A Case Study of Optimistic Computing on the Grid: Parallel Mesh Generation

Nikos Chrisochoides[*], Andriy Fedorov, Bruce B. Lowekamp and Marcia Zangrilli
Computer Science Department
College of William and Mary
Williamsburg VA 23185
{nikos, fedorov, lowekamp, mazang}@cs.wm.edu

Craig Lee
The Aerospace Corporation lee@aero.org

January 28, 2003

Abstract

This paper describes our progress in creating a case study on optimistic computing for the Grid using parallel mesh generation. For the implementation of both methods we will be using a Portable Runtime Environment for Mobile Applications (PREMA) which is extended to provide support for optimistic control using grid performance monitoring and prediction.

Based on the observed performance of a world-wide grid testbed, we will use this case study to develop a methodology for estimating target operating regions for grid applications. The goal of this project is to generalize the experience and knowledge of optimistic grid computing gained through mesh generation into a tool that can be applied to tightly coupled computations in other application domains.

1 Introduction

Computational grids offer a vast pool of computational and storage resources which can be utilized by large-scale engineering and scientific computing applications. Mesh generation is an integral part of many important engineering and scientific computing applications. The projected memory and the time it will take to generate and partition a 3-dimensional billion elements mesh on a single workstation or even on a single supercomputer is expected to be prohibitively large compared to the analysis phase of scientific computing simulations. Although mesh generation is not a natural application for the Grid, its strategic importance for multi-scale simulations and its memory requirements are forcing us to consider the task of generating 3-dimensional meshes on the Grid. In this project we evaluate alternative options for performing this task. In addition we use mesh generation as a case study to examine the feasibility of porting tightly-coupled applications like mesh generation on the Grid. Our approach is based on algorithm re-structuring for tolerating long, variable, and unpredictable communication and synchronization latencies.

In this project we will evaluate two different mesh generation approaches and we will perform the following tasks:

- Develop a runtime infrastructure that enables an application to interact with its performance environment while facilitating quick prototyping, experimentation, and evaluation.
- Explore how optimistic execution and other tech-

[*] Corresponding author

niques must be controlled to scale-up and enable an application to work in its grid "operating region."

- Develop and evaluate new network measurement techniques and performance models that cover a wide spectrum of communication characteristics needed by mesh generation methods to implement properly grid-aware applications.

In the rest of the paper we introduce the concept of optimistic computation and the behavior that an application must exhibit to benefit from it. This application behavior is tied to the concept of an "operating region" for grid applications based on "keeping the pipes full" and matching the computation/communication ratio to the environment. We then introduce parallel mesh generation methods which can be reconceived as optimistic computations. These methods can be supported by a specialized run-time system (see Section 4) that incorporates grid performance monitoring. We conclude in Section 5.

2 Optimistic Computation for Grid Environments

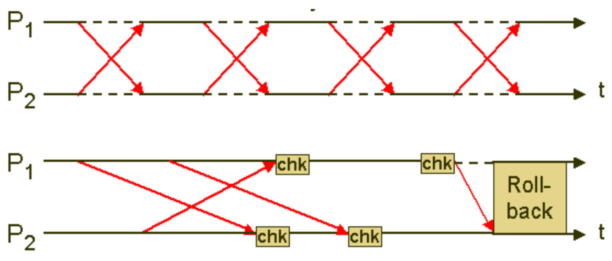

Figure 1: Cost trade-offs between strict control and optimistic control.

Optimistic parallel computation is very similar to optimistic parallel simulation [8]. In that work, multiple end-hosts simulate in parallel and exchange time-stamped events. If a host receives an event with a time-stamp earlier than its current simulated time, i.e., "in its past", then it must *roll-back* its simulation to the earlier time and start again. This requires that an end-host do *incremental state saving* to enable roll-backs, and possibly send *anti-messages* that undo the effect of messages sent to other end-hosts during the rolled-back period. The critical trade-off that determines the advantage of optimistic simulation is the overhead of roll-backs (frequency and severity) versus the benefit of more loosely coupled, parallel simulations that have reduced synchronization and communication delays. This is a fundamental trade-off that applies to all optimistic computation. Optimistic execution can be used to tolerate synchronization and communication latencies but allows *inconsistent* results to be computed.

This trade-off is illustrated in Figure 1. A strictly synchronized application has some overhead due to this synchronization and communication control. An optimistic computation will have some overhead for consistency checking using *validation messages*. Since consistency validation can proceed independently, these messages can be sent asynchronously and also be pipelined. When an inconsistency is detected, however, roll-back must occur which could be a more heavyweight and synchronous operation. For optimistic computation to be advantageous, the following inequality must hold:

$$\text{Cost}_{\text{validation}} + \text{Cost}_{\text{roll-back}} \ll \text{Cost}_{\text{strict-control}} \quad (1)$$

The costs of validation and roll-back should be much less than the cost of strict control. If the costs are similar, or greater, then there will be no advantage, or even a disadvantage. Cost will typically be defined in terms of execution time but this cost could be broken down into communication time, synchronization delays, and also discarded work.

For grid environments, a key issue is "how much optimism is enough"? Optimistic application must have some type of independent parameter, or "control-knob," that determines the amount of optimism. Clearly over-optimistic execution will suffer excessive roll-backs while under-optimistic execution will not improve parallel performance by not being able to tolerate much of communication and synchronization latencies over the Grid. Hence, an application should only generate *just enough* optimistic parallelism to compensate for the current network bandwidth and latency without wasting too much work.

This notion can be used to define an *operating region* for grid applications [12]. Using a simple pipeline model of *work units* arriving at a host, we can make assumptions of the work unit size and compute time it requires at the host. Using a snapshot (see Section 4.2) of a global grid's network bandwidth and latency, we can bound performance regions that represent the number of threads an application must generate, the size of the work units, and their computational time in order to (1) "keep the pipes full," and (2) match the computation/communication ratio. While this pipeline model elides many application issues (such as explicit synchronization and data dependencies, and a distribution of work unit sizes), it nonetheless gives us a potential mechanism whereby we can *monitor the current grid communication performance and control an application's behavior to stay within a targeted operating region.*

Thus, a major goal of this project is to integrate such a control model into PREMA. This optimistic control module will utilize grid performance monitoring tools to evaluate the current environment and then determine how the application behave, e.g., how aggressively it should do optimistic cavity expansion. The next section describes briefly two of the three parallel mesh generation techniques we consider in this project —due to lack of space we will describe in our presentation the "control knob" parameters that can be used to control the behavior of some of the meshers.

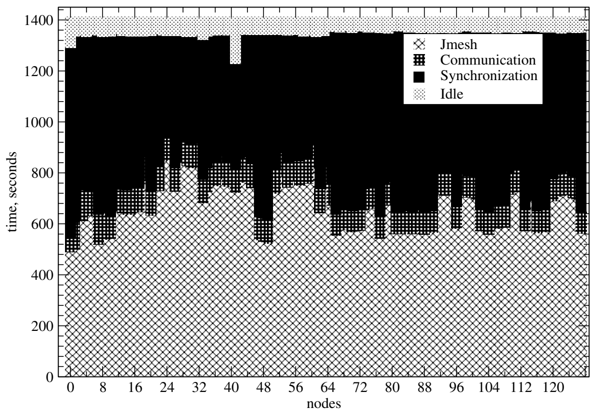

Figure 2: Performance data from generating 168K tet mesh using parallel advance front technique for crack propagation [18] on 128 processor Sun cluster. The graph depicts the cycles spend in communication (due to data movement) and global synchronization due stop-and-repartition load balancing. It also shows the actual computation time spend on the advance front mesh generation.

3 Parallel 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 2) and the Grid. Also, traditional parallel mesh generation methods solve the mesh generation and partitioning problems separately—leading to an unnecessary and expensive memory access overheads for large scale parallel machines and the Grid, 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 (i.e., distributed meshes should retain the high quality of elements and partition properties of the sequentially generated and partitioned meshes).

In [17] we presented the first provable 3-dimensional (3D) parallel guaranteed quality Delaunay mesh (PGQDM) generation and refinement algorithm/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° and 270°. The PGQDM kernel is based on the parallelization and re-structuring of existing sequential algorithms [19] for the meshing of the external boundary. In PGQDM we use concurrency as a mechanism to control optimism required to tolerate long, variable and unpredictable communication and synchronization latencies.

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 disks, (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. These change in the interfaces are the source of setbacks in PGQDM.

In this project we will study the trade-offs between speculative execution and setbacks in PGQDM in the context of the Grid. Although it is desirable to tolerate communication and synchronization and we can currently mask 80% to 90% (in some instances even more) of the communication latency [5] on a CoWs, , it is ideal to reduce communication and synchronization into a single communication phase at the end with minimum possible data exchange, especially for grid applications. Moreover it will be very practical if we could achieve this objective without re-structuring the sequential algorithms/software. Next we present our preliminary results in this direction from a 2-dimensional (2D) parallel mesh guaranteed quality Delaunay mesh generation.

Parallel Domain Decoupling Delaunay Mesh Generation In order to eliminate communication and synchronization and 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 [14]. PD^3 works for 2D domains and we are working to extend it for 3D domains while the PGQDM is working for both 2D and 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, 6]. However, in [14] we solve some of the drawbacks and improve upon the previously published methods.

The two-dimensional triangulation algorithm 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. In contrast, the Parallel Projective Delaunay Meshing (P^2DM) method [6] is suitable for meshing, but it might suffer much more severe setbacks compared to what we observe in PGQDM. The setbacks are in the form of discarding the complete mesh. This might happen (at any time) when the sequentially generated separators fail to be Delaunay admissible for some of the newly inserted points.

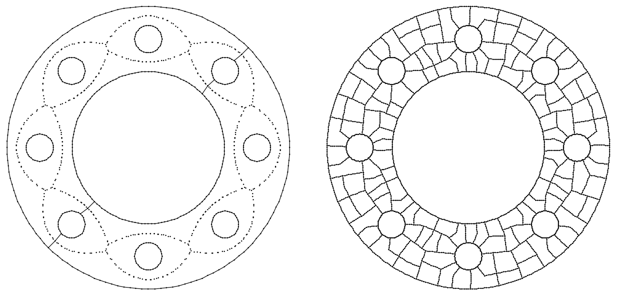
Figure 3: a) Initial boundary conforming mesh used to compute the Medial Axis Transformation (left) which in turn is used to achieve high quality domain decomposition (right).

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	32	64	128
Domains	12	384	768	1200
# elems	19M	600M	1.2B	1.6B
Dec. Time	0.20	0.54	1.27	3.08
Mesh Time	124.18	135.04	135.81	106.31
Total time	124.38	135.58	137.08	109.39

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 weighted graph and an approximation of the Medial Axis Transformation (see Figure 3). Before the construction of the graph, the initial mesh is pre-processed in order to prevent the creation of "bad" angles [14] 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 [14] 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 (see Table 1) that over-refinement is not a major factor of inefficiency for decomposition in the order of thousands of subdomains and for the geometries we tested our method. Similarly the high quality of the elements is preserved.

4 Portable Runtime Environment for Mobile Applications

PREMA's design objective is to assist application developers to implement parallel adaptive codes in an evolutionary way starting from well tested and fine-tuned sequential codes (preferable written in C or C++ programming languages) and transform them into parallel and grid-aware ones.

PREMA's communication layer is the *Data Movement and Control Substrate* (DMCS) which implements one-sided low-latency communication. DMCS is implemented on top of MPI for CoWs and the Grid (using Globus-based MPICH-G2 [7] implementation). No significant overhead has been observed in the preliminary comparison of MPICH-G2/DMCS performance over regular MPICH/DMCS implementation (see Figure 4). DMCS adds a small overhead (less than 3% in the case of MPI libraries and less than 10% for low latency systems like LAPI) 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 *get/put_ops* and *remote procedure calls* [2].

The second communication component is the Mobile Object Layer (MOL). MOL supports global name space

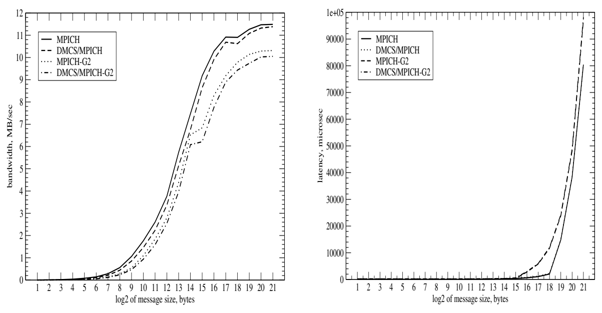

Figure 4: Bandwidth and latency for asynchronous, blocking one-sided communication primitive of DMCS on top of MPICH for CoW and the Grid.

in the context of data (submesh) or object mobility [4]. Specifically, MOL uses global pointers to implement a global logical name space and supports automatic message forwarding in order to ease the implementation of dynamic load balancing for adaptive applications on CoWs and the Grid. 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.

Location management policy (LMP) is a crucial part of the MOL implementation. LMP consists of rules for performing searches, updates, and search-updates of the objects location information. There are different strategies for each of these three operations [10]. Our preliminary results show that the choice of the LMP can significantly affect the performance of the application in some cases. Figure 5 depicts performance data from 1-way and 2-way SMP nodes using different LMPs for parallel bitonic sort where sorted elements move on each iteration. As a part of our research, we are going to evaluate the properties of different LMPs in a Grid environment. We will also explore how the knowledge about the topology and dynamic properties of the network (bandwidth and latency) can be used to improve the overall efficiency of the LMP.

The third layer implements the dynamic Load Balancing Library (LBL) on top of DMCS and MOL. LBL uses the abstraction of a *Schedulable Object* (SO) to represent application-defined data objects like the submeshes, for parallel mesh generation. The SO is the smallest unit of

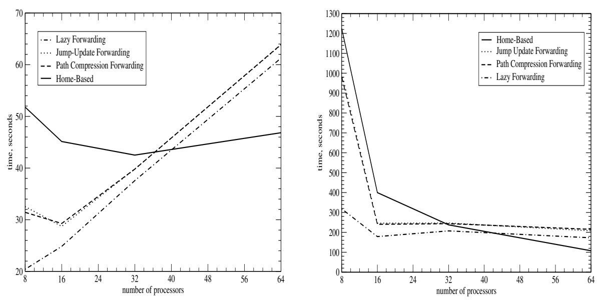

Figure 5: Performance of parallel bitonic sort on 1-way (left) and 2-way (right) SMP nodes using four different LMPs and MPILAM implementation of DMCS.

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 SOs. 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, modules like the *Wren* network monitor system will provide to the scheduler network information it needs for effective dynamic load balancing.

4.1 An Optimistic Control Module

Making effective use of optimistic algorithms for mesh generation is only possible if the algorithms can be adjusted to the right level of optimism. If the control knob is set too high, incorrect optimistic assumptions may be made when the communication would be less costly. If the knob is set too low, the application will wait for communication to complete when it could be performing optimistic computations that may be useful for the overall computation.

PREMA is being extended with an *Optimistic Control Module (OCM)* that is illustrated in Figure 6. The OCM interfaces with the application and with other middleware modules to acquire the information that is necessary to control the optimistic behavior. On a given host, a mesh generation code will build a local submesh that has one or more *boundaries* with submeshes on other hosts. PREMA will maintain a "pipe" connection between each submesh pair. For each connection, the network performance will be monitored such that the current bandwidth and round-trip time (RTT) will be available to the OCM. From this, simple, one-way latency and the bandwidth-delay product can be derived. Simultaneously the DMCS will monitor the number and size of messages sent and received by the application. The application itself will monitor the processing time per message and the number of setbacks and the associated processing time that was wasted.

With this information, the OCM can apply a control model designed to keep the application within the operating region as closely as possible. In the case of mesh generation, this control model will determine the number of interface threads the application should use, the number of parallel cavity expansions that can be underway, and the frequency that cavities can be created. Clearly many classical control issues become relevant here. This is a control loop where hysteresis may have to be applied to prevent oscillations or over-compensations. Another possibility is the use of exponential averaging over a time series of data.

We must also note that while each submesh pair has a unique channel between them, multiple channels may share the same network interfaces and network links. In this case, the control model may have to correct for the optimistic behavior across boundaries derived from point-to-point performance measurements that share physical infrastructure. It is also possible that the OCM could provide per boundary control information to the application, or it could provide aggregate control information across all boundaries. Once basic implementation is complete, these issues will be explored.

4.2 The Wren Measurement System

Setting the control knob to the correct level requires knowledge of the bandwidth and latency available in the environment the application is using. NWS [20] and Remos [15] are two systems that provide network measurements and predictions to applications. However, the grids we are focusing on and the needs of this application do not lend themselves well to either system's measurement approaches.

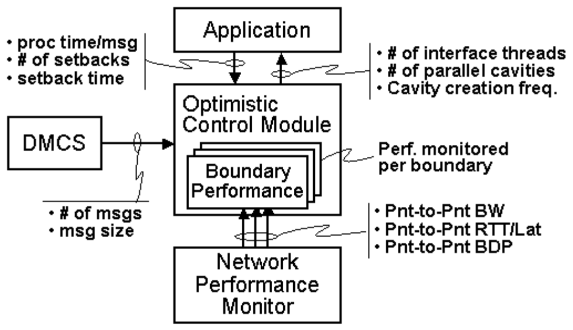

Figure 6: The optimistic control module for PREMA.

The grid environments the application will run on consist of a variety of clusters interconnected by LAN and WAN networks. The application's control knob must be tuned appropriately for each cluster and for each WAN connection—one setting will not work for the entire application. Therefore, we must have latency and bandwidth information available for connections within each cluster as well as between each cluster. NWS is well-suited to performing measurements between the clusters, but will not work as well running within a cluster while our application is running. Remos can monitor utilization within the cluster while the application is running, but cannot measure latency or monitor the actual performance an application can achieve running over the cluster.

We are developing the Wren network measurement system to provide the information needed to set the control knob for optimistic applications. Wren will combine active and passive measurements to effectively and efficiently monitor the communication performance of the cluster whether an application is running or not. Furthermore, Wren utilizes network topology information we gather to partition the grid into clusters with different network performance parameters, which the application requires to set the optimism control knob and LMPs as appropriate as performance and other machine characteristics vary within each cluster.

4.2.1 Passive Measurements at Runtime

Other projects have also looked at passive monitoring of application data [9, 11]. In the development of Wren, we are looking to expand on these approaches by using both active and passive measurements, as well as relying on measurement portability to allow different measurement techniques to be chosen according to available traffic, while preserving what appears to be a series of the same probe to other applications and services.

There are already significant results that indicate that it is possible to predict the value of one form of network measurement with another [16, 20]. To implement the passive approach at runtime on the machines used in the clusters, we are instrumenting the network stack within the 2.4 series of Linux kernels. The connection, sequence numbers, and flags of each packet are recorded with a timestamp as the packet is transferred to or from the network interface. This information is then made available to the user level through an interface in the /proc filesystem. Our implementation minimizes the overhead imposed at kernel level during program execution, and hopefully allows analysis to be scheduled when the application is waiting for data. The overhead imposed by the user-level code is minimal, and the data can be transferred to another machine for processing, as necessary.

We have implemented several different techniques for calculating available and achievable bandwidth using this instrumentation. The simplest is the bulk-transfer benchmark, where we observe the amount of data TCP successfully sends over a given interval of time, duplicating the functionality of traditional user-level probes within the kernel. These techniques are accurate as long as the application is sending sufficient data. Our other techniques are based on variants of the packet dispersion techniques. We have implemented techniques using either instrumentation at one or both ends, using data packets or ACK packets depending on which ends instrumentation is available. Figure 7 illustrates the correlation in the numbers produced by the different techniques in our implementation.

4.2.2 Topology-Based Steering

Part of the development of Wren focuses on using topology information collected for LANs and WANs to minimize the number of probes that must be sent to measure the inter-cluster network's performance. Using the topology knowledge to reduce the number of active probes sent will allow more frequent measurements of the path while still reducing the invasiveness of the traffic by decreas-

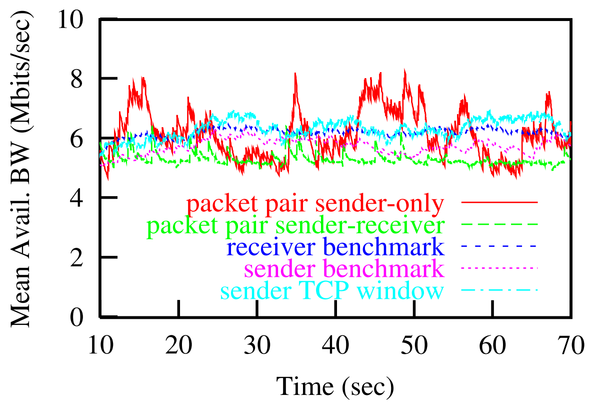

Figure 7: Available bandwidth calculated using different techniques for a pair of hosts exchanging data at full rate across a WAN.

ing the number of pairs of machines exchanging measurements.

Wren acquires the LAN topology information from Remos and uses that information to group the hosts into cliques consisting of the hosts attached to each edge switch. Wren will then group the cliques by measuring the network between the switches to determine if each clique should be merged, effectively considering it a single cluster, or left separate, indicating to the application that different control-knob settings will be required for each clique.

The combination of using passive measurements to observe the performance of the network while the mesh generation is running and using the topology to partition the grid used into clusters which require separate settings of the control knob allows us to optimize the performance of the mesh generation application over all of the machines it uses. The information collected allows the control settings to be updated as needed at runtime and for load-balancing or migration to optimize the application's performance. Although our application runs across the WAN, it still requires accurate information about each cluster and LAN to achieve optimal performance.

5 Summary

We have reported on a comprehensive middleware approach to support optimistic computation in grid environments in the specific context of mesh generation. We note that optimistic computation is one of several grid programming paradigms [13] that could be employed to make tightly coupled applications more amenable for grid execution. Hence, while we expect to see significant benefits for mesh generation, the ultimate goal is to generalize this approach such that it can be used, when appropriate, in other application domains.

6 Acknowledgments

This research is supported in part by ACI-0203974. In addition Chrisochoides' research is supported in part by Career Award grant No. CCR-9876179, ITR grant No. ACI-0085969, and RI grant No. EIA-9972853. Lowekamp's research is supported in part by the ARPA and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287. The data of the Figure 2 and Table 1 were collected with the assistance of K. Barker, A. Chernikov, and L. Linardakis. 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] K. Barker, N. Chrisochoides, A. Chernikov, and K. Pingali. Architecture and evaluation of a load balancing framework for adaptive and asynchronous 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] 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.

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

[6] 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.

[7] http://www.globus.org/mpi.

[8] D.A. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[9] Guojun Kin, George Yang, Brian R. Crowley, and Deborah A. Agarwal. Network characterization server (NCS). In *HPDC11*. IEEE, August 2001.

[10] Pattabhiraman Krishna, Nitin H. Vaidya, and Dhiraj K. Pradhan. Location management in distributed mobile environments. In *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 81–88, Washington, DC, 1994. IEEE Computer Society.

[11] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *In Proceedings of USENIX Symposium on Internet Technologies and Systems*, 2001.

[12] C. Lee and J. Stepanek. On future global grid communication performance. *10th IEEE Heterogeneous Computing Workshop*, May 2001.

[13] C. Lee and D. Talia. Grid programming models: Current tools, issues and directions. In Berman, Fox, and Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003. To appear.

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

[15] Bruce Lowekamp, Nancy Miller, Dean Sutherland, Thomas Gross, Peter Steenkiste, and Jaspal Subhlok. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 189–196. IEEE, July 1998.

[16] Bruce Lowekamp, David O'Hallaron, and Thomas Gross. Direct queries for discovering network resource properties in a distributed environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 38–46. IEEE Computer Society, August 1999.

[17] 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.

[18] 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.

[19] 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.

[20] Martin Swany and Rich Wolski. Multivariate resource performance forecasting in the network weather service. Baltimore, MD, November 2002.