Resource-Aware Scientific Computation on a Heterogeneous Cluster

James D. Teresco*
Department of Computer Science
Williams College
Williamstown, MA 01267

Jamal Faik and Joseph E. Flaherty
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

November 30, 2004

The popularity of cost-effective clusters built from commodity hardware has opened up a new platform for the execution of software originally designed for tightly-coupled supercomputers. Since these clusters can be built to include any number of processors from fewer than ten up to thousands, researchers in high-performance scientific computation at smaller institutions or in smaller departments may wish to maintain local parallel computing resources to support software development and testing. The software can later be moved to larger clusters and supercomputers. However, this has also led to the need for local expertise and effort to set up and maintain these clusters, and to develop software that can execute efficiently on both the smaller local clusters and on the larger ones. These target computing environments will vary in the number of processors, speed of processing and communication resources, size and speed of memory throughout the memory hierarchy. They may also differ in the availability of support tools and preferred programming paradigm. Moreover, software developed and optimized using a particular computing environment may not be efficient when it is moved to others.

1 The Cluster Setup

The cluster described herein, called the "Bullpen Cluster"[1], is located in the Department of Computer Science at Williams College. The majority of nodes in this cluster were purchased in 2001 from Sun Microsystems after soliciting offers from several companies. The initial cluster consisted of one Enterprise 220R server with one 450MHz Sparc UltraII processor and 512 MB memory, acting as file server and interactive login node; two Enterprise 420R servers, each with four 450MHz Sparc UltraII processors and 4 GB memory; and six Enterprise 220R servers, each with two 450MHz Sparc UltraII processors and 512 MB or 1 GB memory. Later, four Sun Ultra 10 Workstations, each with one 300 or 333 MHz Sparc UltraII processor, 128

*Corresponding author. E-mail: terescoj@cs.williams.edu, Tel: (413)597-4251, Fax: (413)597-4250
[1] `http://bullpen.cs.williams.edu/`

MB memory, and 6 GB local disk, were added. Figure 1 shows the cluster in its current configuration.

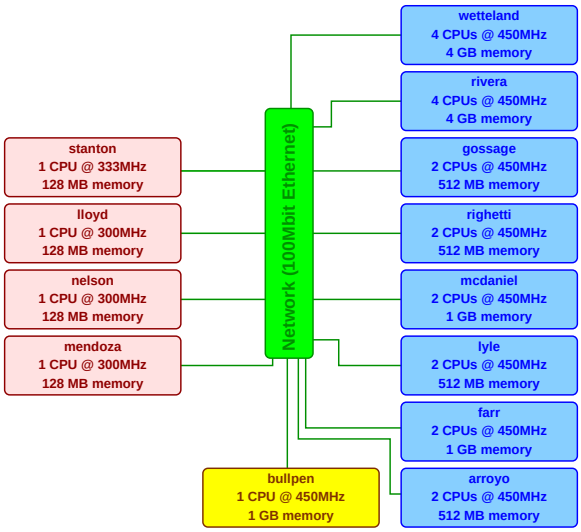


Figure 1: The "Bullpen Cluster" at Williams College.

This heterogeneous mix of nodes, with its variation in processor speeds and number of processors and amount of memory per node, was intended to allow the cluster to be used for studies of scientific computation in heterogeneous and hierarchical environments. Many clusters do not exhibit such heterogeneity when they are first built. There may be some network hierarchy, but typically a collection of identical nodes are acquired. However, clusters are often built incrementally, as opportunities arise to acquire new nodes. The old nodes, likely to provide some useful computing power, are retained as part of the cluster. This leads to heterogeneity. For the Bullpen cluster, the expansion actually involved the addition of slower nodes that were retired from the public labs. We expect future additions to this cluster to introduce further heterogeneity.

While this is not an especially large cluster, the power and cooling requirements (not to mention noise concerns) made it necessary to house it somewhere other than regular office space or in a public lab. A small closet in the computer science laboratory was upgraded with additional power and air conditioning and network ports to be able to house the cluster. In order to fit in the relatively small space, the server nodes are in racks and are connected by a common keyboard-video-mouse switch to a single keyboard and monitor, while the Ultra 10s are stacked and run without keyboards and monitors.

With a relatively small cluster, we decided to forego cluster management tools (e.g., the Sun Grid Engine[2], SCALI Manage[3]). The Solaris operating system's "jumpstart" system-installation utility proved sufficient to install and maintain consistent system software on all nodes. Jumpstart configuration files and installation scripts have been developed to allow installation or reinstallation of nodes to take place automatically by using a network boot.

Several implementations of the Message Passing Interface (MPI) [22] have been installed,

[2] `http://wwws.sun.com/software/gridware/`

[3] `http://www.scali.com/index.php?loc=17`

but MPICH [21] is used most frequently. Processes to be run on the production nodes are required to be executed through an installation of the OpenPBS[4] queueing system. The system allows a user to specify node types (number of processors per node, processor speed, memory requirements). The only tools that are not part of the operating system or freely available are the Solaris Forte compiler suite and the Etnus TotalView debugger[5].

While the setup and maintenance of the cluster has taken considerable time and effort, it has proven to be a useful resource both for research on heterogeneous and hierarchical computation, and for teaching parallel computing to undergraduates. The cluster has been used by Williams undergraduates in Parallel Processing and Operating Systems courses, and has been used occasionally by faculty and students in other courses and departments at Williams.

2 Parallel Adaptive Software

The primary purpose of the cluster, however, is as a testbed on which to run parallel adaptive scientific computation. Our focus is on solvers for systems of partial differential equations using finite element and related methods (e.g., [20, 32, 37]). These applications typically use meshes to discretize problem domains. The work of a simulation is associated with the entities (e.g., elements, surfaces, nodes) of the mesh. Parallelism is achieved by dividing these mesh entities among a number of subdomains and assigning subdomains to the cooperating processes, a procedure called mesh partitioning. The result of this process is shown in Figure 2. Adjacent mesh entities will need to pass messages to exchange information during

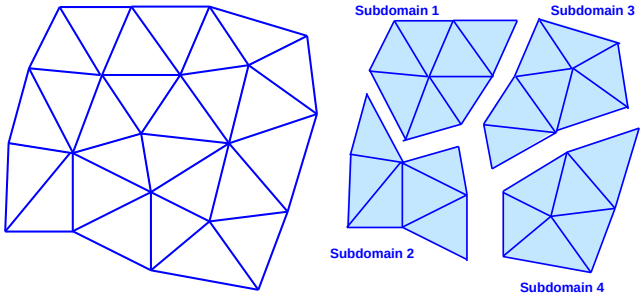


Figure 2: An example of a two-dimensional mesh (left) and a decomposition of the mesh into four subdomains (right).

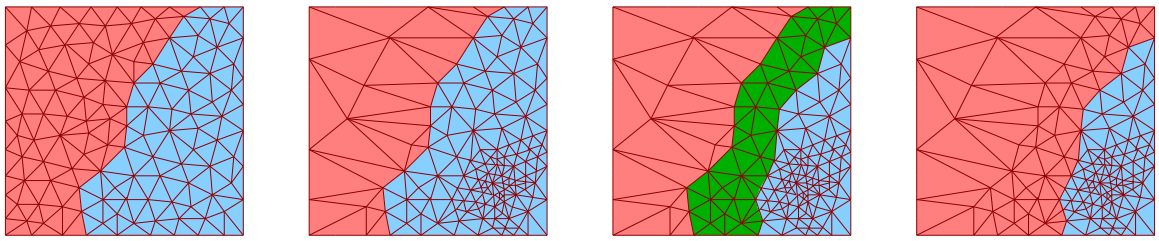
the solution process, so a mesh partitioner attempts to divide the work evenly among the processes while minimizing the number of pairs of adjacent entities which are assigned to different processes.

The problems of interest also use adaptive methods to improve time and space efficiency by concentrating computational effort in parts of the domain where it is needed to achieve a solution to a prescribed accuracy. This can take the form of h-refinement [40], where a mesh is refined or coarsened, respectively, in regions of low or high accuracy; or p-refinement [1, 39], where the method order is increased or decreased, respectively, in regions of low or high

[4] `http://www.openpbs.org/`

[5] `http://www.etnus.com/TotalView/`

accuracy. In either case, adaptive refinement introduces an imbalance in the partitioning, necessitating a dynamic load balancing step (Figure 3).



(a) (b) (c) (d)

Figure 3: The initial balanced mesh partitioning in (a) is disturbed by mesh adaptivity in (b). Mesh migration is needed in (c) to restore balance in (d).

A number of dynamic load balancing procedures have been developed, including recursive bisection methods [4, 41, 49], space-filling curve (SFC) partitioning [8, 31, 34, 35, 48] and graph partitioning (including spectral [36, 41], multilevel [7, 23, 26, 46], and diffusive methods [12, 24, 28]). Sandia National Laboratories' Zoltan Parallel Data Services Toolkit [14, 16] provides implementations of, or interfaces to, high-quality implementations of many of these dynamic load balancing procedures. Zoltan allows application developers to switch partitioners simply by changing a run-time parameter. Its design is "data-structure neutral." That is, Zoltan does not require applications to construct or use specific data structures. It operates on generic "objects" that are specified by calls to application-provided callback functions. These callbacks are simple functions that return to Zoltan information such as the lists of objects to be partitioned, coordinates of objects, and topological connectivity of objects.

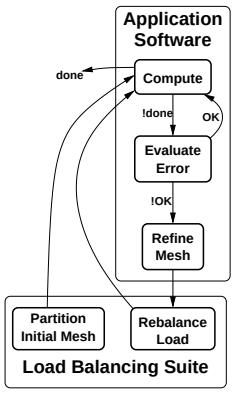


Figure 4: Program flow of a typical parallel adaptive computation. A load balancing suite such as Zoltan is used to partition the initial mesh. Application software takes computational steps, periodically evaluating error to determine if adaptive refinement is needed. If so, the refinement is performed, and the load balancing suite is called to rebalance.

Figure 4 shows the interaction between parallel adaptive application software and a dynamic load balancing suite such as that available in Zoltan. The load balancer is used to

partition the initial mesh. The application performs computation steps, periodically evaluating error estimates and checking against specified error tolerances. If the error is within the tolerance, the computation continues. Otherwise, an adaptive mesh refinement takes place, and the load balancer is called to compute a new partitioning before the computation resumes.

We target three software packages that we have been using for parallel adaptive computation for use in cluster environments. The first is called *LOCO* [20], which implements (in C) a parallel adaptive discontinuous Galerkin [5, 10, 11] solution of the compressible Euler equations. We consider specifically the "perforated shock tube" problem, which models the three-dimensional unsteady compressible flow in a cylinder containing a cylindrical vent [19]. The second package, called *DG* [37], also implements a parallel adaptive discontinuous Galerkin method, but in C++. DG is used to solve a wide range of problems including Rayleigh-Taylor flow instabilities [37]. The third package is Mitchell's Parallel Hierarchical Adaptive MultiLevel software (PHAML) [32]. PHAML is written in Fortran 90 and used to compute an adaptive solution of a Laplace equation on the unit square. While these three packages all take the same basic approach of partitioning the computational mesh and assigning the subdomains to the cooperating processes, each has its own underlying data structures and is capable of solving different problems. They each use Zoltan's partitioning and dynamic load balancing procedures.

Our goal is to run this software efficiently on heterogeneous clusters. Design decisions and optimizations were made based on the platform for which the software is first developed. The acceptance of the MPI standard has enhanced the portability of much of this software. Much of the software used herein was originally designed and developed at Rensselaer Polytechnic Institute for IBM SP systems. While much of the software has been used in other environments, the basic design remains from the initial implementations. We want to minimize the modifications needed to the existing software base, while improving efficiency in cluster environments.

3 Resource-Aware Computation

Moving from a tightly-coupled supercomputer to a cluster, or even from one cluster to another may reduce efficiency. Load imbalance may be introduced because of heterogeneous or non-dedicated processors. The relative costs of computation and communication may change, suggesting a different partitioning strategy. Deep memory hierarchies may be extended to include varying off-processor data access costs due to non-uniform memory access or hierarchical network structures. The Bullpen cluster added complications including nonuniform processor speeds, a mixture of 1-, 2-, and 4-processor nodes, and a slower network relative to processing speed than previous target platforms.

Any resource-aware computation must have knowledge of the computing environment, knowledge of software performance characteristics, and tools to make use of this knowledge. Our approach relies on a combination of a manual specification and automatic discovery of the characteristics of the computing environment. We evaluate the computing environment's performance using both *a priori* benchmark data and dynamic performance monitoring.

We can choose to use such knowledge of the computing environment at any of a number

of the common levels of abstraction. Application programmers can make high-level decisions in a resource-aware manner, e.g., through their choice of programming languages, parallel programming paradigm, or by adjusting memory management techniques. One of the most fundamental choices is the parallelization paradigm. The single-program multiple-data (SPMD) with message passing approach is often used because MPI is widely-available and highly portable. Other options include shared memory/multithreading [9, 29], a hybrid of SPMD with multithreading [3], the actor/theater model [2], and the Bulk Synchronous Parallel (BSP) [33] model. However, this choice is made early in development process and any change is likely to involve significant modification to application software and support libraries. Our software was developed with the SPMD model using MPI, so any optimizations will need to be done within this model.

Compiler developers and low-level tool developers (e.g., MPI implementers) can make resource-aware optimizations that are beneficial to a wide range of applications. These might include support for overlapping of communication and computation, reordering of computation and/or communication to take advantage of capabilities of the target environment [25], managing communication so that, given the buffer sizes and other characteristics of a particular interconnect, small messages can be concatenated and large messages can be split to achieve an optimal message size [30]. These approaches can be valuable, and can often be utilized without significant modification to applications.

Other tool developers, such as those designing and implementing partitioners and dynamic load balancers or numerical libraries, can make their software resource aware and benefit all users of their tools. These can also be used without significant modification to the existing code base, so we focus on the partitioning and dynamic load balancing procedures. Here, we can make tradeoffs for imbalance *vs.* communication minimization, or can adjust optimal partition sizes, and can partition to avoid communication across the slowest interfaces [18, 42, 44, 47].

4 Dynamic Resource Utilization Model

Our desire to support resource-aware load balancing, has led to the development of the Dynamic Resource Utilization Model (DRUM) [15, 18, 44]. We use existing application software and existing partitioning and dynamic load balancing tools, but use DRUM to determine the characteristics of the computing environment and to distill this information into a single "power" value, readily used by the load balancing procedures to produce appropriately-sized partitions.

The most straightforward way to account for heterogeneous processor speeds is to assign more of the work to faster processors, as those would otherwise be idle while slower processors are completing their work. This can be done with many existing load balancing procedures, including all procedures in Zoltan, by requesting different portions of the work be assigned to the partitions. Zoltan's load balancing procedures take an optional parameter which is an array of partition sizes. On the Bullpen cluster, for example, this can be accomplished by assigning 50% more work to the "fast"(450 MHz) processors than to "slow" (300 or 333 MHz) processors. An application can construct the array of partition sizes, if it can determine these relative processor speeds.

DRUM maintains such information about relative processor speeds. Processor "speed" (megahertz or gigahertz) ratings are not sufficient to determine relative processing powers. Other factors such as cache, memory and input/output subsystem performance also play important roles in determining how quickly a processor can perform computation. We measure processing power by running benchmarks. The benchmark by default uses LINPACK [17], but any program including the target application itself can be used to obtain the benchmark ratings. The benchmarks are run *a priori* either manually or from within DRUM's graphical configuration tool. The benchmarks are stored in a model of the computing environment

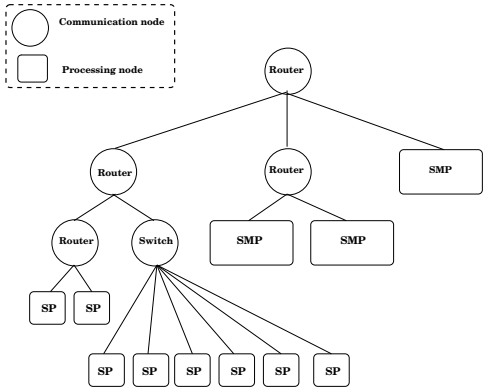


Figure 5: Tree constructed by DRUM to represent a heterogeneous cluster.

(Figure 5) that encapsulates information about hardware resources, their capabilities and their interconnection topology in a tree structure. The root of the tree represents the total execution environment. The children of the root node are high level divisions of different networks connected to form the total execution environment. Sub-environments are recursively divided, according to the network hierarchy, with the tree leaves being individual single-processor (SP) nodes or shared-memory multiprocessing (SMP) nodes. *Computation nodes* at the leaves of the tree have data representing their relative computing and communication power. *Network nodes*, representing routers or switches, have an aggregate power calculated as a function of the powers of their children and the network characteristics.

DRUM also provides a mechanism for dynamic monitoring and performance analysis. Monitoring agents in DRUM are threads that run concurrently with the user application to collect memory, network, and CPU utilization and availability statistics. Figure 6 shows the interaction among an application code, a load balancing suite such as Zoltan, and a resource monitoring system such as DRUM for a typical adaptive computation. The monitoring system gathers performance statistics during the application's execution. When load balancing is requested, the load balancer queries the monitoring system's performance analysis component to determine appropriate parameters and partition sizes for the rebalancing step.

A computation node's processing power is based on its static capabilities (as determined by the benchmarks) and monitored performance. We evaluate the processing power for each process on a given node based on (*i*) CPU utilization by the application process, (*ii*) the fraction of time that CPUs are idle, and (*iii*) the node's static benchmark rating. We assume

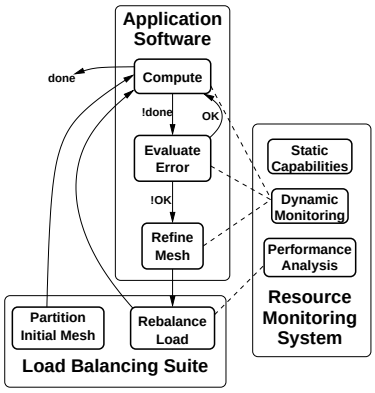


Figure 6: A typical interaction between an adaptive application code and a dynamic load balancing suite, where a resource monitoring system (e.g., DRUM) is monitoring application performance and system utilization during the application's execution and the dynamic load balancer queries the monitoring system to determine optimal partition sizes.

that idle time in the node can potentially be used by the application process, if it is assigned more work.

We also want to be able to account for heterogenous, hierarchical, and non-dedicated network resources. To do this, we estimate a node's communication power based on the communication traffic at the node. Agents estimate the average rate of incoming and outgoing network on each relevant communication interface. We view a node's communication power as inversely proportional to this "communication activity factor" at that node. Giving more work to a node with a larger communication power can take advantage of the fact that it is less busy with communication, so should be able to perform some extra computation while other nodes are in their communication phase.

The processing and communication powers are combined as a weighted sum to obtain the single value that can be used to request appropriately-sized partitions from the load balancer. The weights are now chosen manually to reflect an estimate of the relative costs of computation and communication for the application on a particular environment, but work is underway to automate this selection and to investigate other ways to combine the powers more effectively.

We have used DRUM in several studies using both the PHAML and DG application software [18]. These studies have confirmed that DRUM's dynamic monitoring agents introduce a very small overhead cost, and that DRUM-guided partitioning shows significant benefits over uniformly sized partitions, approaching, in many instances, the optimal relative change in execution times. We have found that for these applications on this cluster, only small weights (0–10%) should be given to communication power when computing overall node powers. We have also seen that DRUM can effectively adjust to dynamic changes, such as shared use of some nodes. This cannot be done with a static model that takes into account only node capabilities.

5 Hierarchical Partitioning and Load Balancing

While the approach in DRUM has been successful, it does not directly deal with hierarchical networks. We observe that each dynamic load balancing algorithm has characteristics and requirements that make it appropriate for certain applications [6, 45, 43]. For hierarchical and heterogeneous systems, different choices may be appropriate in different parts of the parallel environment. There are tradeoffs in execution time and partition quality (e.g., partition boundary sizes, interprocess connectivity, strictness of load balance) [45] and some may be more important than others in some circumstances. For example, consider a cluster of SMP nodes connected by Ethernet. A more costly graph partitioning can be done to partition among the nodes, to minimize communication across the slow network interface, possibly at the expense of some computational imbalance. Then, a fast geometric algorithm can be used to partition independently within each node. This is illustrated in Figure 7.

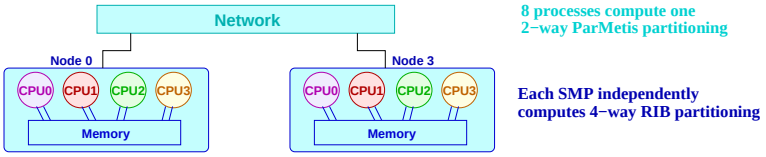


Figure 7: Hierarchical balancing algorithm selection for two 4-way SMP nodes connected by a network.

We have developed a Hierarchical Partitioning and Dynamic Load Balancing system (HIER) [44] within the Zoltan Toolkit that automates the creation of these hierarchical partitions. HIER is implemented entirely within Zoltan, but can be guided by DRUM's machine model.

The HIER implementation utilizes a lightweight "intermediate hierarchical balancing structure" (IHBS) and a set of callback functions. This allows us to compute hierarchical partitions automatically and efficiently, using any of the procedures available within Zoltan without modification and in any combination. HIER is easy to integrate into an application, as it is invoked in the same way as other Zoltan procedures. A hierarchical balancing step begins by building an IHBS using the same callbacks as those used for all Zoltan procedures. The IHBS is an augmented version of the distributed graph structure that Zoltan builds to make use of the ParMetis [27] and Jostle [46] libraries. HIER then provides its own callback functions, essentially "tricking" existing Zoltan procedures into operating on the IHBS at each level of a hierarchical balancing. After all levels of the hierarchical balancing have been completed, Zoltan's usual migration arrays are constructed and returned to the application. This allows HIER to migrate lightweight objects internally between levels, not the (larger and more costly) application data.

We have tested HIER using the LOCO solver on the Bullpen cluster. Here, the hierarchy comes from 2- and 4-processor nodes connected by fast Ethernet. We compared running times for an adaptive solution of a perforated shock tube problem in two subsets of the cluster: two 4-processor nodes, and four 2-processor nodes. We tried each combination of traditional and hierarchical procedures and found that while ParMetis multilevel graph partitioning alone often achieves the fastest computation times, there is some benefit to using

hierarchical load balancing where ParMetis is used for inter-node partitioning and inertial recursive bisection is used within each node. In cases where the top-level multilevel graph partitioner can find a good decomposition without introducing load imbalance, it provides the fastest time-to-solution in our studies to this point. When there is imbalance introduced at the top level, using a recursive bisection procedure to ensure strict balance within an SMP is beneficial. We expect that hierarchical balancing will be most beneficial when the extreme hierarchies, as found in larger clusters and in grid environments, are considered.

6 Discussion

DRUM and HIER have allowed application software of interest to run more efficiently, without modification, on the Bullpen cluster. Since these applications were already using Zoltan, the changes were entirely within DRUM and Zoltan to achieve the resource-aware partitions.

We are enhancing the DRUM library to run on a variety of computing environments. Since some of the monitoring uses fairly low-level operating system calls, some work is needed when the library is first installed on a new type of system. We are working to provide interfaces to other tools that may be available on a particular system (e.g., the Globus Monitoring and Discovery Service (MDS) [13], the Network Weather Service (NWS) [50], Ganglia [38]) but we do not intend to require any of these packages. We are also enhancing DRUM to include better ways to combine the processing and communication performance information into the single power value, and to include other information such as memory and cache availability and performance into the performance analysis. More information about DRUM is available at `http://www.cs.williams.edu/drum` and the software itself will be available for download at that location after recent extensions have been more thoroughly tested.

HIER is part of the Zoltan Toolkit and is expected to be available in a future public release. Zoltan can be downloaded from Sandia National Laboratories at `http://www.cs.sandia.gov/Zoltan`.

Acknowledgements

The development of DRUM and the hierarchical partitioning implementation in Zoltan was supported in part by contract 15162 with Sandia National Laboratories, a multi-program laboratory operation by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. The authors would like to thank William Mitchell for his help with the PHAML software. Karen Devine, Erik Boman, and Bruce Hendrickson of Sandia National Laboratories and Luis Gervasio of Rensselaer Polytechnic Institute have contributed to the design of DRUM.

References

[1] S. Adjerid, J. E. Flaherty, P. Moore, and Y. Wang. High-order adaptive methods for parabolic systems. *Physica-D*, 60:94–111, 1992.

[2] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[3] S. B. Baden and S. J. Fink. A programming methodology for dual-tier multicomputers. *IEEE Transactions on Software Engineering*, 26(3):212–216, 2000.

[4] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36:570–580, 1987.

[5] R. Biswas, K. D. Devine, and J. E. Flaherty. Parallel, adaptive finite element methods for conservation laws. *Appl. Numer. Math.*, 14:255–283, 1994.

[6] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, M. Heroux, and R. Preis. LDRD report: Parallel repartitioning for optimal solver performance. Technical Report SAND2004–0365, Sandia National Laboratories, Albuquerque, NM, February 2004.

[7] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization". In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452. SIAM, 1993.

[8] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.

[9] R. Chandra, R. Menon, L. Dagum, D. Koh, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP.* Morgan Kaufmann, 2000.

[10] B. Cockburn, S.-Y. Lin, and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-Dimensional systems. *J. Comput. Phys.*, 84:90–113, 1989.

[11] B. Cockburn and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: General framework. *Math. Comp.*, 52:411–435, 1989.

[12] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7:279–301, 1989.

[13] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.

[14] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[15] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. Technical Report Technical Report CS-04-02, Williams College Department of Computer Science, 2004. To appear, *Appl. Numer. Math.*

[16] K. D. Devine, B. A. Hendrickson, E. Boman, M. St. John, and C. Vaughan. *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377. Open-source software distributed at `http://www.cs.sandia.gov/Zoltan`.

[17] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, 1979.

[18] J. Faik, L. G. Gervasio, J. E. Flaherty, J. Chang, J. D. Teresco, E. G. Boman, and K. D. Devine. A model for resource-aware load balancing on heterogeneous clusters. Technical Report CS-04-03, Williams College Department of Computer Science, 2004. Presented at Cluster '04.

[19] J. E. Flaherty, R. M. Loy, M. S. Shephard, M. L. Simone, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Distributed octree data structures and local refinement method for the parallel solution of three-dimensional conservation laws. In M. Bern, J. Flaherty, and M. Luskin, editors, *Grid Generation and Adaptive Algorithms*, volume 113 of *The IMA Volumes in Mathematics and its Applications*, pages 113–134, Minneapolis, 1999. Institute for Mathematics and its Applications, Springer.

[20] J. E. Flaherty, R. M. Loy, M. S. Shephard, and J. D. Teresco. Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In B. Cockburn, G. Karniadakis, and S.-W. Shu, editors, *Discontinous Galerkin Methods Theory, Computation and Applications*, volume 11 of *Lecture Notes in Compuational Science and Engineering*, pages 113–124, Berlin, 2000. Springer.

[21] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[22] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1999.

[23] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*, 1995.

[24] Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. Preprint DL-P-95-011, Daresbury Laboratory, Warrington, WA4 4AD, UK, 1995.

[25] N. T. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the Message Passing Interface. *J. Parallel Distrib. Comput.*, 63(5):551–563, May 2003.

[26] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scien. Comput.*, 20(1), 1999.

[27] G. Karypis and V. Kumar. Parallel multivelel k-way partitioning scheme for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[28] E. Leiss and H. Reddy. Distributed load balancing: design and performance analysis. *W. M. Kuck Research Computation Laboratory*, 5:205–270, 1989.

[29] B. Lewis and D. J. Berg. *Multithreaded Programming with pthreads*. Sun Microsystems Press, 1997.

[30] R. M. Loy. AUTOPACK version 1.2. Technical Memorandum ANL/MCS-TM-241, Mathemetics and Computer Science Division, Argonne National Laboratory, 2000.

[31] W. F. Mitchell. Refinement tree based partitioning for adaptive grids. In *Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computing*, pages 587–592. SIAM, 1995.

[32] W. F. Mitchell. The design of a parallel adaptive multi-level code in Fortran 90. In *Proc. 2002 International Conference on Computational Science*, 2002.

[33] M. Nibhanapudi and B. K. Szymanski. *High Performance Cluster Computing*, volume I of *Architectures and Systems*, chapter BSP-based Adaptive Parallel Processing, pages 702–721. Prentice Hall, New York, 1999.

[34] A. Patra and J. T. Oden. Problem decomposition for adaptive *hp* finite element methods. *Comp. Sys. Engng.*, 6(2):97–109, 1995.

[35] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Trans. on Parallel and Distributed Systems*, 7(3):288–300, 1996.

[36] A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430–452, 1990.

[37] J.-F. Remacle, J. Flaherty, and M. Shephard. An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Review*, 45(1):53–72, 2003.

[38] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide area cluster monitoring with Ganglia. In *Proc. IEEE Cluster 2003*, Hong Kong, 2003.

[39] M. S. Shephard, S. Dey, and J. E. Flaherty. A straightforward structure to construct shape functions for variable p-order meshes. *Comp. Meth. in Appl. Mech. and Engng.*, 147:209–233, 1997.

[40] M. S. Shephard, J. E. Flaherty, H. L. de Cougny, C. Özturan, C. L. Bottasso, and M. W. Beall. Parallel automated adaptive procedures for unstructured meshes. In *Parallel Comput. in CFD*, number R-807, pages 6.1–6.49. Agard, Neuilly-Sur-Seine, 1995.

[41] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Comp. Sys. Engng.*, 2:135–148, 1991.

[42] S. Sinha and M. Parashar. Adaptive system partitioning of AMR applications on heterogeneous clusters. *Cluster Computing*, 5(4):343–352, October 2002.

[43] J. D. Teresco, K. D. Devine, and J. E. Flaherty. Partitioning and dynamic load balancing for the numerical solution of partial differential equations. Technical Report CS-04-11, Williams College Department of Computer Science, 2005. Chapter submitted to *Numerical Solution of Partial Differential Equations on Parallel Computers*, Are Magnus Bruaset, Petter Bjørstad, Aslak Tveito, editors.

[44] J. D. Teresco, J. Faik, and J. E. Flaherty. Hierarchical partitioning and dynamic load balancing for scientific computation. Technical Report CS-04-04, Williams College Department of Computer Science, 2004. Submitted to Proc. PARA '04.

[45] J. D. Teresco and L. P. Ungar. A comparison of Zoltan dynamic load balancers for adaptive computation. Technical Report CS-03-02, Williams College Department of Computer Science, 2003. Presented at COMPLAS '03.

[46] C. Walshaw and M. Cross. Parallel Optimisation Algorithms for Multilevel Mesh Partitioning. *Parallel Comput.*, 26(12):1635–1660, 2000.

[47] C. Walshaw and M. Cross. Multilevel Mesh Partitioning for Heterogeneous Communication Networks. *Future Generation Comput. Syst.*, 17(5):601–623, 2001. (originally published as Univ. Greenwich Tech. Rep. 00/IM/57).

[48] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proc. Supercomputing '93*, pages 12–21. IEEE Computer Society, 1993.

[49] R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency*, 3:457–481, October 1991.

[50] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Comput. Syst.*, 15(5-6):757–768, October 1999.