Resource-aware Parallel Adaptive Computation for Clusters

James D. Teresco, Laura Effinger-Dean and Arjun Sharma

Department of Computer Science, Williams College
Williamstown, MA 01267 USA
terescoj@cs.williams.edu

Abstract. Smaller institutions can now maintain local cluster computing environments to support research and teaching in high-performance scientific computation. Researchers can develop, test, and run software on the local cluster and move later to larger clusters and supercomputers at an appropriate time. This presents challenges in the development of software that can be run efficiently on a range of computing environments from the (often heterogeneous) local clusters to the larger clusters and supercomputers. Meanwhile, the clusters are also valuable teaching resources. We describe the use of a heterogeneous cluster at Williams College and its role in the development of software to support scientific computation in such environments, including two summer research projects completed by Williams undergraduates.

Cluster computing environments at smaller institutions have provided a new platform for research and teaching in high-performance computing. Such local computing resources support development of software which can be executed on the local cluster or can be moved later to larger clusters or supercomputers for execution of larger problems. Meanwhile, clusters provide valuable local resources for teaching and the support of student projects. This paper describes a cluster at Williams College and provides an overview of a research effort that has been motivated and supported by this cluster, in particular two undergraduate projects which have contributed to this effort.

1 A Cluster Environment

Our cluster (known as "Bullpen"[1]) is located in the Department of Computer Science at Williams College. It consists of one Sun Enterprise 220R server with one 450MHz Sparc UltraII processor; two Enterprise 420R servers, each with four 450MHz Sparc UltraII processors; and six Enterprise 220R servers, each with two 450MHz Sparc UltraII processors; and four Sun Ultra 10 Workstations, each with one 300 or 333 MHz Sparc UltraII processor.

This cluster is intentionally heterogeneous, with its nodes having different processor speeds, numbers of processors and amount of memory per node. This makes it an excellent platform for studies of scientific computation in heterogeneous and hierarchical

[1] http://bullpen.cs.williams.edu/

environments. While most clusters are initially built using identical nodes, incremental growth is an attractive features of clusters. As new (likely faster) nodes are added, old nodes remain part of the cluster, leading to heterogeneity.

In addition to the support of the research described herein, this cluster has been used in Computer Science courses at Williams, most extensively in the Parallel Processing course. Students have been able to write multithreaded code using both POSIX threads [5] and OpenMP[2] to use the symmetric multiprocessing (SMP) nodes. They have used the Message Passing Interface (MPI)[3] to use multiple nodes to perform parallel computation with distributed memory and message passing. Student projects have included a parallel discrete event simulation, parallel particle simulations, a parallel photon mapper and a parallel ray tracer. Having the local cluster available meant that the students were not competing for processor cycles on lab workstations and did not have to develop software remotely at a supercomputing center.

2 Parallel Adaptive Computation on Clusters

Our goal is to develop tools and techniques to allow efficient parallel adaptive scientific computation on heterogeneous clusters such as Bullpen. We focus on solvers for systems of partial differential equations using finite element and related methods (e.g., [4, 6, 7]) that use meshes to discretize problem domains. The mesh is partitioned into subdomains consisting of disjoint subsets of mesh entities (e.g., elements, surfaces, nodes) and these subdomains are assigned to the cooperating processes of a parallel computation. Adjacent mesh entities will exchange information during the solution process. So in addition to its attempts to divide the work evenly among the processes (to achieve load balance), a mesh partitioner attempts to minimize the number of pairs of adjacent entities which are assigned to different processes (to minimize interprocess communication).

The methods are adaptive, where time and space efficiency is improved by concentrating computational effort in parts of the domain where it is needed to achieve a solution to a prescribed accuracy [1]. However, adaptivity will disturb a balanced partitioning, necessitating a dynamic load balancing step. Dynamic load balancing procedures have similar goals to mesh partitioners, but must operate on already-distributed data and should minimize the change between the existing decomposition and the new decomposition (to limit mesh migration).

A number of approaches to dynamic load balancing have been proposed ([8] includes a survey). The Zoltan Parallel Data Services Toolkit [2] provides a common interface to high-quality implementations of several such procedures. With Zoltan, applications can quickly make use of and can easily switch among available load balancing methods. Fig. 1 shows the interaction between parallel adaptive application software and a dynamic load balancing suite such as that in Zoltan. After an initial partitioning, the application performs computation steps, periodically evaluating error estimates and checking against specified error tolerances. If the error is within tolerance, the compu-

[2] http://www.openmp.org
[3] http://www-unix.mcs.anl.gov/mpi/

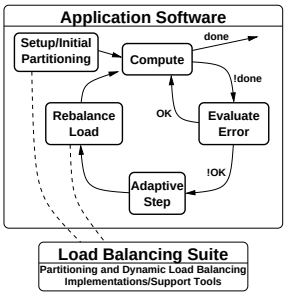

Fig. 1. Program flow of a typical parallel adaptive computation using a load balancing suite such as Zoltan.

tation continues. Otherwise, an adaptive refinement takes place, followed by dynamic load balancing before the computation resumes.

Our goal is to run parallel adaptive computations efficiently on heterogeneous clusters, while making minimal changes to the application software. We have been working with three software packages in cluster environments. *LOCO* [4] and *DG* [7] implement parallel adaptive discontinuous Galerkin procedures. The Parallel Hierarchical Adaptive MultiLevel software (PHAML) [6] implements a variety of parallel adaptive solution procedures. Each of these uses Zoltan's dynamic load balancing procedures.

3 Resource-Aware Load Balancing

In cluster environments, load imbalance may be introduced because of heterogeneous or non-dedicated processors. The relative costs of computation and communication may change from one environment to the next, suggesting a different partitioning strategy. On Bullpen, we are faced with nonuniform processor speeds, the mixture of 1-, 2-, and 4-processor nodes, and a slower network relative to processing speed than previous target platforms. A *resource-aware* computation, which requires knowledge of the computing environment and tools to make use of this knowledge, is needed to take full advantage of the computing environment. Resource-aware adjustments can be made anywhere from low-level tools to application programs (see [11] for examples).

Our focus is on resource-aware dynamic load balancing, guided by the Dynamic Resource Utilization Model (DRUM) [3, 10][4]. Processor "speed" (megahertz or gigahertz) ratings must be combined with other factors such as cache, memory and input/output subsystem performance, and current usage to determine how quickly a processor can perform computation. DRUM evaluates the computing environment's performance using data from both benchmarks which are run *a priori* either manually or from within DRUM's graphical configuration tool (Section 4) and a dynamic performance monitors. DRUM distills this information into a single "power" value, readily used by load balancing procedures (including all Zoltan procedures) to produce appropriately-sized partitions.

[4] DRUM web page: http://www.cs.williams.edu/drum/

Benchmark results are stored in a model of the computing environment 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 symmetric 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.

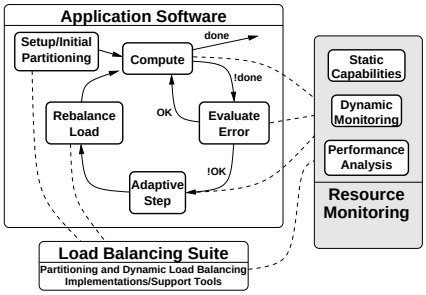

Fig. 2. A typical interaction between an adaptive application code and a dynamic load balancing suite, when using a resource monitoring system (e.g., DRUM).

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. Fig. 2 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. 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.

DRUM can also adjust for heterogenous, hierarchical, and non-dedicated network resources by estimating a node's communication power based on the communication traffic at the node. Information about network interfaces may be gathered using kernel statistics, a more portable but still limited library called `net-snmp`[5], which implements the Simple Network Management Protocol (SNMP), or the Network Weather Service (NWS) [13] (Section 5). 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 communication power is combined with processing power as a weighted sum to obtain the single value that can be used to request appropriately-sized partitions from the load balancer.

[5] `http://www.net-snmp.org`

We have used DRUM to guide resource-aware load balancing for both the PHAML and DG application software. DRUM-guided partitioning shows significant benefits over uniformly sized partitions, approaching, in many instances, the optimal relative change in execution times. 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. Our focus in this paper is on the two DRUM enhancements described in the following sections; see [3] and [10] for performance studies using DRUM.

4 A Graphical Configuration Tool for DRUM

DRUM constructs its run-time model of the computing environment using information stored in an XML-format configuration file that describes properties of the system (e.g., benchmark results, network topology). We have developed a graphical configuration program in Java called *DrumHead* that aids in the construction of these configuration files[6]. DrumHead can be used to draw a description of a cluster, automatically run the benchmarks on the cluster nodes, and then create the configuration file for DRUM to read in when constructing its model. Fig. 3 shows an excerpt from an XML configuration file generated by DrumHead for the Bullpen Cluster configuration.

```
<machinemodel>
<node type="NETWORK" nodenum="0" name="" IP="" isMonitorable="false"
parent="-1" imgx="361.0" imgy="52.0">
<lbmethod lbm="HSFC" KEEP_CUTS="1"></lbmethod></node>
<node type="SINGLE_COMPUTING" nodenum="2" name="mendoza.cs.williams.edu"
IP="137.165.8.140"  isMonitorable="true" parent="0"
benchmark="52.43" imgx="50.0" imgy="138.0"></node>
<node type="MULTIPLE_COMPUTING" nodenum="3" name="rivera.cs.williams.edu"
IP="137.165.8.130" isMonitorable="false" parent="0"
imgx="74.64 "imgy="263.0" benchmark="82.55" numprocs="4">
<lbmethod lbm="HSFC" KEEP_CUTS="1">
</lbmethod></node>
...
</machinemodel>
```

Fig. 3. An excerpt from a configuration file generated by DrumHead for Bullpen.

The layout of the main window (Fig. 4) is simple: a panel of tools and buttons on the left and a workspace (starting out empty) on the right. The tool pane shows the current properties of the entire cluster, all the changeable features of the selected node and buttons to save changes to the selected node's parameters. In the middle pane, the user can draw nodes, represented by rectangles (the computing nodes) and ovals (networking nodes), connected by lines. These nodes can be dragged, so the user can place them in a meaningful arrangement.

[6] The design and implementation of DrumHead was part of the research project of Williams undergraduate Arjun Sharma during Summer 2004.

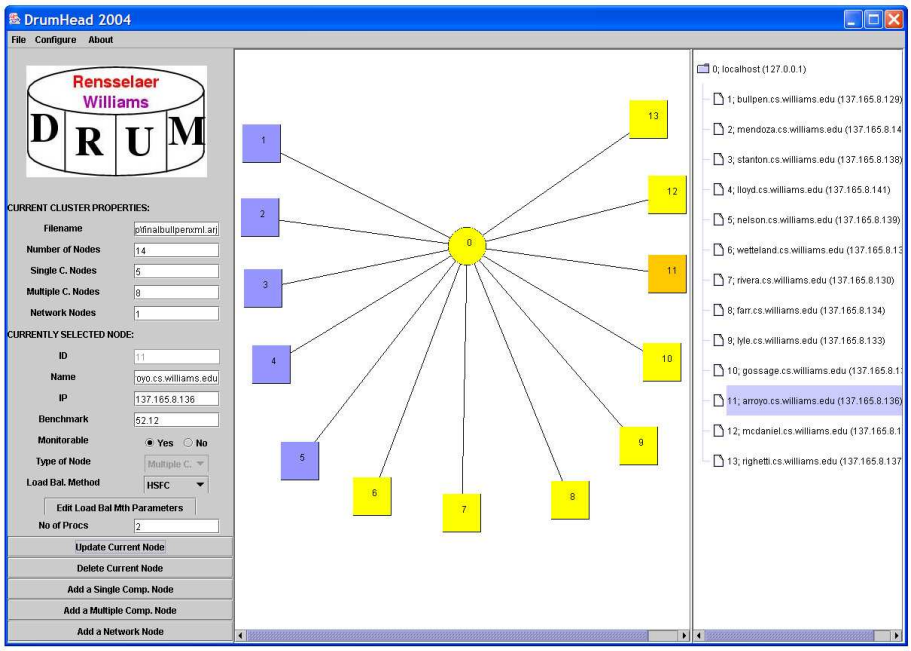

Fig. 4. DrumHead editing a description of the Bullpen Cluster.

DrumHead allows specification of dynamic load balancing methods and parameters for network and SMP computing nodes. These parameters can be used by DRUM to guide a hierarchical load balancing, where different load balancing procedures are used in different parts of the computing environment. The available procedures present trade-offs in execution time and partition quality (*e.g.*, surface indices, interprocess connectivity, strictness of load balance) [12] and some may be more important than others in some circumstances. For example, consider a run using two or more of Bullpen's SMP nodes. A more costly graph partitioning can be done to partition among the SMPs, 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. Hierarchical balancing, which is implemented in Zoltan, is described in detail in [9].

5 Interface to the Network Weather Service

DRUM is intended to work on a variety of architectures and operating systems. We do not want to require that DRUM users install additional software packages, but we do want DRUM to take advantage of such packages when available. We have developed an interface that allows DRUM to access information from NWS[7], which provides infor-

[7] The implementation of the DRUM interface to NWS was part of the research project of Williams undergraduate Laura Effinger-Dean during Summer 2004.

mation about network and CPU usage for Unix-based systems. NWS is more intrusive than DRUM's other network monitoring capabilties, as it will send its own messages to measure network status.

NWS uses a set of "sensor" servers which run separately on each node of the parallel system, interacting with a "nameserver" and one or more "memory" servers. The nameserver allows easy searching for servers ("hosts"), sensor resources ("activities" or "skills"), and previously-collected data ("series"). For instance, to search for statistics about bandwidth between machine **A** and machine **B**, you would query the nameserver for an object with properties `objectClass` "nwsSeries," `resource` "bandwidthTcp," `host` "A:8060," and `target` "B:8060," where 8060 is the port used by the sensors on **A** and **B**. Network data is gathered within "cliques" of nodes: sets of machines which trade packets to measure bandwidth, connect time, and latency. The concept of cliques fits well with DRUM's tree model, as a clique may be defined as all leaves of a network node.

DRUM relies on the user or system administrator to configure and launch the appropriate NWS servers on each node within the parallel system. NWS activities could be started from within DRUM, but this would be ineffective early in a computation as NWS needs at least a few minutes to collect enough data to provide useful information. When it needs to gather network statistics from NWS, DRUM searches the nameserver for available "bandwidthTcp" series and randomly selects three. These series are limited to those whose host is the current machine and whose target shares a parent node with the host. From these three series, DRUM calculates the communication power of the node based one of three methods: an average of 20 measurements, the most recent single measurement, or an NWS "forecast," which essentially provides a normalized estimate of bandwidth, undisturbed by small variations. This bandwidth calculation substitutes for the "communication activity factor" used by the kstat- and SNMP-based implementations for the determination of communication powers and weights in DRUM's overall power formulas [3].

Acknowledgments

Teresco was supported in part by Sandia contract PO15162 and the Computer Science Research Institute at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. Effinger-Dean and Sharma were supported by the Williams College Summer Science Research program. DRUM was developed with Jamal Faik (Rensselaer). Erik Boman, Karen Devine, and Bruce Hendrickson (Sandia) and Luis Gervasio (Rensselaer) also contributed to the design of DRUM.

References

1. K. Clark, J. E. Flaherty, and M. S. Shephard. *Appl. Numer. Math., special ed. on Adaptive Methods for Partial Differential Equations*, 14, 1994.

2. 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.
3. J. Faik, J. D. Teresco, K. D. Devine, J. E. Flaherty, and L. G. Gervasio. A model for resource-aware load balancing on heterogeneous clusters. Technical Report CS-05-01, Williams College Department of Computer Science, 2005. Submitted to Transactions on Parallel and Distributed Systems.
4. 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.
5. B. Lewis and D. J. Berg. *Multithreaded Programming with pthreads*. Sun Microsystems Press, 1997.
6. W. F. Mitchell. The design of a parallel adaptive multi-level code in Fortran 90. In *International Conference on Computational Science (3)*, volume 2331 of *Lecture Notes in Computer Science*, pages 672–680. Springer, 2002.
7. 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.
8. J. D. Teresco, K. D. Devine, and J. E. Flaherty. *Numerical Solution of Partial Differential Equations on Parallel Computers*, chapter Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations. Springer-Verlag, 2005.
9. 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.
10. J. D. Teresco, J. Faik, and J. E. Flaherty. Resource-aware scientific computation on a heterogeneous cluster. Technical Report CS-04-10, Williams College Department of Computer Science, 2005. To appear, *Computing in Science & Engineering*.
11. J. D. Teresco, J. E. Flaherty, S. B. Baden, J. Faik, S. Lacour, M. Parashar, V. E. Taylor, and C. A. Varela. Approaches to architecture-aware parallel scientific computation. Technical Report CS-04-09, Williams College Department of Computer Science, 2005. Submitted to *Proc. PP'04: Frontiers of Scientific Computing*.
12. 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.
13. 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.