a parallel advance front method on 32 processors. More and better methods need to be developed for larger processors configurations. Fortunately the plug-and-play software design of PREMA makes this task very easy. From the last 15 years of experience we have seen that there is no single dynamic load balancing method for all applications or even parallel mesh generation techniques.

Conclusions

We have described three parallel guaranteed quality Delaunay mesh generation methods; the PGQDM for 3D polyhedral domains and two methods (PCDM and PD^3) for 2D planar geometries. In addition we briefly described a parallel runtime system, PREMA, we used to implement these methods. PREMA can be used to solve the dynamic load balancing problem. Thus PREMA allows us to focus on yet another very important and still open problem for parallel meshing, the domain decomposition problem.
The PGQDM is the first parallel provable 3D meshing method and it is independent of the quality of the domain decomposition. Its disadvantage are: (1) it involves complex data structures for maintaining the consistency and Delaunay properties of the distributed mesh, (2) it requires high communication which affects reduces its fixed speedup to O(logP), P is the number of processors and (3) it has zero code re-use. However, despite these difficulties it is the only method that does not dependent on domain decomposition and thus we are working [24] to alleviate some of the problems we mentioned above and extended it for 3D geometries with curved boundary.
Contrary to PGQDM the PCDM is easier to implement with an order of magnitude less communication and with linear speedup, but it depends on high quality domain decompositions. Current techniques for domain decomposition are not capable to always provide domain decompositions suitable for PCDM. We have solved the domain decomposition problem for 2D planar geometries and we are working [6] to solve this problem for 3D geometries.
Finally, the PD^3 method we believe is the "holy grail" for parallel meshing. It requires zero communication and synchronization with 100% code re-use. Unfortunately, like PCDM requires special domain decompositions which are very challenging to generate for general geometries. For 2D planar geometries we have been very successful to generate such decompositions. We have a very good progress in applying the same techniques for 3D polyhedral geometries, but for 3D domains with curved boundary the problem is still open.

6 Acknowledgments

Many results that appear in this paper are due to my work with my students Andrey Chernikov, Andriy Fedorov, Demian Nave, Leonidas Linardakis and Chaman Verma, as well as my colleagues from Cornell University especially Dr. Paul Chew. This research has been funded by NSF CISE Challenge #EIA-9726388, Career Award CCR-9876179, Research Infrastructure grant #EIA-9972853, ITR grant #ACI-0085969 and NGS grant EIA-0203974.

References

[1] K. Barker, N. Chrisochoides, A. Chernikov, and K. Pingali. Architecture and evaluation of a load balancing framework for adaptive and asyn-

chronous applications. *IEEE Trans. Parallel and Distributed Systems,*, Under revision, 2003.

[2] Kevin Barker, Nikos Chrisochoides, Jeff Dobbelaer, Démian Nave, and Keshav Pingali. Data movement and control substrate for parallel, adaptive applications. *Concurrency and Computation Practice and Experience*, 14, 2002.

[3] G. Blelloch, J. Hardwick, G. Miller, and D. Talmor. Design and implementation of a practical parallel delaunay algorithm. *Algorithmica*, 24:243–269, 1999.

[4] Adrian Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[5] Bruce Carter, Chuin-Shan Chen, L. Paul Chew, Nikos Chrisochoides, Guang R. Gao, Gerd Heber, Antony R. Ingraffea, Chris Myers Roland Krause and, Démian Nave, Keshav Pingali, Paul Stodghill, Stephen Vavasis, and Paul A. Wawrzynek. Parallel fem simulation of crack propagation – challenges, status. In *Lecture Notes in Computer Science*, volume 1800, pages 443–449. Springer-Verlag, 2000.

[6] Andrey Chernikov. *Domain Decomposition for Guaranteed Quality Parallel Mesh Generation*. PhD thesis, College of William and Mary, 2005.

[7] P. Chew, N. Chrisochoides, and F. Sukup. Parallel constrained delaunay meshing. In *Special Symposium on Trends in Unstructured Mesh Generation*. ASME/ASCE/SES, 1997.

[8] Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[9] Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. 9th Annu. ACM Sympos. Cornput. Geom.*, pages 274–280, 1993.

[10] Nikos Chrisochoides. A survey of parallel unstructured mesh generation methods. *SIAM Review*, To be submitted, 2003.

[11] Nikos Chrisochoides, Kevin Barker, Demian Nave, and Chris Hawblitzel. Mobile object layer: A runtime substrate for parallel adaptive and irregular computations. *Advances in Engineering Software*, 31(8-9):621–637, 2000.

[12] Nikos Chrisochoides and Demian Nave. Simultaneous mesh generation and partitioning. *Mathematics and Computers in Simulation*, 54(4-5):321–339, 2000.

[13] Nikos Chrisochoides and Démian Nave. Parallel delaunay mesh generation kernel. *IJNME*, To appear in 2003.

[14] Nikos Chrisochoides and Florian Sukup. Task parallel implementation of the BOWYER-WATSON algorithm. In *Proceedings of Fifth International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, 1996.

[15] H. de Cougny and M. Shephard. *CRC Handbook of Grid Generation*, chapter Parallel unstructured grid generation, pages 24.1–24.18. CRC Press, Inc., 1999.

[16] H. de Cougny and M. Shephard. Parallel volume meshing using face removals and hierarchical repartitioning. *Comp. Meth. Appl. Mech. Engng.*, 174(3-4):275–298, 1999.

[17] J. Galtier and P. L. George. Prepartitioning as a way to mesh subdomains in parallel. In *Special Symposium on Trends in Unstructured Mesh Generation*. ASME/ASCE/SES, 1997.

[18] L. Linardakis and N. Chrisochoides. Parallel delaunay domain decoupling method for planar geometries. *Algorithmica*, To be submitted in 2003.

[19] R. Lohner and J. Cebral. Parallel advancing front grid generation. In *International Meshing Roundtable*. Sandia National Labs, 1999.

[20] D. Nave, N. Chrisochoides, and P. Chew. Guaranteed-quality parallel delaunay refinement for restricted polyhedral domains. In *Proceedings of 8th ACM Symposium on Computational Geometry*, pages 135–144, 2002.

[21] J. B. Cavalcante Neto, P. A. Wawrzynek, M. T. M. Carvalho, L. F. Martha, and A. R. Ingraffea. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *EwC*, 17(1):75–91, 2001.

[22] R. Said, N. Weatherill, K. Morgan, and N. Verhoeven. Distributed parallel delaunay mesh generation. *Comp. Methods Appl. Mech. Engrg.*, 177:109–125, 1999.

[23] Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Carnegie Mellon University, School of Computer Science, May 1997. Available as Technical Report CMU-CS-97-137.

[24] Chaman Verma. Parallel guaranteed quality delaunay mesh generation for 3d geometries with curved boundaries. Master's thesis, College of William and Mary, 2003.

[25] David F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.