Parallel Tools and Environments: A Survey

William D. Gropp* and Andrew Lumsdaine**

*Mathematics and Computer Science Division, Argonne National Laboratory
**Computer Science Department, Indiana University

This work was supported in part by the U.S. Department of Energy under Contract W-31-109-Eng-38 and by a grant from the Lilly Endowment.

Writing parallel programs is difficult. Besides the inherent difficulties associated with writing any kind of software, parallel programs have additional complexities due to data management, process management, and process synchronization. Further, even the basic activities involved in writing and using parallel programs are often more difficult than those same activities on a conventional, uniprocessor computer. Some of this difficulty is due to the traditional dearth of tools for parallel programming and parallel computing. Early users of parallel systems had to write their own tools from scratch, sometimes even including basic system software. Some features, such as robust, fast file systems, were simply unavailable.

Today the situation is quite different. While parallel computing environments are still not as easy to use or as robust as workstation environments, great strides have been made in improving parallel computing for end users. These improved environments have been driven by the rapid expansion in the number of parallel computers and the number of people using them (enabled in large part by the exploitation of commodity components, e.g., by the Beowulf project [8]). Similarly, improved parallel programming has been enabled by the development of a standard programming model and applications programmer interface for developing parallel scientific applications. The Message Passing Interface (MPI) standard [3, 5, 6] allows the development of both parallel programs and parallel libraries. By supporting software libraries, MPI allows programmers to build applications in terms of the natural operations for their application, such as solving a system of nonlinear equations, rather than low-level, specialized parallel programming commands. As a result, an active community of builders and users of parallel tools has arisen.

This chapter surveys the categories of tools useful for parallel computing and briefly describes some particular tools in each category. Section 0.1 describes software and tools that can be used to set up and manage a parallel computing cluster. Section 0.2 focuses on tools for computational science, including numerical libraries, software environments, and complete applications.

Many tools and environments are already available; no single chapter (or even book!) could list them all. We cover some of the most widely used open-source and research tools, with particular emphasis on tools that were featured at the 2004 SIAM Conference on Parallel Processing for Scientific Computing. (The presence or absence of any tool in our survey should not be considered as an endorsement or otherwise. To be fair, we have not specifically named commercial products.) New tools are being developed, and existing tools continue to evolve; we encourage users to ask others what tools they use and to search the Web for new developments.

0.1 Software and Tools for Building and Running Clusters

In the early days of cluster computing, building a cluster meant ordering boxes of computers, unpacking them, loading software onto each one, and then writing custom code to manage the collection of machines as a parallel computer. Today one can order a cluster, complete with all of the software needed to operate it, from a vendor who will install it, test it, and service it. Understanding the different

system tools available for clusters is still necessary, however, and those who choose to build their own cluster (still often the best choice for small clusters) will need to acquire the tools that operate the cluster and allow users to build and run programs. This section can only touch on these tools; a cluster reference such as [4, 7] will provide a more thorough discussion.

Setting up and running a cluster involves three aspects:

Node Configuration: How will each individual node be configured (e.g,. what operating system and what software will go on each node)?

Cluster Configuration: How will the individual nodes be connected and configured together to make up a parallel computing resource?

Cluster Management: How will the individual nodes and the cluster configuration be managed over time (e.g., how will new software be installed and how will jobs be submitted)?

0.1.1 Node Configuration

Clusters comprise a collection of individual computers. An important aspect of cluster setup is therefore the configuration of each of those nodes in the cluster.

Operating System. The most fundamental choice in setting up individual cluster nodes is the operating system. Linux is a common choice in many cases. Linux distributions are available from a number of providers, and the cost of acquisition is typically quite low (often free). Linux has been ported to the various microprocessors that are used in computational clusters, including new 64-bit architectures. Because of the popularity and ubiquity of Linux, a large selection of software packages is available for it (in source and packaged binary form), including most of the tools for parallel computing that are mentioned in this chapter. There are a huge number of Linux distributions (both well known and obscure). Some of the more widely used distributions for cluster computing are RedHat, Debian, SuSE, Fedora, Mandrake, Gentoo, and Yellowdog. In choosing a distribution, one should consider issues such as ease of installation, availability of support, availability of packaged software, and compatibility with specialized hardware, such as high-speed interconnects.

Compilers and Debuggers. In order to develop programs, compilers are needed. The freely available GNU compilers support all of the popular languages in scientific computing (including Fortran 95 with the recent release of g95). Higher performance and additional features, such as support for the OpenMP standard [1], are available from a range of commercial compiler vendors. Program development tools, including commercial parallel debuggers, are also available.

Sequential Libraries. High levels of parallel performance depend on high levels of sequential performance. Many hardware vendors provide highly tuned versions of

sequential libraries such as the BLAS. Many of the BLAS routines are also available as part of the ATLAS project, which uses automatic tuning techniques to provide performance equal to vendor-tuned BLAS in many cases.

0.1.2 Cluster Configuration

Beyond the configuration of each individual node, an important aspect of cluster setup is the infrastructure that allows the collection of nodes to work together effectively as a cluster.

Network Architecture. The nodes in a computational cluster will almost certainly be equipped out of the box with 100 Mbit/s or even (as is becoming more and more common) 1 Gbit/s Ethernet. This network will be used for cluster administration and most cluster services (such as basic NFS file sharing, directory services, and remote login). One important decision is whether to make this a public network or a private network (having public or private IP addresses, respectively). With private addressing, the cluster accesses the public Internet via a gateway machine or router, and this setup can provide certain security and administrative benefits.

For some classes of computational problems, Ethernet may offer sufficient performance. In many cases, however, one will want to add a second high-performance network such as Myrinet, Infiniband, or Quadrics to explicitly support communication for parallel computing. Also desirable will be the availability of an MPI library that is ported (and tuned) for the selected interconnect. Hardware vendors providing high-performance interconnects may also provide a corresponding MPI implementation. Several of the open-source implementations support these interconnects as well.

Basic Services. The nodes in a cluster will need basic services such as shared file systems, directory services, gateway services, and cluster administration. For reasons of symmetry (and load balancing), it is best not to use for this purpose nodes that will also be used to run compute jobs. Rather, some number of "head node" servers should be apportioned to handle such services. The exact ratio of service nodes to compute nodes will depend on individual needs, but typically one could expect to support 8 to 32 compute nodes with a single service node.

File System. Any computer needs a file system on which to store data. Unix-base (including Linux-based) clusters will usually provide the Network File System (NFS). This is the system commonly used on Unix desktop workstations and Unix servers and is often a good choice as the file system for source files. NFS may be a poor choice for use by parallel applications, however, because it was not designed to support multiple processes accessing (particularly writing to) the same file. For I/O to and from parallel applications, a parallel file system should be used. There are a number to choose from, including freely available (such as PVFS2) and commercially supported, such as Lustre (`www.clusterfs.com`), GPFS (`www-1.ibm.`

com/servers/eserver/clusters/software/gpfs.html), Panasas (www.panasas.com), and GFS (www.redhat.com/software/rha/gfs).

Middleware. Parallel programs require support for a parallel programming model. While some parallel languages are available with compilers for clusters, most users in computational science use the Message Passing Interface standard. This standard describes a library of routines that allow processes running on different nodes to communicate and coordinate. There are a number of implementations of the MPI standard; any application that uses MPI may be compiled with the mpi header files provided by the implementation and linked with the MPI library provided by the implementation. Thus, one can use any MPI implementation with any MPI program. Popular MPI implementations are MPICH2 (www.mcs.anl.gov/mpi/mpich2/) and LAM/MPI (www.lam-mpi.org); another implementation, OpenMPI (www.open-mpi.org), should be available when this book is released. Commercial implementations of MPI are also available.

0.1.3 Cluster Management

Once a cluster is up and running, various on-going administrative tasks and policies must be attended to.

Scheduling. A cluster will typically be a shared resource with multiple users. The computational resources for a parallel job should be considered together and allocated as a single resource. Inadvertently sharing even a single node between parallel jobs can cause severe load-balancing problems for those jobs. Manually managing the use of multiple compute nodes (which in a cluster are also essentially workstations) is extraordinarily difficult when even moderate numbers of users and nodes are involved. The most effective way to allocate groups of nodes for exclusive use is via a batch scheduler. Popular batch schedulers include OpenPBS (www.openpbs.org), Torque (http://www.clusterresources.com/pages/products/torque-resource-manager.php), and SLURM (www.llnl.gov/linux/slurm).

Integrated Node and Cluster Configuration Management. In a cluster, each node (typically a single PC) needs to have an operating system loaded onto it; the node must also be configured to use the interconnection network that allows the nodes to communicate with each other.

Setting up the individual nodes one by one in a cluster of any size can be a tedious and error-prone task (as would be upgrading a cluster in such a fashion). Fortunately, tools exist to automate this process as well as to automate the process of transforming a collection of nodes into an integrated cluster. Three popular tools are

ROCKS (rocks.npaci.edu/Rocks),

OSCAR (oscar.openclustergroup.org), and

Cplant www.cs.sandia.gov/cplant.

These packages represent different approaches to the management of cluster system software. ROCKS is a *description-based* method. This approach uses tools built into the operating system to describe and assemble the software needed for each node. The advantage of this approach is that it works well with nearly any hardware; but because it is built atop installation tools in an operating system, it limits the choice of operating system. In the case of ROCKS, only certain Linux distributions are allowed.

OSCAR is a *disk image-based* method. This approach uses a custom installation program to build a disk image that can then be copied to all of the nodes in the cluster. This approach allows greater flexibility in the choice of operating system (some, such as the Chiba City system, allow both Windows and Linux to be used), but may limit the choice of hardware (such as disks or new network interfaces).

Cplant is provided as source code and contains a complete environment targeted at scalability, including runtime utilities and debugging support.

Regardless of which toolkit chosen, when a cluster management system finishes its initial setup process, the computational cluster will be ready to use. And, since these toolkits are aimed at supporting high-performance science computing, the cluster as initially configured will include the other system software components discussed above.

One of the reasons for the success of clusters has been the ability to take advantage of commodity components. This is most obvious in hardware, but it is also true in software. There are multiple commodity operating systems, compilers, file systems, MPI implementations, and parallel programming environments. However, not all of these tools work together seamlessly—particularly the tools that manage the cluster. The Scalable Systems Software project (`www.scidac.org/ScalableSystems`), part of the DOE SciDAC (Scientific Discovery through Advanced Computing) program, seeks to develop standards for interfaces between system software components to enable the development of new functionalities.

0.2 Tools for Computational Science

Many tools are available for all conducting all phases of computational science on clusters. Many of the chapters in this book discuss particular tools in depth; in this chapter, we briefly summarize some of the available tools. Tools chosen for this section were described at the 2004 SIAM Parallel Processing conference in one or more talks. Not all tools are included, and many other tools are available for clusters. This section is intended to give a sampling that illustrates the breadth of tools available for parallel computing. The section is organized by category, starting with software libraries for solving linear and nonlinear equations.

0.2.1 Solvers for Linear and Nonlinear Equations

The development of solvers for linear and nonlinear systems of equations for parallel computers is as old as parallel computers themselves. As a result, many mature and efficient libraries are available for these problems. Chapters **??** and **??** discuss the state of the art in these areas.

Following are some of the tools featured at the 2004 SIAM meeting on parallel processing, along with a URL that gives access to more data and to the software.

- pARMS (`http://www-users.cs.umn.edu/~saad/software/pARMS`) provides preconditioned Krylov solvers for linear systems, using recursive multilevel ILU preconditioning.
- Prometheus (`www.cs.berkeley.edu/~madams/prometheus`) is a scalable unstructured finite element solver employing multigrid.
- SuperLU (`crd.lbl.gov/~xiaoye/SuperLU`) is a sparse direct linear solver, with versions for nonparallel machines, shared memory and distributed memory.
- PETSc (`www.mcs.anl.gov/petsc`) is a package for solving linear and nonlinear systems of equations, emphasizing support for equations from PDE discretizations.
- hypre (`www.llnl.gov/CASC/linear_solvers`) provides parallel preconditioners featuring multigrid.
- Petra is part of Trilinos (`software.sandia.gov/trilinos`) and provides the basic linear algebra support such as parallel sparse matrix operations.
- PLAPACK (`www.cs.utexas.edu/users/plapack`) is a parallel dense linear algebra package.
- ScaLAPACK (`www.netlib.org/scalapack/scalapack_home.html`) is another parallel dense linear algebra package.

In evaluating libraries, beyond the usual issues of correctness and accuracy, one should consider completeness and interoperability. While these terms are not precise, they describe important qualitative properties of a library. A library is *complete* if it provides all of the routines needed to create and use the data structures that it needs. For example, a sparse matrix library is not complete if there are no routines to help assemble the sparse matrix data structures. Libraries that are not complete in this sense require more effort to use.

Interoperability is the property that allows an application to use multiple libraries. Some libraries and tools may assume that no other tools are used in the application, an assumption that can limit the applications for which the library is suitable.

0.2.2 Parallel Programming and Languages

Parallel programming is considered by many to be too hard. Without question, it is more difficult than programming a single processor; moreover, writing bad parallel programs seems to be easier than writing bad regular programs. Many efforts have been undertaken to simplify parallel programming. One is to develop a new, general-purpose parallel language—often, an extension of an existing language.

For example, the partitioned global address space (PGAS) languages provide a unified view of the entire parallel computer, rather than using MPI to connect otherwise separate programs on each node. Important examples of these languages are Unified Parallel C (UPC, `upc.gwu.edu`) and CoArray Fortran (CAF)[1]. PGAS languages have the concept of local and remote memory and hence promise efficient implementations on clusters. Users have reported some positive experience with these languages, but they are not yet widely available and do not always offer the highest performance.

Another approach has been to build tools optimized for a particular domain or class of algorithms. For example, both the language Cilk (`supertech.lcs.mit.edu/cilk`) and the library MasterWorker (`www.cs.wisc.edu/condor/mw`) provide good support for task-parallelism. Charm++ (`charm.cs.uiuc.edu/research/charm`) also provides a programming model that supports parallelism through the creation and management of large numbers of "virtual" tasks.

In the future, we expect to see more such tools. Domain-specific languages are simply programming languages tuned to a particular domain and usually a particular data structure within that domain (data-structure-specific language is often a more accurate description). These languages can take advantage of knowledge about the domain to raise the level of abstraction and hide many of the details of parallel programming from the user. One example is parallel-R (`www.aspect-sdm.org/Parallel-R`), a parallel version of the statistical language R.

The DARPA High Productivity Computer Systems project is an important project to watch (`www.highproductivity.org`). This project seeks to develop a combination of hardware and software (including new computer languages) to significantly increase programmer productivity. Even if these specific efforts never become commercial systems, the ideas developed will undoubtedly stimulate further work in computer architecture and programming languages.

0.2.3 Performance Evaluation

Two major factors motivate the use of parallel computing: the need for more computing performance and the need for more memory in which to perform the calculations. Thus, tools to identify and repair performance bugs are a critical part of any parallel computing environment.

For most applications, most of the gap between the performance of an application and the peak performance is due not to the parallelism but to the capabilities of the individual nodes. Thus, tuning the single-node performance is the first and often most important step.

An important step toward making high-quality performance evaluation possible has been the development of a set of routines that provide portable access to the performance counters maintained by most modern CPUs. The PAPI library (`http://icl.cs.utk.edu/papi`) is available for many operating systems and processors and provides such a portable interface.

Tools that work closely with the compiler or the source code can provide

[1] Earlier known as F--

more context for understanding the reasons for the measured performance and suggest ways to improve performance. Tools such as those in the HPCToolkit (`www.hipersoft.rice.edu/hpctoolkit`) provide detailed information about the behavior of an application.

Once the single-processor or single-node performance of an application has been evaluated and tuned, one should look at the parallel performance. Most tools for understanding the parallel performance of applications fall into two categories: tools that create a log file of every parallel computing event, such as sending or receiving a message, along with tools to analyze and display the contents of this long file, and tools that create a summary of the parallel computing events, for example, by counting the amount of data sent between two processes.

The Tuning and Analysis Utilities (TAU, `www.cs.uoregon.edu/research/paracomp/tau/tautools`) is another package that provides tools for instrumenting and analyzing applications.

Examples of logfile-based tools are Paraver (Chapter **??**), SvPablo (`www.renci.unc.edu/Project/SVPablo/SvPabloOverview.htm`), Jumpshot (`www.mcs.anl.gov/perfvis/software/viewers`), and Vampir (now part of the Intel Cluster Tools, `www.intel.com/software/products/cluster`).

Summary tools include FPMPI (`www.mcs.anl.gov/fpmpi/WWW`) and mpiP (`www.llnl.gov/CASC/mpip`).

The MPI standard provides support for the development of customized logging tools through what is called the "profiling interface." This feature of MPI provides a way to intercept any MPI call, perform any user-specified action, and then invoke the original MPI operation. In fact, many of the parallel performance tools use this interface. Users that need special kinds of information should consider using the profiling interface (a required part of all MPI implementations).

0.2.4 Problem Solving Environments

Parallel programming is not necessary in some applications areas because of the availability of problem solving environments (PSEs). These provide substantial or complete support for computations, turning a parallel computer into just another source of computer cycles.

Among the PSEs featured at the SIAM 2004 parallel processing meeting were the following:

- BioPSE (`www.sci.utah.edu/ncrr/software`) is a problem solving environment for biomedical problems and includes, in addition to solver and visualization tools, support for computational steering.

- SCIRun (`software.sci.utah.edu/scirun.html`) is an environment for building problem solving environments (BioPSE is built on top of SCIRun).

- Cactus (`www.cactuscode.org`) is a problem solving environment supporting the collaborative development of parallel applications in science and engineering, and is well known for work in CFD and astrophysics.

- NWChem (`www.emsl.pnl.gov/docs/nwchem/nwchem.html`) is a package for computational chemistry.

Commercial applications include fluid dynamics, structural mechanics, and visualization. More applications can be expected because of the rapidly increasing number of clusters provides a market for them.

0.2.5 Other Tools

Many other categories of tools exist in addition to those described above. This book, for example, includes chapters on mesh generation (Chapter **??**), component architectures for interoperable software components (Chapter **??**), and fault tolerance (Chapter **??**). Other sources of information about parallel tools and environments include books such as [2] and [4] as well as on-line sources such as the Beowulf list (`www.beowulf.org`). And, of course, using Web search engines will help one discover new tools as they continue to be developed and deployed.

0.3 Conclusion

For many scientists and engineers, parallel computing has been made practical by the combination of commodity hardware and commodity software, aided by the development of standards—particularly those for parallel computing such as the MPI standard—and a healthy competition between groups developing software and hardware to these standards.

We close this chapter by summarizing some general recommendations for users and developers of computational science tools for parallel computers.

For users, first and foremost: Don't write code if you don't need to!

1. If you are setting up your own cluster, use one of the setup tools.
2. Use a problem solving environment if possible.
3. Use one or more parallel libraries to raise the level of abstraction, essentially turning MPI into a higher-level, domain-specific language for your application area.
4. Use an appropriate programming model and performance tools.

For tool developers, perhaps the most important recommendation is to ensure that your tool can interoperate with other tools. Tools should also be complete in terms of providing not just the core algorithm but the routines or tools that get a user from their problem description to your tool.

Index

Bibliography

[1] L. Dagum and R. Menon, *Openmp: An industry-standard api for shared-memory programming*, IEEE Comput. Sci. Eng., 5 (1998), pp. 46–55.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, eds., *Sourcebook of Parallel Computing*, Morgan Kaufmann, 2003.

[3] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI—The Complete Reference: Volume 2, The MPI-2 Extensions*, MIT Press, Cambridge, MA, 1998.

[4] W. Gropp, E. Lusk, and T. Sterling, eds., *Beowulf Cluster Computing with Linux*, MIT Press, 2nd ed., 2003.

[5] Message Passing Interface Forum, *MPI: A Message-Passing Interface standard*, International Journal of Supercomputer Applications, 8 (1994), pp. 165–414.

[6] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI—The Complete Reference: Volume 1, The MPI Core,* 2nd edition, MIT Press, Cambridge, MA, 1998.

[7] T. Sterling, ed., *Beowulf Cluster Computing with Windows*, MIT Press, 2002.

[8] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer, *BEOWULF : A parallel workstation for scientific computation*, in International Conference on Parallel Processing, Vol.1: Architecture, Boca Raton, USA, Aug. 1995, CRC Press, pp. 11–14.