

RESOURCE-AWARE SCIENTIFIC COMPUTATION ON A HETEROGENEOUS CLUSTER

Although researchers can develop software on small, local clusters and move it later to larger clusters and supercomputers, the software must run efficiently in both environments. Two efforts aim to improve the efficiency of scientific computation on clusters through resource-aware dynamic load balancing.

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. Because these clusters can be built to include any number of processors ranging from fewer than 10 to thousands, researchers in high-performance scientific computation at smaller institutions or in smaller departments can maintain local parallel computing resources to support software development and testing, then move the software to larger clusters and supercomputers.

As promising as this ability is, it has also led to the need for local expertise and resources to set up and maintain these clusters. The software must execute efficiently both on smaller local clusters and on larger ones. These computing environments vary in the number of processors, speed of processing and communication resources, and size and speed of memory throughout the memory hierar-

chy as well as in the availability of support tools and preferred programming paradigms. Software developed and optimized using a particular computing environment might not be as efficient when it's moved to another one.

In this article, we describe a small cluster along with two efforts to improve the efficiency of parallel scientific computation on that cluster. Both approaches modify the dynamic load-balancing step of an adaptive solution procedure to tailor the distribution of data across the cooperating processes. This modification helps account for the heterogeneity and hierarchy in various computing environments.

The Cluster Setup

The Bullpen cluster (<http://bullpen.cs.williams.edu/>) is located in the Department of Computer Science at Williams College. The initial cluster, constructed in 2001, consisted of one Enterprise 220R server with a 450-MHz Sparc UltraII processor and 512 Mbytes of memory, acting as both a file server and an interactive login node; two Enterprise 420R servers, each with four 450-MHz Sparc UltraII processors and 4 Gbytes of memory; and six Enterprise 220R servers, each with two 450-MHz Sparc UltraII processors and 512 Mbytes or 1 Gbyte of memory. Later, we added four Sun Ultra 10 workstations, each with one 300- or 333-MHz Sparc UltraII processor, 128 Mbytes of memory, and 6 Gbytes of local disk space. Figure 1

1521-9615/05/\$20.00 © 2005 IEEE
Copublished by the IEEE CS and the AIP

JAMES D. TERESCO

Williams College

JAMAL FAIK AND JOSEPH E. FLAHERTY

Rensselaer Polytechnic Institute

shows the cluster in its current configuration.

This heterogeneous mix of nodes, with its variation in numbers of processors, processor speeds, and amount of memory per node, was intended to allow the cluster to be used for studies of scientific computation in different environments. Many clusters don't exhibit much heterogeneity when they're first built, but as they acquire new nodes—and retain the old ones for computing power—heterogeneity appears. For the Bullpen cluster, our expansion involved the addition of slower nodes retired from public labs; we expect that future additions to this cluster will introduce further heterogeneity.

Although ours isn't an especially large cluster, its power and cooling requirements (not to mention noise concerns) made it necessary to house it somewhere other than in regular office space or in a public lab. Accordingly, we upgraded a small closet in the computer science laboratory with additional power, air conditioning, and network ports. To fit it into this relatively small space, we put the server nodes in racks and connected them with a common keyboard-video-mouse switch to a single keyboard and monitor; we also stacked the Ultra 10s but ran them without keyboards and monitors.

With this relatively small cluster, we decided to forego cluster management tools such as the Sun Grid Engine (<http://www.sun.com/software/gridware/>) or SCALI Manage (www.scali.com/index.php?loc=17). The Solaris operating system's "jumpstart" system-installation utility used a network boot to automatically install and maintain consistent system software on all nodes.

We installed several implementations of the message-passing interface (MPI),¹ but we tend to use MPICH (<http://www-unix.mcs.anl.gov/mpi/mpich/>) most frequently. Processes to be run on the production nodes must be executed through an installation of the OpenPBS queuing system (www.openpbs.org), which lets the user specify node types (number of processors per node, processor speed, and memory requirements). The only tools that aren't part of the operating system or aren't freely available are the Solaris Forte compiler suite and the Etnus TotalView debugger (www.etnus.com/TotalView/).

Although the cluster's setup and maintenance has taken considerable time and effort, it has proven to be a useful resource for research on heterogeneous and hierarchical computation and for teaching parallel computing to undergraduates. Williams undergraduates in parallel processing and operating systems courses have used it, as have faculty and students in other courses and departments on campus.

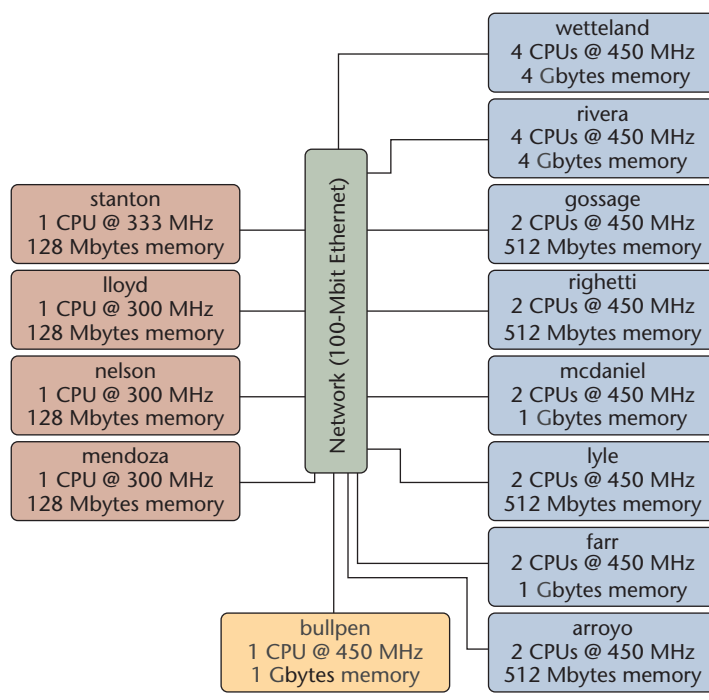


Figure 1. The Bullpen cluster at Williams College. Yellow indicates the server (Bullpen), blue shading indicates fast nodes, and brown indicates slower ones.

Parallel Adaptive Software

The cluster's primary purpose is as a test bed on which to run parallel adaptive scientific computation—specifically, solvers for systems of partial differential equations using finite-element and related methods.²⁻⁴ These applications typically use meshes to discretize problem domains. Because a simulation's work is usually associated with a mesh's entities (elements, surfaces, and nodes), we achieve parallelism by dividing these entities among several subdomains and then assigning them to cooperating processes, a procedure called *mesh partitioning*. Figure 2 shows the result of this process for a small two-dimensional mesh. Adjacent mesh entities pass messages to exchange information during 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 assigned to different processes.

The typical 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*,⁵ in which a mesh is refined or coarsened in regions of low or high accuracy, respectively, or *p-refinement*,⁶ in which the method order is increased or decreased in regions of low or

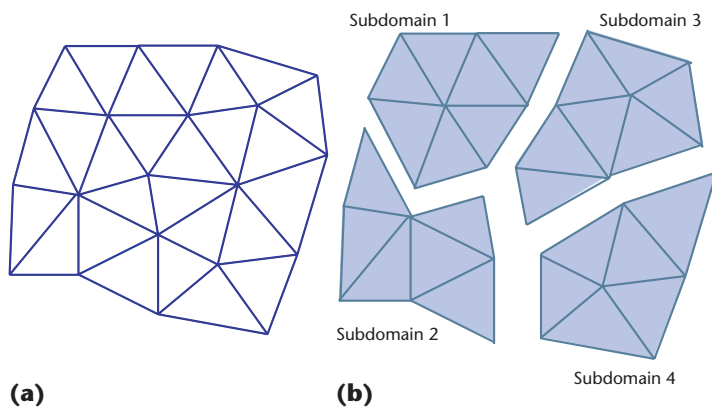


Figure 2. Mesh partitioning. To achieve parallelism, a mesh partitioner divides the mesh's entities into subdomains: (a) the two-dimensional mesh here is (b) decomposed into four subdomains, with each mesh element (face) assigned uniquely to a subdomain.

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

Several dynamic load-balancing procedures exist, including recursive bisection methods, space-filling curve (SFC) partitioning, spectral and multilevel graph partitioning, and various diffusive methods.⁷ Sandia National Laboratories' Zoltan Parallel Data Services Toolkit⁸ provides a common interface to high-quality implementations of many of these procedures. Some are implemented directly within Zoltan, whereas others are accessible through interfaces to existing implementations in third-party libraries. Zoltan lets application developers switch partitioners simply by changing a runtime parameter. Its design is data-structure neutral—that is, Zoltan doesn't require applications to construct or use specific data structures. Rather, it operates on generic "objects" 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, the coordinates of these objects, and the objects' topological connectivity.

Figure 4 shows the interaction between parallel adaptive application software and a dynamic load-balancing suite such as Zoltan. The load balancer partitions the initial mesh, then the application performs its 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 occurs, and the load balancer is called to compute a new partitioning before the computation resumes.

We use three software packages for parallel adaptive computation in cluster environments. The first is LOCO,² which implements in C a parallel adaptive discontinuous Galerkin⁹ solution of the compressible Euler equations. We consider specifically the perforated shock-tube problem, which models the 3D unsteady compressible flow in a cylinder containing a cylindrical vent.¹⁰ The second package, DG,⁴ also implements a parallel adaptive discontinuous Galerkin method, but in C++. DG also helps solve a wide range of problems, including Rayleigh-Taylor flow instabilities.⁴ The third package is Mitchell's Parallel Hierarchical Adaptive MultiLevel software (PHAML),³ which is written in Fortran 90 and used to compute an adaptive solution of a Laplace equation on the unit square. Although these three packages all take the same basic approach to 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 also use Zoltan's partitioning and dynamic load-balancing procedures.

The different software programs' designers developed their products with certain platform optimizations in mind. Fortunately, the acceptance of the MPI standard has enhanced this software's portability. Much of our software was originally designed and developed at Rensselaer Polytechnic Institute for IBM SP systems, and even though it's been used in other environments, the basic design remains from the initial implementations. Ultimately, we want to improve efficiency in cluster environments while minimizing the modifications needed to the existing software base.

Resource-Aware Computation

Moving from a tightly coupled supercomputer to a cluster, or even from one cluster to another, can reduce efficiency and introduce load imbalance because of heterogeneous or nondedicated processors. Moreover, the relative costs of computation and communication can change, suggesting a different partitioning strategy. Varying off-processor data access costs due to nonuniform memory access or hierarchical network structures can be thought of as extensions of the memory hierarchy. The Bullpen cluster had several complications, including nonuniform processor speeds, a mixture of one-, two-, and four-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 and the software's performance characteristics as well as

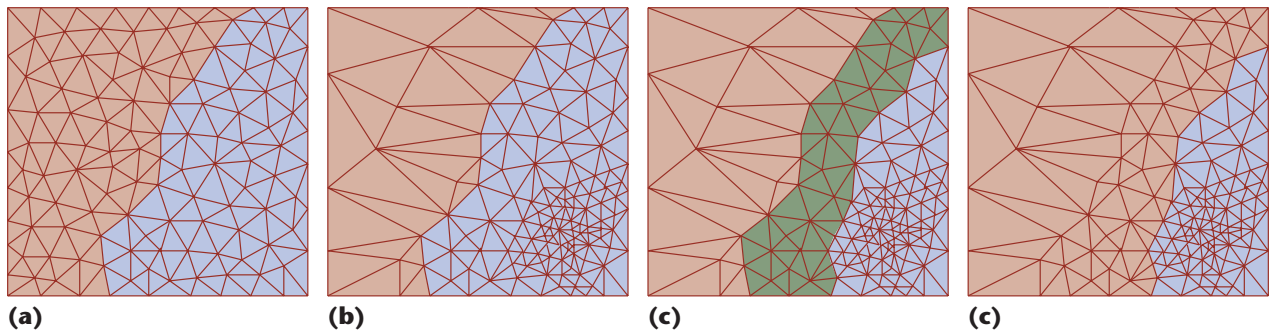


Figure 3. Mesh adaptivity and dynamic load balancing. The (a) initial balanced mesh partitioning is (b) disturbed by mesh adaptivity until (c) mesh migration (d) restores balance.

tools to exploit this knowledge. Our approach relies on a combination of a manual specification and automatic discovery of a computing environment’s characteristics. To start, we evaluate the environment’s performance using both a priori benchmark data and dynamic performance monitoring. We can choose to use such knowledge at any of several common levels of abstraction. Application programmers can make high-level decisions in a resource-aware manner—for example, with their choice of programming languages, a 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 (as opposed to, say, threads or a hybrid approach) is often used because MPI is widely available and highly portable. However, this choice must be made early in the development process, and any change will likely involve significant modification to application software and support libraries. We developed our software with the SPMD model using MPI, so any optimizations must be done within this model.

Compiler and low-level tool developers (such as MPI implementers) can make resource-aware optimizations that benefit a wide range of applications. This might include support for overlapping between computation and communication, reordering computation or communication to take advantage of the target environment’s capabilities,¹¹ or managing communication so that, given a particular interconnect’s buffer sizes and other characteristics, we can concatenate small messages and split larger ones to achieve an optimal message size.¹² Happily, these approaches often don’t require a significant modification to the applications themselves.

Other tool developers, such as those who design

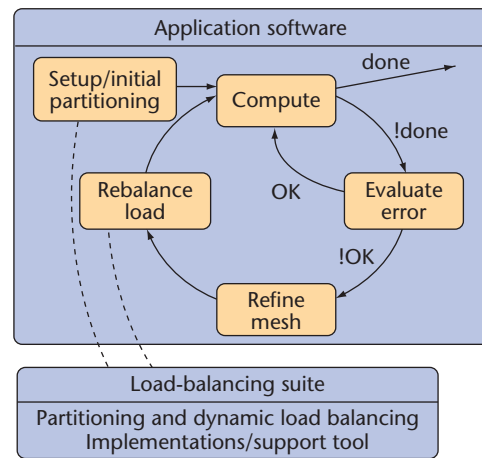


Figure 4. Program flow of a typical parallel adaptive computation. A load-balancing suite such as Zoltan partitions the initial mesh, and the application software periodically evaluates error and determines if adaptive refinement is needed. If so, refinement occurs, and the load-balancing suite is called to rebalance.

and implement numerical libraries or partitioners/dynamic load balancers, can make their software resource-aware and benefit all tool users without requiring significant modification to the existing code base. For our cluster, we can make trade-offs for imbalance versus communication minimization, adjust optimal partition sizes, or partition to avoid communication across the slowest interfaces.^{13–16}

Dynamic Resource Utilization Model

Our desire to support resource-aware load balancing led to the development of the Dynamic Resource Utilization Model (Drum).^{13,15,17} Drum helps determine the computing environment’s characteristics and distills this information into a single “power” value, readily used by the load-balancing procedures to produce appropriate partitions.

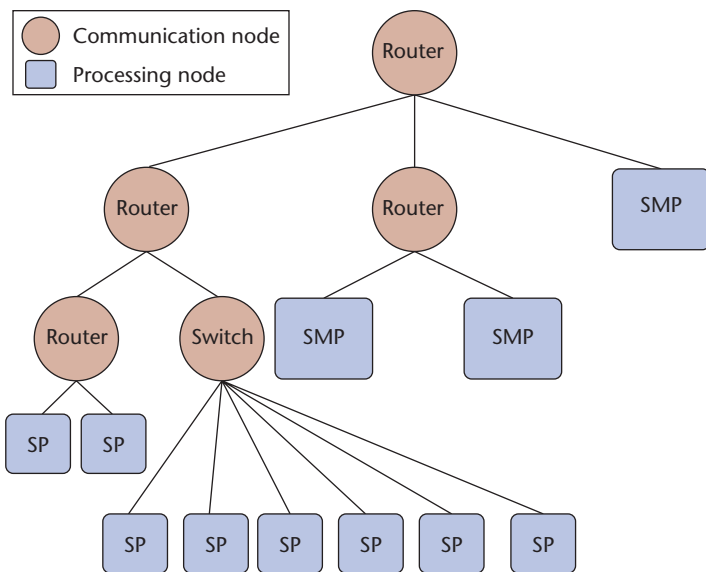


Figure 5. An example computing environment. The tree constructed by Drum to represent a heterogeneous cluster has “leaves” of individual single-processor (SP) nodes or shared-memory multiprocessing (SMP) nodes.

The most straightforward way to account for heterogeneous processor speeds is to assign more of the work to the faster processors because they would otherwise be idle while the slower ones complete their work. We can do this with many existing load-balancing procedures, including Zoltan’s, by requesting that different portions of the work be assigned to the partitions. (Zoltan’s load-balancing procedures take an optional parameter that is an array of partition sizes.) On the Bullpen cluster, for example, we can specify an allocation of 50 percent more work to the fast (450-MHz) processors than to slower (300- or 333-MHz) ones. An application can construct the array of partition sizes if it can determine these relative processor speeds.

Drum maintains the information about relative processor speeds, but processor speed (MHz or GHz) ratings aren’t sufficient for determining relative processing powers. Other factors such as cache, memory, and I/O subsystem performance play important roles in determining how quickly a processor can perform a computation. Instead, we can measure processing power by running benchmarks; by default, we use Linpack (www.netlib.org/linpack), but we can use any program, including the target application itself, to obtain benchmark ratings. Benchmarks are run a priori either manually or from within Drum’s graphical configuration tool; they’re stored in a model of the computing environment that encapsulates in-

formation about hardware resources, their capabilities, and their interconnection topology in a tree structure (see Figure 5). The root of the tree represents the total execution environment, and the children of the root node are high-level divisions of different networks connected to form the total execution environment. Subenvironments 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 tree’s leaves 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 network characteristics. Figure 6 shows the machine model constructed by Drum to represent the Bullpen cluster.

During runtime initialization, Drum constructs its model of the computing environment using information stored in an XML-format configuration file that describes system properties (such as benchmark results and the network’s topology). Drum includes a graphical configuration program in Java called DrumHead that aids in the construction of these configuration files. We can use DrumHead 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. Figure 7 shows an excerpt from an XML configuration file generated by DrumHead for the Bullpen cluster configuration; Figure 8 shows a screenshot of DrumHead.

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 9 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, and 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 benchmarks) and monitored performance. We can evaluate the processing power for each process on a given node based on CPU utilization by the appli-

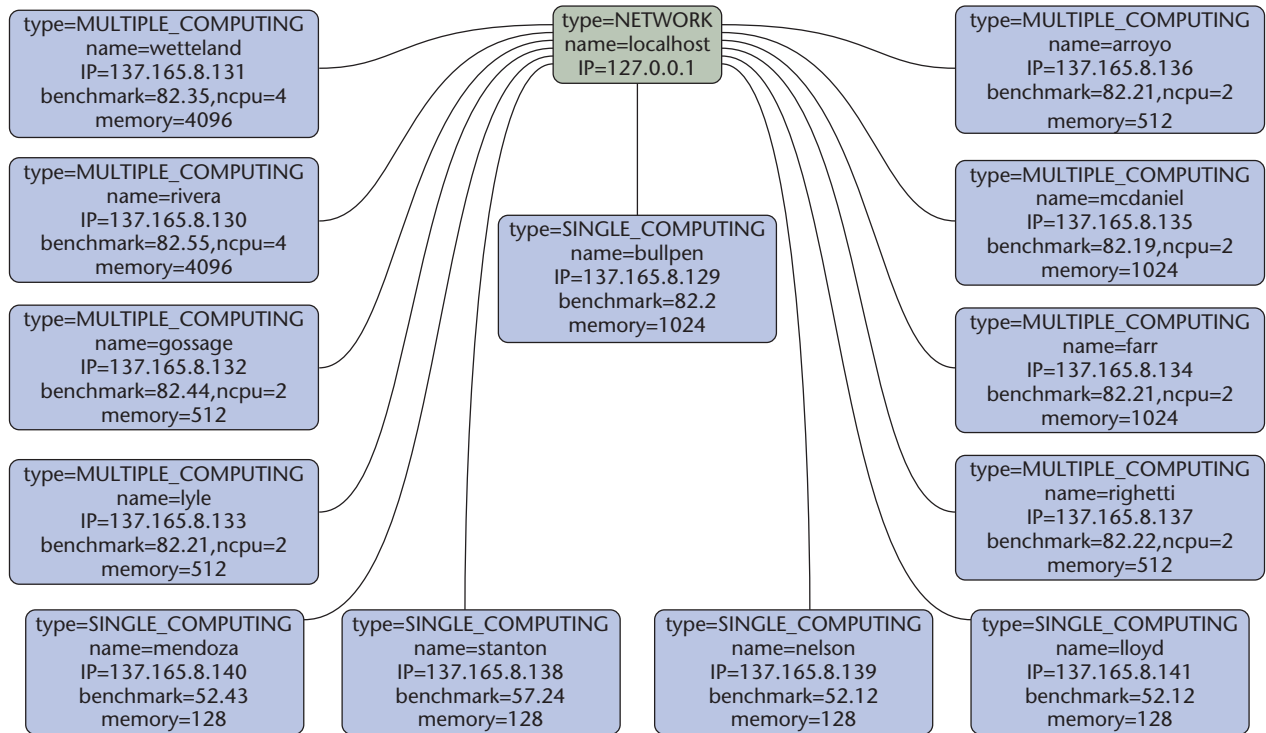


Figure 6. The Bullpen cluster. The tree constructed by Drum to represent the Bullpen cluster includes benchmark ratings that quantify relative processor speeds, the number of processors in each node, and the amount of physical memory in Mbytes.

```

<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>
  
```

Figure 7. An XML configuration file. This excerpt generated by DrumHead for the Bullpen cluster shows the network node, one single-processor node (mendoza), and one multiprocessing node (rivera).

ation process, the fraction of time that CPUs are idle, and the node's static benchmark rating. We assume that the application process can potentially use idle time in the node if it's assigned more work. This formulation lets Drum adjust the computa-

tion appropriately while other processes that aren't part of the computation use the nodes.

We also want to be able to account for heterogeneous, hierarchical, and nondedicated network resources. To do this, agents estimate a node's com-

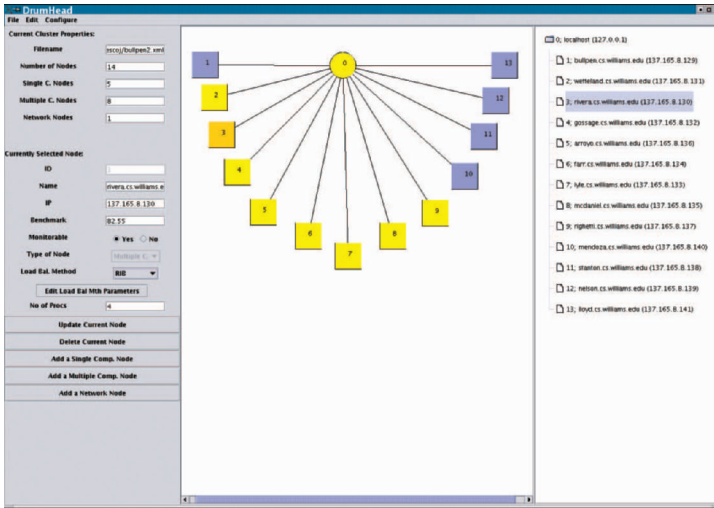


Figure 8. DrumHead in action. In this screenshot, the user edits a description of the Bullpen cluster.

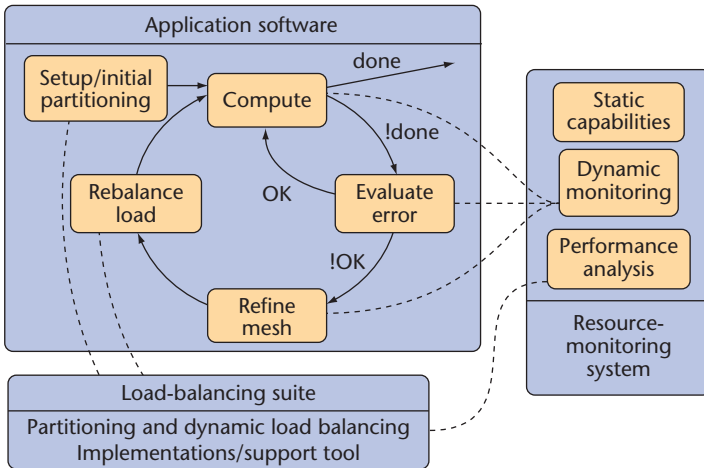


Figure 9. An adaptive computation. In a typical interaction between an adaptive application code and a dynamic load-balancing suite, a resource-monitoring system (such as Drum) monitors application performance and system utilization during the application’s execution. The dynamic load balancer queries the monitoring system to determine optimal partition sizes.

munication power based on the communication traffic at the node—essentially, 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 exploit the fact that it’s currently less busy with communication, so it should be able to perform some extra computation while the other

nodes are in their communication phase. Alternatively, if Drum can get a measure of available bandwidth (such as when the Network Weather Service¹⁸ is monitoring the cluster), it can use this data to determine network powers in place of the communication activity factor.

Drum combines the processing and communication powers as a weighted sum to get the single value for each process to request appropriately sized partitions from the load balancer. Currently, the weights are chosen manually to reflect an estimate of the relative costs of computation and communication for the application in a particular environment, but work is underway to automate this selection. We’re also investigating other ways to combine the powers more effectively.

We’ve used Drum in several studies, many of which are presented elsewhere.¹³ 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. The PHAML studies compute an adaptive solution of a Laplace equation on the unit square. We let the solution progress through 17 adaptive refinement steps, using Zoltan’s Hilbert SFC procedure to rebalance after each refinement. After the last refinement, the mesh has 524,500 nodes, enough to justify the parallel computation.

Figure 10 shows the runtime of a computation using PHAML on various combinations of the Bullpen cluster’s fast and slow processors. In this example, we vary the weight given to communication in Drum’s power computation. Blue bars show running times when Drum isn’t used, and the remaining bars show runtimes when Drum computes partition sizes. Yellow bars indicate times when Drum considers only processing power and ignores communication power, green bars show times when the communication weight is 0.1 (meaning that communication power determines 10 percent of the overall node power), and the brown bars show the communication weight as 0.25. When the computation uses only the fast nodes, times are nearly the same with or without Drum, suggesting very low overhead incurred by Drum’s dynamic monitoring and power computations. On heterogeneous configurations, experiments using Drum’s resource-aware partitions consistently show improvement in execution time compared to those with uniformly sized partitions. In most cases, we get the fastest runtimes when ignoring communication power or when using a weight of only 0.1.

Studies with other applications and on different combinations of cluster nodes have shown similar trends.

We designed Drum to be applicable to dynamic environments in which cluster nodes might not be dedicated or their loads could vary during the course of a Drum-guided computation. To demonstrate this capability, we ran the same PHAML example but with two additional compute-bound processes (which aren't part of the PHAML computation and aren't explicitly monitored by Drum) running on the last node. Figure 11 shows the resulting running times, measured in seconds.

The set of nodes used is indicated along the x -axis. The first set, labeled 2:ppn=4, indicates that two 4-way SMPs are used for a total of eight processors; both extra processes are added to the second SMP, effectively slowing the processors in that node, because the operating system would have six active processes to schedule among the four processors. For the set labeled 2:ppn=4+2 slow, we add two of the slower SP nodes; now the extra processes are both added to one of these slower nodes, resulting in the one already slower node having to schedule three active processes on its single processor. The 2:ppn=4+2:ppn=2 uses two 4-way SMPs and two 2-way SMPs, with both extra processes on the same node, which ends up having four active processes to schedule on its two processors. Finally, the 2:ppn=4+2:ppn=2+2 slow case adds two slower SP nodes and again places both extra processes on the same slow node. In the figure, brown bars show times when we use uniform partitions, and green and yellow bars show times when Drum creates resource-aware partitions with processing power only ($drum=0.0$) or with a 10 percent communication weight ($drum=0.1$), respectively.

The computations with uniform partitions slow down significantly because some processors are overloaded, yet they continue to get an equal share of the work. In particular, when we use the slow nodes, this causes a significant imbalance, with most processors spending time waiting for the overloaded node to complete its part of the computation. In all cases, the Drum-guided partitions lead to shorter execution times, even in cases when the processors are all the same and the only source of heterogeneity is the external load detected at runtime.

Hierarchical Partitioning and Load Balancing

Drum's resource-aware partitioning support doesn't directly deal with hierarchical networks.

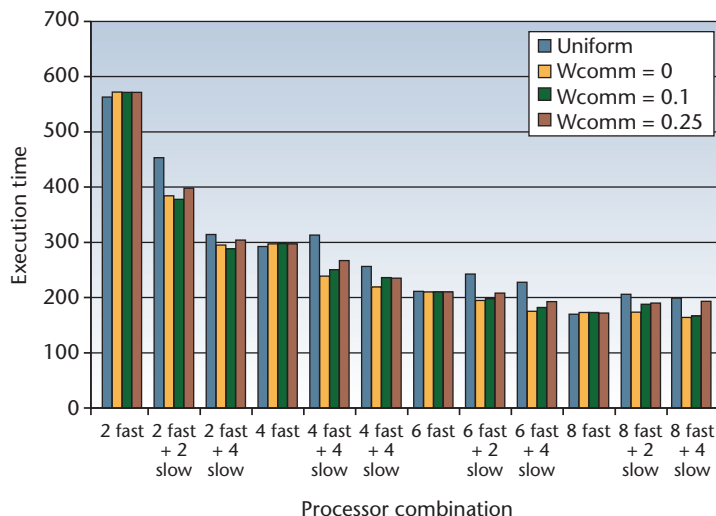


Figure 10. Execution times in seconds. When we use Drum on different combinations of fast and slow processors, with uniform partition sizes and resource-aware partition sizes, we also use various values for W_{comm} , the communication weight in Drum's power formula. In this example, we run just one application process on each node.

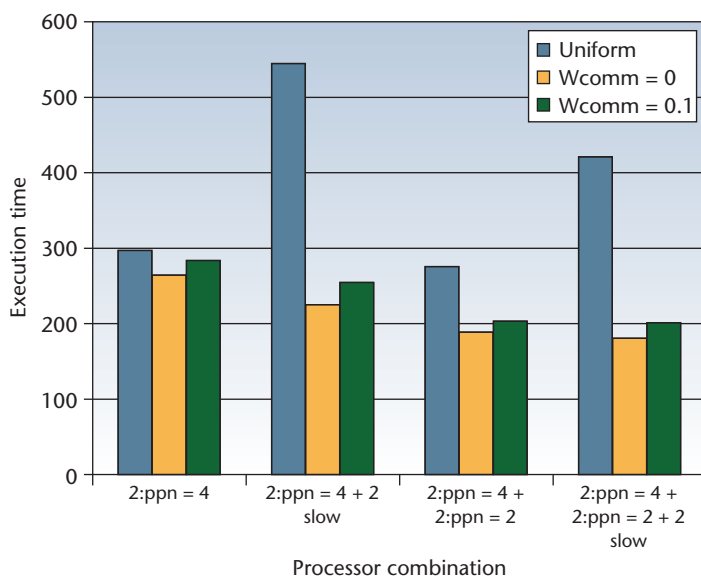


Figure 11. Execution times in seconds. When we use Drum on different combinations of processors with two compute-bound external processes also running on the node with the highest rank, Drum redistributes the work away from the overloaded node, reducing overall execution time. Times are shown for uniform partitions and Drum-guided resource-aware partitions ($W_{comm} = 0$ for processing power only; $W_{comm} = 0.1$ with a communication weight of 0.1 applied). In this example, we run one application process for each CPU in each node.

Each dynamic load-balancing algorithm has characteristics and requirements that make it appro-

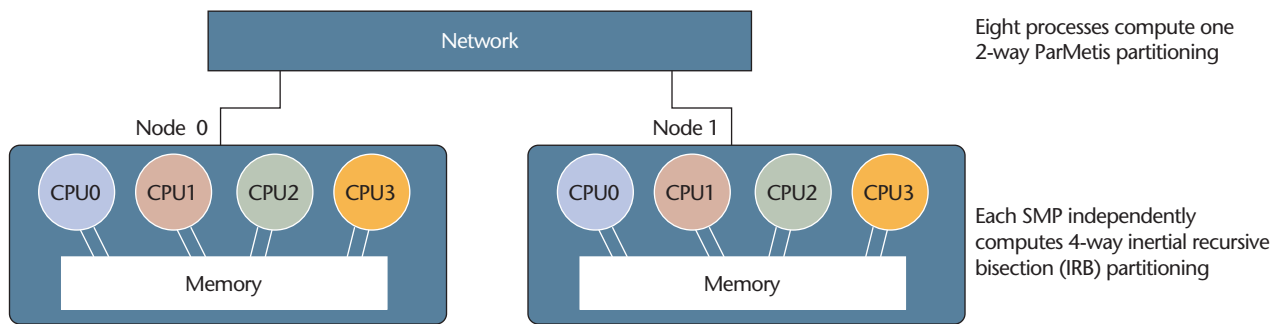


Figure 12. Hierarchical partitioning. A hierarchical balancing algorithm selection for two 4-way shared-memory multiprocessing (SMP) nodes connected by a network uses a graph partitioner to minimize the number of elements on the slow network interface, possibly at the expense of load balance, then a recursive bisection procedure to balance load strictly within each node.

appropriate for certain applications.^{7,19,20} For hierarchical and heterogeneous systems, different choices might be more appropriate in different parts of the parallel environment. Trade-offs in execution time and partition quality (partition boundary sizes, interprocess connectivity, strictness of load balance, and so on),²⁰ for example, could be more important than others in some circumstances. Consider a cluster of SMP nodes connected via Ethernet. We could perform a more costly graph partitioning 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 could partition independently within each node (see Figure 12).

We developed a hierarchical partitioning and dynamic load-balancing system (called HIER)¹⁵ within Zoltan to automate the creation of these hierarchical partitions. Although HIER is implemented entirely within Zoltan, Drum’s machine model can guide it. The DrumHead configuration program includes the ability to specify load-balancing procedures and parameters at each network or SMP node. This information is stored in the computing environment’s configuration file, and Drum can use it to specify appropriate methods and parameters for HIER.

The HIER implementation uses a lightweight intermediate hierarchical balancing structure (IHBS) and a set of callback functions, which lets us compute hierarchical partitions automatically and efficiently with any of the procedures available in Zoltan without modification. HIER is easy to integrate into an application because it’s invoked in the same way as other Zoltan procedures. A hierarchical balancing step begins by building an IHBS by 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 use the ParMetis²¹ and Jostle²² 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 lets HIER migrate lightweight objects internally between levels, not the (larger and more costly) application data.

Our preliminary studies with HIER appear elsewhere.¹⁴ We tested it by using the LOCO solver on the Bullpen cluster. As we mentioned earlier, LOCO helps solve the perforated shock-tube problem, which models the 3D unsteady compressible flow in a cylinder containing a cylindrical vent.¹⁰ The initial mesh contains 69,572 tetrahedral elements, so for our experiments, we stopped the computation after four adaptive steps, when the mesh contains 254,510 elements.

On Bullpen, the hierarchy we wish to account for comes from two- and four-processor nodes connected by fast Ethernet. We compared computation times for the perforated shock-tube computation in two subsets of the cluster: two 4-processor nodes and four 2-processor nodes. Recall that all SMP nodes contain the same-speed processors, so we use HIER without Drum in this case. We tried each combination of traditional and hierarchical procedures and found that although ParMetis multilevel graph partitioning alone often achieves the fastest computation times, there’s a benefit to using hierarchical load balancing when ParMetis is used for internode partitioning or when inertial recursive bisection (IRB) is used within each node. For the environment shown in Figure 12, the computation time following the fourth adaptive step (the computation on the largest mesh) is 571.7 sec-

onds for the hierarchical procedure with ParMetis and IRB, compared with 574.9 seconds for ParMetis alone, 702.7 seconds for Hilbert SFC partitioning alone, 1,508.2 seconds for recursive coordinate bisection alone, and 822.9 seconds for IRB alone. It's higher for other hierarchical combinations of methods. In cases in which the multilevel graph partitioning can find a good decomposition without introducing load imbalance, it provides the fastest time to solution in our studies to this point. When straightforward multilevel graph partitioning introduces a significant load imbalance, hierarchical partitioning can produce a better decomposition. Here, any imbalance is introduced only when partitioning among nodes (with ParMetis) but the load within each node is strictly balanced (by IRB). These are the cases where we've observed the greatest benefit to hierarchical partitioning. We expect that hierarchical balancing will be most beneficial when the extreme hierarchies found in larger clusters and in Grid environments are considered.

We're enhancing the Drum library to run on a variety of computing environments. Because 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're working to provide interfaces to other tools such as the Network Weather Service¹⁸ that might be available on a particular system, but we don't intend to require such packages. We're 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 in the performance analysis. More information about Drum is available at www.cs.williams.edu/drum, and the software itself will be available for download at that location after we more thoroughly test recent extensions.

HIER is part of the Zoltan Toolkit (www.cs.sandia.gov/Zoltan) and is expected to be available in a future public release, hopefully later this year. 

Acknowledgments

The development of Drum and the hierarchical partitioning implementation in Zoltan was supported in part by contract number 15162 with Sandia National Laboratories, a multiprogram laboratory operation by Sandia Corporation, a Lockheed Martin Company, for

the US Department of Energy under contract DE-AC04-94AL85000. The authors 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 contributed to Drum's design. Williams undergraduates Laura Effinger-Dean and Arjun Sharma contributed to Drum as part of the Williams College Summer Science Research program.

References

1. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1999.
2. J.E. Flaherty et al., "Software for the Parallel Adaptive Solution of Conservation Laws by Discontinuous Galerkin Methods," *Discontinuous Galerkin Methods Theory, Computation and Applications*, B. Cockburn, G. Karniadakis, and S.-W. Shu, eds., Springer, pp. 113–124.
3. W.F. Mitchell, "The Design of a Parallel Adaptive Multilevel Code in Fortran 90," *Proc. Int'l Conf. Computational Science*, LNCS 2331, Springer, 2002, pp. 672–680.
4. J.-F. Remacle, J. Flaherty, and M. Shephard, "An Adaptive Discontinuous Galerkin Technique with an Orthogonal Basis Applied to Compressible Flow Problems," *SIAM Rev.*, vol. 45, no. 1, 2003, pp. 53–72.
5. M.S. Shephard et al., "Parallel Automated Adaptive Procedures for Unstructured Meshes," *Parallel Computation in CFD*, no. R-807, 1995, pp. 6.1–6.49.
6. S. Adjerid et al., "High-Order Adaptive Methods for Parabolic Systems," *Physica-D*, vol. 60, 1992, pp. 94–111.
7. J.D. Teresco, K.D. Devine, and J.E. Flaherty, "Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations," *Numerical Solution of Partial Differential Equations on Parallel Computers*, A.M. Bruaset, P. Bjørstad, and A. Tveito, eds., Springer-Verlag, 2005.
8. K. Devine et al., "Zoltan Data Management Services for Parallel Dynamic Applications," *Computing in Science & Eng.*, vol. 4, no. 2, 2002, pp. 90–97.
9. R. Biswas, K.D. Devine, and J.E. Flaherty, "Parallel, Adaptive Finite Element Methods for Conservation Laws," *Applied Numerical Mathematics*, vol. 14, Apr. 1994, pp. 255–283.
10. J.E. Flaherty et al., "Distributed Octree Data Structures and Local Refinement Method for the Parallel Solution of Three-Dimensional Conservation Laws," *IMA Volumes in Mathematics and Its Applications*, M. Bern, J. Flaherty, and M. Luskin, eds., Springer, 1999, pp. 113–134.
11. N.T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *J. Parallel and Distributed Computing*, vol. 63, no. 5, 2003, pp. 551–563.
12. R.M. Loy, *AUTOPACK version 1.2*, tech. memorandum ANL/MCS-TM-241, Mathematics and Computer Science Division, Argonne National Lab., 2000.
13. J. Faik et al., *A Model for Resource-Aware Load Balancing on Heterogeneous Clusters*, tech. report CS-04-03, Williams College, Dept. of Computer Science, 2004.
14. S. Sinha and M. Parashar, "Adaptive System Partitioning of AMR Applications on Heterogeneous Clusters," *Cluster Computing*, vol. 5, no. 4, 2002, pp. 343–352.
15. J.D. Teresco, J. Faik, and J.E. Flaherty, *Hierarchical Partitioning and Dynamic Load Balancing for Scientific Computation*, tech. report CS-04-04, Williams College, Dept. of Computer Science, 2004.
16. C. Walshaw and M. Cross, "Multilevel Mesh Partitioning for Heterogeneous Communication Networks," *Future Generation Com-*

putational Systems, vol. 17, no. 5, 2001, pp. 601–623.

17. K.D. Devine et al., "New Challenges in Dynamic Load Balancing," *Applied Numerical Mathematics*, vol. 52, nos. 2 and 3, 2005, pp. 133–152.
18. R. Wolski, N.T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computational Systems*, vol. 15, no. 6, 1999, pp. 757–768.
19. E. Boman et al., *Parallel Repartitioning for Optimal Solver Performance*, tech. report SAND2004-0365, Sandia Nat'l Labs, Feb. 2004.
20. J.D. Teresco and L.P. Ungar, *A Comparison of Zoltan Dynamic Load Balancers for Adaptive Computation*, tech. report CS-03-02, Williams College, Dept. of Computer Science, 2003.
21. G. Karypis and V. Kumar, "Parallel Multilevel k -Way Partitioning Scheme for Irregular Graphs," *SIAM Rev.*, vol. 41, no. 2, 1999, pp. 278–300.
22. C. Walshaw and M. Cross, "Parallel Optimisation Algorithms for Multilevel Mesh Partitioning," *Parallel Computing*, vol. 26, no. 12, 2000, pp. 1635–1660.

James D. Teresco is an assistant professor in the Department of Computer Science at Williams College. His technical interests include parallel scientific computation and dynamic load balancing for adaptive computation in heterogeneous, hierarchical, and Grid computational environments. Teresco has a PhD in

computer science from Rensselaer Polytechnic Institute. He is a member of the ACM, SIAM, and the IEEE Computer Society. Contact him at terescoj@cs.williams.edu.

Jamal Faik is a PhD candidate in the Department of Computer Science at Rensselaer Polytechnic Institute. His technical interests include parallel and distributed processing, load balancing, interconnection networks, and workload management. Faik has an MS in computer science from Al-Akhawayn University. He is a member of SIAM and the IEEE. Contact him at faikj@cs.rpi.edu.

Joseph E. Flaherty is Dean of Science and the Amos Eaton Professor of Computer Science at Rensselaer Polytechnic Institute. His technical interests include numerical analysis and the solution of ordinary and partial differential equations with special emphasis on singularly perturbed systems and problems in solid and fluid mechanics. Flaherty has a PhD in applied mechanics from the Polytechnic Institute of Brooklyn. He is a member of the ACM, the AMS, the IEEE Computer Society, IMACS, the Mathematical Association of America, and SIAM. Contact him at flaherje@cs.rpi.edu.

PURPOSE The IEEE Computer Society is the world's largest association of computing professionals, and is the leading provider of technical information in the field.

MEMBERSHIP Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEB SITE The IEEE Computer Society's Web site, at www.computer.org, offers information and samples from the society's publications and conferences, as well as a broad range of information about technical committees, standards, student activities, and more.

BOARD OF GOVERNORS

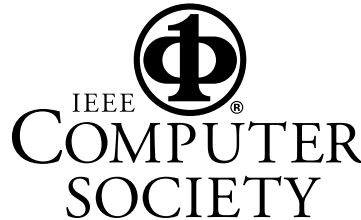
Term Expiring 2005: Oscar N. Garcia, Mark A. Grant, Michel Israel, Robit Kapur, Stephen B. Seidman, Kathleen M. Swigger, Makoto Takizawa

Term Expiring 2006: Mark Christensen, Alan Clements, Annie Combelles, Ann Q. Gates, James D. Isaak, Susan A. Mengel, Bill N. Schilit
Term Expiring 2007: Jean M. Bacon, George V. Cybenko, Richard A. Kemmerer, Susan K. (Kathy) Land, Itaru Mimura, Brian M. O'Connell, Christina M. Schober

Next Board Meeting: 11 Mar. 2005, Portland, OR

IEEE OFFICERS

President: W. CLEON ANDERSON
President-Elect: MICHAEL R. LIGHTNER
Past President: ARTHUR W. WINSTON
Executive Director: TBD
Secretary: MOHAMED EL-HAWARY
Treasurer: JOSEPH V. LILLIE
VP, Educational Activities: MOSHE KAM
VP, Pub. Services & Products: LEAH H. JAMIESON
VP, Regional Activities: MARC T. APTER
VP, Standards Association: JAMES T. CARLO
VP, Technical Activities: RALPH W. WYNDRUM JR.
IEEE Division V Director: GENE F. HOFFNAGLE
IEEE Division VIII Director: STEPHEN L. DIAMOND
President, IEEE-USA: GERARD A. ALPHONSE



COMPUTER SOCIETY OFFICES

Headquarters Office

1730 Massachusetts Ave. NW
Washington, DC 20036-1992
Phone: +1 202 371 0101
Fax: +1 202 728 9614
E-mail: hq.ofc@computer.org

Publications Office

10662 Los Vaqueros Cir., PO Box 3014
Los Alamitos, CA 90720-1314
Phone: +1 714 821 8380
E-mail: help@computer.org
Membership and Publication Orders:
Phone: +1 800 272 6657
Fax: +1 714 821 4641
E-mail: help@computer.org

Asia/Pacific Office

Watanabe Building
1-4-2 Minami-Aoyama, Minato-ku
Tokyo 107-0062, Japan
Phone: +81 3 3408 3118
Fax: +81 3 3408 3553
E-mail: tokyo.ofc@computer.org



EXECUTIVE COMMITTEE

President:

GERALD L. ENGEL*

Computer Science & Engineering
Univ. of Connecticut, Stamford
1 University Place
Stamford, CT 06901-2315
Phone: +1 203 251 8431
Fax: +1 203 251 8592
g.engel@computer.org

President-Elect: DEBORAH M. COOPER*

Past President: CARL K. CHANG*

VP, Educational Activities: MURALI VARANASI†

VP, Electronic Products and Services:

JAMES W. MOORE (2ND VP)*

VP, Conferences and Tutorials:

YERVANT ZORIAN†

VP, Chapters Activities:

CHRISTINA M. SCHOBER*

VP, Publications: MICHAEL R. WILLIAMS (1ST VP)*

VP, Standards Activities: SUSAN K. (KATHY) LAND*

VP, Technical Activities: STEPHANIE M. WHITE†

Secretary: STEPHEN B. SEIDMAN*

Treasurer: RANGACHAR KASTURI†

2004–2005 IEEE Division V Director:

GENE F. HOFFNAGLE†

2005–2006 IEEE Division VIII Director:

STEPHEN L. DIAMOND†

2005 IEEE Division V Director-Elect:

OSCAR N. GARCIA*

Computer Editor in Chief: DORIS L. CARVERT

Executive Director: DAVID W. HENNAGE†

* voting member of the Board of Governors

† nonvoting member of the Board of Governors

EXECUTIVE STAFF

Executive Director: DAVID W. HENNAGE
Assoc. Executive Director: ANNE MARIE KELLY
Publisher: ANGELA BURGESS
Assistant Publisher: DICK PRICE
Director, Administration: VIOLET S. DOAN
Director, Information Technology & Services: ROBERT CARE
Director, Business & Product Development: PETER TURNER