Crossroads
The ACM Student Magazine

Parallel Computing With Linux

by *Forrest Hoffman* and *William Hargrove*

Linux is just now making a significant impact on the computing industry, but it has been a powerful tool for computer scientists and computational scientists for a number of years. Aside from the obvious benefits of working with freely-available, reliable, and efficient open source operating system [1], the advent of Beowulf-style cluster computing--pioneered by Donald Becker, Thomas Sterling, et al. [2] at NASA's Goddard Space Flight Center--extends the utility of Linux to the realm of high performance parallel computing. Today, these commodity PC-based clusters are cropping up in federal research laboratories, industrial R&D centers, universities, and even small colleges [3, 4]. If a computational problem can be solved in a loosely-coupled distributed memory environment, a Beowulf cluster--or Pile of PCs (POP)--may be the answer; and it "weighs in" at a price point traditional parallel computer manufacturers cannot touch.

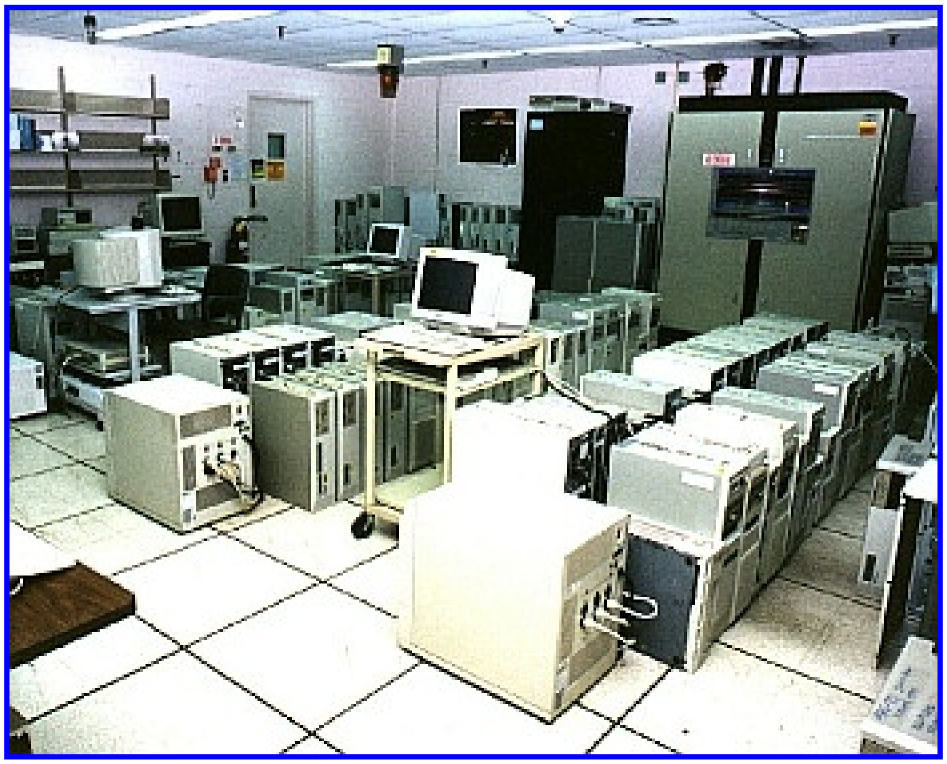


Figure 1: The Stone SouperComputer at Oak Ridge National Laboratory.

We became involved in cluster computing more than two years ago, after developing a proposal for the construction of a Beowulf cluster to support a handful of research projects. The proposal was rejected, but because we had already begun development of a new high-resolution landscape ecology application, we decided to build a c lusterout of surplus PCs (primarily Intel 486s) destined for salvage. We began intercepting excess machines at federal facilities in Oak Ridge, Tennessee, and processing them into usable nodes. By September 1997, we had a functional parallel computer system built out of no-cost hardware. Today we have a constantly-evolving 126 node highly heterogeneous Beowulf-style cluster, called the Stone SouperComputer, which is continuously upgraded as better hardware becomes available and which is used for algorithm development and for running parallel applications. For more information about the Stone SouperComputer, see Hoffman and Hargrove [3] or visit our World Wide Web (WWW) site at **www.esd.ornl.gov/ facilities/beowulf**.

Building A Beowulf - A Lite Tutorial

Anyone can construct a parallel computer adequate for teaching parallel programming and running parallel codes--often using existing or excess PCs. PCs in an established computer laboratory can be adapted for dual use, dual-boot systems so that they can be rebooted into either Linux or Microsoft Windows, depending on the present need. Alternatively, unused equipment can be collected and fashioned into a parallel system as we have done.

No two Beowulf clusters are the same. In fact, their hardware and software configurations are so flexible and customizable that they present a wide array of possibilities. In this tutorial, we hope to provide some guidelines and considerations for narrowing this wide field of choices. While every Beowulf cluster is different and configurations are dictated by application needs, some minimum requirements can be specified.

Minimum Requirements

A node should contain at least an Intel 486 CPU and motherboard. Intel 386 processors will work, but performance will make it hardly worth the effort. Memory requirements depend on the target applicati on'sdata requirements and parallelism, but a node should have at least 16 MB of memory. Most applications will need 32 MB or more per node. By using centralized disk space, nodes can boot from a floppy disk, a small hard drive, or a network file system. Then nodes can access their root partitions from a file server over the network, usually via the Network File System (NFS). This configuration works best in an environment with a high bandwidth connection and a high performance file server. For best performance under other conditions, local disk space for the operating system, swap space, and data should be available on each node. Each node should have at least 200 MB of disk space for operating system components and swap space, but 400 MB or more allows for some free space which may be used by applications at run time. At least one Ethernet or Fast Ethernet network interface must be included in each node. Alternatively, higher performance interconnects, including Gigabit Ethernet and Myrinet, could be used in conjunction with faster CPUs. Finally, any video card, a floppy disk drive, and a case and power supply round out a functional node. Keyboards and monitors are required only for the initial loading and configuration of the operating system, unless the individual machines are used interactively in addition to serving as nodes in the parallel system.

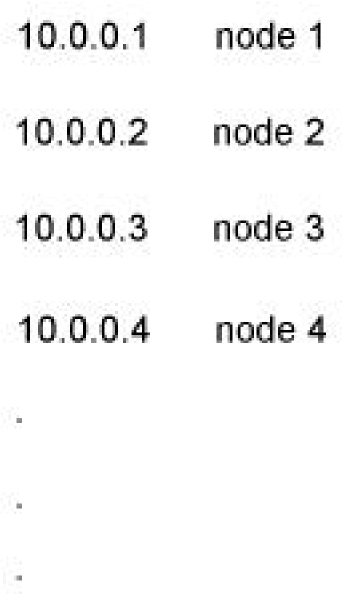
All the chosen hardware components need support from drivers or modules in Linux. Generally this is not a problem if the hardware is more than a few months old. Many sources of information about supported hardware are available on the WWW (see **Resources** below). Of particular interest is Donald Becker's fairly complete set of drivers and excellent documentation for a wide variety of network interface cards.

Video card support is not important on nodes since they do not usually run an X server; however, the master node may handle management of the cluster. An X server would be handy for this machine. If a particular component presen tsa problem, inquiries on Usenet news groups can help find information about hardware supported by Linux.

Network Connectivity

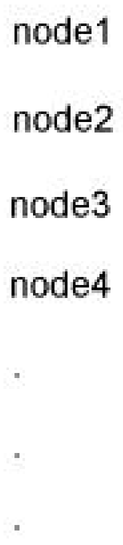
If possible, the nodes should be isolated on a private local area network with their own Ethernet hub or switch. This will keep normal network traffic from interfering with inter-node communication and vice versa. In order to further increase inter-node bandwidth, additional network interface cards can be installed in nodes. Channel Bonding software available at **www.beowulf.org** can be used to operate parallel communication networks. The first or master node in the cluster should have an additional Ethernet card to connect it to the normal, routed network as well as to the private network. This is useful for user logins and file transfers. On the private network, it is necessary to use a block of IP addresses not used elsewhere on the Internet. It is usually easiest to use the Class A `10.0.0.0` address space which is reserved for non-routed networks such as these. In this case, the `/etc/hosts` file on each node would appear as shown in, and the `/etc/hosts.equiv` file on each node would appear as shown in. An example `ifcfg-eth0` configuration file (used by Red Hat Linux) for node number 2 is shown in.

Table 1: `/etc/hosts` file for all nodes



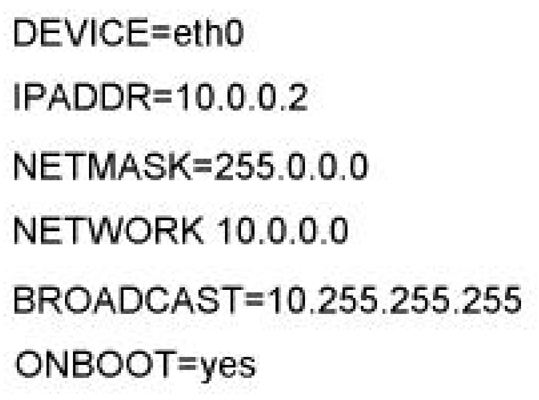
```
10.0.0.1     node 1

10.0.0.2     node 2

10.0.0.3     node 3

10.0.0.4     node 4

.

.

.
```

Table 2: `/etc/hosts.equiv` **file for all nodes**



```
node1

node2

node3

node4

.

.

.
```

Table3: `/etc/sysconfig/network-scripts/ifcfg-eth0` **file for node 2**



```
DEVICE=eth0
IPADDR=10.0.0.2
NETMASK=255.0.0.0
NETWORK 10.0.0.0
BROADCAST=10.255.255.255
ONBOOT=yes
```

In addition, it is often desirable to have a Domain Name Server (DNS) available, particularly if node names or addresses are likely to change on the private network. DNS can run on the first node to provide name/address resolution for the nodes on the private network.

Local Storage

At the point of loading the operating system, Beowulf builders must commit to a number of storage configuration decisions. These decisions merit careful forethought, since changes require re-installing all nodes. The majority of Linux-based Beowulf clusters are running the Red Hat Linux distribution, but any distribution should support basic clustering. Optional kernel patches for providing a Global PID space and other tweaks were engineered primarily for Red Hat, but these features are not required for parallel computing. Loading Red Hat is fairly straightforward and can be accomplished via a CD-ROM or the network interface from the Internet or from the first node in the cluster, which would have to keep a copy of the distribution at the ready. We have found it beneficial to actually load the operating system onto the disks of each node via FTP from the primary node instead of mounting root partitions via NFS. Thus we sacrifice some convenience in maintenance for better run-time performance. This practice eliminates unnecessary network traffic, preserving bandwidth for message passing while applications are running.

The Red Hat Linux run-time environment requires only about 100 MB of disk space on each node; however, we find it useful to include compilers and other utilities on all nodes so thatparallel compiling is possible. Therefore, our configuration requires about 175 MB of disk space for the operating system. While some clusters are setup to swap on a file on the regular filesystem, it is more efficient to use a dedicated swap partition on a local disk for virtual memory. It is generally accepted that a reasonable amount of swap space is twice the size of available memory up to 64 MB. For nodes with greater than 64 MB of memory, swap space should be equal to the amount of memory. In practice, one may wish to allocate 128 MB of swap for a memory size of 64 MB through 128 MB. As a result, if a prospective node with 32 MB of memory has two 202 MB disk drives, one would load Linux onto the primary drive (as a single partition) and use the secondary drive for swap space (64 MB) and local run-time space (138 MB).

Cluster Management

System management and maintenance can be tedious, particularly for large clusters. However, a number of utilities and scripts are available on the WWW to assist with this task (see **Resources** below). For example, the nodes must be kept "in-sync" with each other with respect to time and systems files (e.g., `/etc/passwd`, `/etc/group`, `/etc/hosts`, `/etc/hosts.equiv`, etc.). Simple shell scripts executed regularly by `cron` can take care of the majority of synchronization.

Once all the nodes are loaded and configured, parallel applications can be designed and developed to take advantage of the new system.

Developing Parallel Applications For Cluster Computing

Both commercial and free compilers are available for Linux. The GNU Project's C (gcc), C++ (g++), and FORTRAN (g77) compilers are included with most standard Linux distributions. The C and C++ compilers are excellent and the FORTRAN compiler is constantly improving. Commercial compilers are available from Absoft, The Portland Group (PGI), The Numerical Algorithms Group (NAG), and others. Some of the commercial FORTRAN-90 compilers will automatically parallelize some operations if set up properly. In general, though, developing parallel code will require explicit message passing between processors using PVM (Parallel Virtual Machine), MPI (Message Passing Interface), or another communications library. Both PVM and MPI are freely available and allow the programmer to easily define the nodes used for running a parallel code and to pass data between nodes during computation using simple library calls.

Of course, not every computational task is amenable to parallelization. Often, completely new strategies for solving computational problems must be developed in order to take advantage of parallelism. In fact, it may be necessary to step back from algorithms in existing serial code in order to conceive of the most scalable parallel approach. Many scientific problems can benefit from domain decomposition: a splitting up of space or time into relatively independent chunks which can be processed on separate nodes in a cluster. For instance, image processing tasks can often be subdivided so that each node works on a section of a single image. This approach works particularly well if sections in an image are processed independently (i.e., processing a section does not require information about other sections) or if the radius of influence for each section is predetermined and constant.

One under-appreciated technique for jumping into the world of parallel computing with almost no code development is to employ a "poor man's parallelism," i.e., multiple serial codes running simultaneously on multiple nodes. This "multiply up" strategy, because it has **no** communication between processes, is perfectly scalable, and yields a significant performance advantage with little additional effort.

One of the most dangerous pitfalls--whether one is parallelizing existing code or developing new parallel code--is turning a computational problem into a communications problem. This can ha ppenwhen the problem is over-divided such that the time it takes to communicate data between nodes and to synchronize the nodes exceeds the actual CPU computing time. In this case, using fewer nodes may result in better run times and more efficient utilization of resources. This trade-off between local computational load on a node and communications to coordinate efforts among nodes must be optimized for every parallel application.

Finally, cluster heterogeneity plays an important role in developing parallel algorithms. Factors of two or more between node CPU speeds are significant when running parallel applications. If the work is simply distributed evenly over all CPUs in a heterogeneous cluster, the faster CPUs must wait for the slower CPUs to complete their part of the overall task. Properly designed algorithms can handle this heterogeneity by over-dividing the task and "dealing out" new aliquots as nodes complete previously assigned aliquots. Of course, this approach must be balanced by considering communications overhead.

Parallel processes can be organized in many different ways, but the master/slave organization is the easiest to understand and program. In a master/slave organization, one node serves as process master while the other nodes serve as slaves. The master node usually decides how to divide up the work and generally "directs traffic" while the slave nodes process their assigned aliquots and "report in" when finished. Variants of this organization include a master-less approach in which all nodes perform the same tasks or a hierarchical approach in which multiple tiers work together or report to a super-master. The best approach is determined by the algorithm or application needs.

Source code can be organized in different ways as well. In a master/slave processing organization, one might develop one set of source code for the master and another set for the slaves. This was common practice on early parallel systems, but today the Single Program Multiple Data (SPMD) method is themost popular and often easiest to maintain. With SPMD, a single set of source code is developed so that one binary is executed on all nodes. The program branches depending on the desired functionality of each node; masters perform one set of operations while slaves perform another.

Many approaches are possible for the organization and distribution of data used by parallel codes. One possibility is to serve data to all nodes via a central NFS server. However, if several nodes attempt to access the data simultaneously, this strategy will lead to network contention. One alternative is to pass chunks of data between nodes using message passing, but this also puts a heavy load on the network depending on the size of the data and nature of the problem. Another alternative is to distribute the data to local disks on all nodes prior to run time. This manual distribution of the data to all nodes leaves the network clear for actual message passing during run time. Similarly, the nodes can assemble the answers and store the results centrally, or results may be stored locally and harvested after the completion of the run. We have found this "distribute and harvest" approach to be particularly useful.

Conclusions

There are no hard and fast rules about either building a machine or developing parallel code because **it depends on your application**. Optimizing hardware configurations and algorithm approaches cannot be predicted without knowing specific details about **your application**. In a shared or heterogeneous system, optimizing the balance between the load on nodes and the communication between nodes depends on the nature of the hardware. Faster communication encourages finer division of the task, while coarser division is favored by having faster nodes.

The "Beowulf movement" has brought parallel computing to the grass-roots level. Parallel codes developed on Beowulf systems using standard message passing libraries can be run without modification on commerc ialsupercomputers. Thus, Beowulf clusters represent an entry point and later a bridge to even the largest supercomputers in the world. Further, the availability of inexpensive, commodity clusters means that parallel computer environments can be dedicated to specific tasks, unlike large commercial supercomputers which are too expensive to dedicate and cannot be optimized for any single application. As parallel environments become increasingly available to programmers, the collective experience gained will promote parallel applications in every field of endeavor.

Resources

- The Stone SouperComputer**www.esd.ornl.gov/facilities/beowulf**
- The Beowulf Project**www.beowulf.org**
- The Beowulf Consortium and Beowulf Mailing Lists**www.beowulf.org/consortium.html**
- Extreme Linux**www.extremelinux.org**
- Red Hat Linux**www.redhat.com**
- Radajewski and Eadline's Beowulf HOWTO**www.sci.usq.edu.au/staff/jacek/beowulf/HOWTO/**
- Lindheim's Beowulf Tutorial: Building a Beowulf System **www.cacr.calte ch.edu/beowulf/tutorial/building.html**
- Don Becker's Linux Network Drivers**www.beowulf.org/linux/drivers/**
- Linux Newsgroups: alt.os.linux.*, comp.os.linux.*, linux.*
- **alt.os.linux**,
- **comp.os.linux.hardware**,
- **comp.os.linux.misc**,
- **comp.os.linux.networking**,
- **comp.os.linux.setup**
- Myricom, Inc. (manufacturers of Myrinet)**www.myri.com**
- Sterling, et.al., *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters* (expected May 1999)**mitpress.mit.edu/book-home.tcl?isbn=026269218X**
- PVM (Parallel Virtual Machine)**www.epm.ornl.gov/pvm**
- MPI (Message Passing Interface)**www.mcs.anl.gov/mpi**

References

1

Alper, Joseph. 11 December 1998. "From Army of Hackers, an Upstart Operating System." *Science*, Vol. 282, No. 5396, pp. 1976-1979.

2

Becker, Donald J., Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V. Packer. 1995. Beowulf: A Parallel Workstation for Scientific Computation. *Proceedings, International Conference on Parallel Processing*.

3

May, Mike. March 1998. "Piles of PCs." *American Scientist*, Vol. 26, pp. 128-129.

4

"Supercomputers: Plug and Play." 21 November 1998. *The Economist*.

5

Hoffman, F. M. and W. W. Hargrove. March 1999. "Cluster Computing: Linux Taken to the Extreme." *Linux Magazine*, Vol. 1, No. 1, pp. 56-59.

Biography

Forrest M. Hoffman**forrest@esd.ornl.gov** is a computer specialist in the Environmental Sciences Division at Oak Ridge National Laboratory in Oak Ridge, Tennessee, where he develops parallel (and serial) environmental models as well as tools for scientific visualization, large data management and archival, systems administration, and Internet technologies. In his spare time, he builds parallel computers.

William W. Hargrove **hnw@fire.esd.ornl.gov** is a member of the research faculty at the University of Tennessee's Energy, Environment, and Resources Center, serving on contract to the Oak Ridge National Laboratory's Geographic Information and Spatial Technologies Group, and has expertise in computer algorithms, Geographic Information Systems, landscape ecology, and simulation modeling.

**Oak Ridge National Laboratory, managed by Lockheed Martin Energy Research Corp. for the U.S. Department of Energy under contract number DE-AC05-96OR22464.*

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-96OR22464. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."