U.S. DEPARTMENT OF THE INTERIOR
U.S. GEOLOGICAL SURVEY

A Parallel-Processing Approach to Computing for the Geographic Sciences: Applications and Systems Enhancements

by

Michael Crane,[1] Dan Steinwand,[2] Tim Beckmann,[2] Greg Krpan,[2] Shuguang Liu,[2] Erin Nichols,[2] Jim Haga,[3] Brian Maddox,[4] Chris Bilderback,[4] Mark Feller,[5] and George Hamer[6]

Open-File Report 01-465

2001

[1] U.S. Geological Survey, EROS Data Center, Sioux Falls, SD 57198-0001
[2] USGS, EROS Data Center, Raytheon ITSS, Sioux Falls, SD 57198-0001
[3] USGS, EROS Data Center/Alaska Field Office, Raytheon ITSS, Anchorage, AK 99508-4664
[4] U.S. Geological Survey, Mid-Continent Mapping Center, Rolla, MO 65401
[5] U.S. Geological Survey, Rocky Mountain Mapping Center, Denver, CO 80225
[6] South Dakota State University, Computer Science Department, Brookings, SD 57007

CONTENTS

KEY WORDS…… 6

ABSTRACT…… 6

INTRODUCTION…… 7

BACKGROUND…… 7

HYPOTHESES…….. 8

PARTICIPATING SITES…….. 8

SYSTEMS ENHANCEMENTS AND APPLICATIONS …… 9

ERROS DATA CENTER…….. 9

Hardware and Systems Software Enhancements……. 10

Urban Growth Model: Benchmarks and the 3x Problem…….. 11

Map Projection Change Resampling – A “Process Parallel” Approach……... 15

Beowulf Tile-Based Image Resampler…… 19

Master Node Process Flow……. 22

Slave Node Process Flow……. 24

Improvements……. 25

Image Processing with the Land Analysis System ……. 28

Implementation of the Century Model……. 28

The Linux Computer Lab as a Beowulf Cluster……. 30

MID-CONTINENT MAPPING CENTER……. 30

Parallel Virtual File System…… 33

Flat Network Neighborhood…….. 34

Processing Partitioning Method…….. 35

Fiscal Year 2001 Research……. 37

Future Work……. 46

ROCKY MOUNTAIN MAPPING CENTER…….. 47

Hardware and Systems Software Enhancements…… 50

SLEUTH Urban Growth Model…….. 51

ALASKA GEOGRAPHIC SCIENCE OFFICE……. 53

Hardware and Systems Software Enhancements…….. 53

Applications…… 55

FUTURE DIRECTIONS AND ACTIVITIES…….. 55

Summary of Fiscal Year 2001 Accomplishments……... 55

Proposed Fiscal Year 2002 Workplan……. 56

BIBLIOGRAPHY…….. 61

SELECTED REFERENCES…….. 62

IMPORTANT WEB SITES……. 66

APPENDIXES…….. 67

APPENDIX A: Quick and Easy GUIs with Tcl/Tk…… 68

APPENDIX B: A User Guide for the Beowulf Resampler……. 82

ILLUSTRATIONS

Figure 1. The 13-node Beowulf cluster at EDC…… 9

2. SLEUTH benchmarks for a subset of the Sioux Falls dataset…. 12

3. SLEUTH benchmarks for a subset of the Sioux Falls dataset after system upgrades……. 12

4. Benchmarks illustrating the factor of 3 problem…….. 13

5. Runtimes, early finish times, and late finish times…….. 14

6. SLEUTH benchmark with Kernal 2.4 and a new random number generator……. 15

7. Nearest neighbor resampling method…… 16

8. Maximum occurring pixel resampling method…….. 17

9. Example of Tcl/Tk-based GUI on the EDC cluster……. 19

10. Grid model and bilinear resampling with a 13-node cluster…….. 26

11. Grid model and weight-table-based resampling with a 13-node cluster…….. 27

12. Polynomial model and nearest neighbor resampling with a 13-node cluster…… 27

13. Polynomial model and nearest neighbor resampling with a three-node cluster…… 27

14. Grid model and cubic convolution resampling with a three-node cluster…… 28

15. Grid model and bilinear resampling on a 13-node cluster…… 28

16. At 18 nodes, the MCMC system is the largest of the Beowulf clusters…….. 31

17. PVFS logical system design (from the PVFS Web site)…….. 34

18. Diagram of a simple FNN (from FNN Web site)…… 35

19. Processing partitioning diagram…… 36

20. Chart of the minimum runtime theory…..…….…….….. 38

21. Minimum runtimes with curve fit…..……. 39

22. Runtimes – Kernel 2.4 vs. 2.2…….……..……...…. 42

23. PVFS access versus NFS access run times…….……. 43

24. The MCMC FNN setup……... 44

25. The 16-node Beowulf cluster at RMMC……... 49

26. Benchmarks based on data from the Isleta, NM, quadrangle……... 53

27. At six nodes, the AGSO Beowulf cluster makes a compact and space-efficient layout…….. 54

KEY WORDS

Parallel Processing, Beowulf Clusters, High-Performance Computing, Modeling

ABSTRACT

The overarching goal of this project is to build a spatially distributed infrastructure for information science research by forming a team of information science researchers and providing them with similar hardware and software tools to perform collaborative research. Four geographically distributed Centers of the U.S. Geological Survey (USGS) are developing their own clusters of low-cost, personal computers into parallel computing environments that provide a cost-effective way for the USGS to increase participation in the high-performance computing community. Referred to as Beowulf clusters, these hybrid systems provide the robust computing power required for conducting information science research into parallel computing systems and applications.

INTRODUCTION

The project described in this paper is a continuation of work that commenced in fiscal year (FY) 2000 with the identification of individuals at U.S. Geological Survey (USGS) Mapping Centers interested in building an information science research infrastructure within the National Mapping Discipline (NMD). At that time, employees at USGS sites (the EROS Data Center (EDC) in Sioux Falls, SD, the EDC/Alaska Field Office[7] (AFO) in Anchorage, AK, the Mid-Continent Mapping Center (MCMC) in Rolla, MO, and the Rocky Mountain Mapping Center (RMMC) in Denver, CO) prepared and submitted a research proposal to begin investigations into high-performance computing. A follow-on proposal for continued funding, approved for FY 2001, has enabled the Centers to make performance and communication enhancements on their existing clusters and to test a variety of applications on these systems.

By focusing on the clustering of low-cost commodity computers into larger (but still low-cost), parallel computing systems, this project has provided a cost-effective way for the USGS to broaden participation in the high-performance computing community.

BACKGROUND

In recent years, there has been significant interest in the clustering of low-cost computers into affordable, multiprocessor, parallel computing systems. These low-cost parallel computers are typically constructed with personal computers either running open-source UNIX-like operating systems, such as the Beowulf concept (Sterling et al., 1998), or using the platform's native operating system, as in Project AppleSeed (Decyk et al., 1999). The most common implementation of these systems is the Beowulf concept, which is typically constructed with commercial off-the-shelf Intel Pentium-based computers running the open-source Linux operating system.[8] Although the concept of building these systems is no longer cutting-edge research (their construction is well documented – see references), the implementation of applications on these systems is still very much in the forefront of research. Dealing with the increasingly large amounts of data needed to investigate complex geospatial-temporal problems and deciding how to implement these applications in the message-passing environment (Snir et al., 1998; Gropp et al., 1999a) common to Beowulf systems were not well understood.

Open-File Report 01-244, entitled "A Parallel-Processing Approach to Computing for the Geographic Sciences," details the hardware and software configurations

[7] The Alaska Field Office has recently been renamed the Alaska Geographic Science Office (AGSO).

[8] Any use of trade, product, or firm names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

as installed at each Center up to the mid-point of FY 2001. The current report is an update to that earlier work and documents the modifications made to each system to optimize numeric processing and large image dataset processing performance. It also documents advances made in numeric and image processing algorithms by the project.

The implications of having supercomputer performance in a package with a price comparable to that of a UNIX-based workstation may be significant to those attempting to use compute-intensive modeling, visualization techniques, and data-rich analysis in their research. These systems provide cost-effective alternatives to purchasing expensive computing servers or access fees at supercomputer centers.

HYPOTHESES

A goal of this project is to evaluate objectively the suitability of Beowulf clusters for supporting the work of the NMD. This will be accomplished by testing several hypotheses about Beowulf clusters. In no particular order, the hypotheses that will be tested are as follows:

- Runtimes of numerically intense modeling processes will be significantly faster on Beowulf clusters than on workstation environments currently available to scientists.

- While network and computer configurations are held constant, runtimes of numerically intense modeling processes can be improved on Beowulf clusters through data-partitioning methods.

- Algorithm-partitioning methods can improve runtimes of numerically intense modeling applications on Beowulf clusters while holding network and computer configurations constant.

Finding more efficient methods for running models and processing extremely large datasets can aid these programs by offering faster runtimes and by allowing more data to be processed than is currently feasible on single-node computer systems. This is especially true in cases involving the handling of gigabyte-sized, multitemporal databases, such as those composed of satellite images.

PARTICIPATING SITES

Four centers of the NMD are participating in this project: the EDC in Sioux Falls, SD, which serves as the Nation's archive for land remote sensing data and manages the Landsat 7 satellite system in partnership with NASA; the EDC/AGSO, located in Anchorage, AK, which conducts research and applications projects with a broad spectrum of other Alaska-based Federal and

State agencies and international collaborators; the MCMC in Rolla, MO and the RMMC in Denver, CO, both of which engage in the generation of cartographic products, research related to cartographic and spatio/temporal data production and application, and integrated science investigations.

SYSTEMS ENHANCEMENTS AND APPLICATIONS

EROS DATA CENTER

The Beowulf cluster at the EDC is installed in the Showcase Visualization Lab, a hub facility that provides centerwide access to high-end demonstration and visualization systems and technical support (fig. 1). The system comprises 1 master node, 1 data node, and 11 computing nodes, all connected to a dedicated cluster network to ensure the best communications for any given computing task. The master node is a dual-processor Dell Precision 420 Workstation running dual

Figure 1. The 13-node Beowulf cluster at EDC.

Pentium III processors at 733 MHz with 768 MB of RDRAM memory and 50 GB of disk capacity. The computer has three network interface cards (NIC); one is connected to the EDC network, one to the data node, and one to the dedicated

cluster network. The master node is equipped with a 21-inch color monitor, a keyboard, and a mouse. The data node is a dual-processor Pentium III running at 933 MHz with 512 MB of memory and 144 GB of fast SCSI disk capacity. The system is equipped with three NICs. Dedicated data lines are provided to each cluster computing node, with the addition of a second NIC in each computing node (the third for the master node) and a dedicated switch. The data node will eventually be connected to the EDC network and will have other data ingest/backup capabilities.

The computing nodes consist of 11 Dell OptiPlex GX110 computers, each running a Pentium III processor at 733 MHz with 256 MB of memory and 50 GB of disk capacity. Each computing node is currently equipped with dual NICs; one connected to the master node network and one to the data node network. The computing nodes are not accessible as individual computers on the EDC network.

The cluster network currently consists of two 24-port, 3Com SuperStack 3 3300 switches. In addition, a KVM switch is used, allowing the master, data, and computing nodes to share a keyboard, video monitor, and mouse for debugging and maintenance purposes. All computers in the cluster are protected with APC battery backup units (three APC Back-UPS Pro 1400s and one APC Back-UPS 700).

Hardware and Systems Software Enhancements

Since the beginning of the FY 2001 research period, hardware and software upgrades were performed to the EDC Beowulf cluster, which resulted in effectively rebuilding the cluster from the ground up. The first change was the addition of the data node to the cluster. This server functions as a dedicated data repository, sharing data across the cluster. In conjunction with the data server, a second network interface was installed on each node within the cluster. A second 3Com OfficeConnect switch was added to provide connectivity on the second network. Performance became an issue on the second network as the switch was not able to maintain a 100-mBit, full-duplex link. To correct the performance issue, both existing 16-port, 3Com OfficeConnect switches were replaced with 24-port, 3Com SuperStack 3 3300 switches.

During the system refinement process, a problem was identified with Network File System (NFS) timeouts. When a computing node would attempt to communicate results back to the master node, it would often be blocked owing to stale NFS file handles. This means that the NFS daemon on the master node would not respond to the computing node. All computing nodes use an NFS-mounted file system to write their results. The solution to the problem was found after some investigation into NFS; the number of daemon processes started on the server was less than the maximum number of possible requests that could come in at one time. Because there are 12 client nodes in the cluster, the value

of the variable RPCNFSDCOUNT was increased to 14 in the daemon file /etc/rc.d/init.d/nfs. This solved the timeout problem.

The operating system was also upgraded, bringing the system from Red Hat Linux 6.2 to SuSE Linux 7.1. The Linux kernel was upgraded from version 2.2.14 to 2.4.0 during this procedure. Lessons learned in constructing the original cluster allowed the rebuilt cluster to be more streamlined and efficient in design and operation. The inclusion of packages relevant to the operation of the cluster in the operating system install allowed a greater ease in configuring the base cluster for use.

Currently, the data server is functioning as a computing node, and the second network segment is functioning as an NFS data network. When processing requirements shift from being CPU and memory intensive to being data intensive, the data server will be transitioned out of its role as a computing node and will become a dedicated data server.

Urban Growth Model: EDC Benchmarks and the 3x Problem

SLEUTH, formerly referred to as the Clarke Urban Growth Model, is a cellular automata model algorithm obtained from the Environmental Protection Agency (EPA), where it was modified to run with the Message Processing Interface (MPI) on their Cray supercomputer system (a more detailed description of SLEUTH may be found on p. 47). This set of software was compiled with Local Area Multicomputer (LAM)/MPI, and it ran on the EDC cluster with very little modification. Further investigation revealed that several minor modifications and bug fixes were required for accurate and efficient execution on the Linux-based cluster. A geospatial dataset (Sioux Falls, SD), has been run through SLEUTH with the EDC cluster configured with 4, 8, and 12 nodes. The model and data were also run on a Sun Ultra 60 workstation that has a purchase price similar to that of the cluster. Initial results yielded run times of 77 hrs. 30 min. on 4 cluster nodes, 39 hrs. 52 min. on 8 cluster nodes, 31 hrs. 9 min. on 12 cluster nodes, and 1,354 hrs. 39 min. on the Sun Ultra 60.

Additional benchmarks were run on a subset of the Sioux Falls dataset, showing the speeds obtained with each cluster node added. Before the data node, the new switches, and the operating system upgrade were added, the runtimes shown in figure 2 below were observed:

Number of Nodes	Run 1 (mm:ss)	Run 2 (mm:ss)	Avg. Speedup	% of Theoretical Max
12	95:43	95:46	3.76	31
11	70:17	70:15	5.13	47
10	80:58	80:58	4.45	45
9	120:35	120:34	2.99	33
8	100:10	100:09	3.60	45
7	114:33	114:34	3.14	49
6	180:57	181:00	1.99	33
5	154:12	154:13	2.34	47
4	194:37	194:38	1.85	46
3	357:19	357:22	1.01	34
2	375:50	375:55	0.96	48
1	360:08	360:08	1.00	100

Figure 2. SLEUTH benchmarks for a subset of the Sioux Falls dataset.

After performing the system upgrades described above and using the data node as a the first computing node, researchers observed the run times shown below in figure 3:

Number of Nodes	Run 1 (mm:ss)	Run 2 (mm:ss)	Avg. Speedup	% of Theoretical Max
13	35:44	35:38	10.06	77
12	50:38	50:25	7.10	59
11	38:58	39:11	9.19	84
10	42:53	42:44	8.39	84
9	60:05	60:02	5.98	66
8	48:08	48:18	7.44	93
7	57:12	57:09	6.28	90
6	90:01	90:16	3.98	66
5	75:53	75:50	4.73	95
4	95:11	95:44	3.76	94
3	177:43	177:30	2.02	67
2	180:49	185:37	1.96	98
1	358:48	359:06	1.00	100

Figure 3. SLEUTH benchmarks for a subset of the Sioux Falls dataset after system upgrades.

It should be noted that the data node is comparable in price/performance with the master node and, therefore, runs the problem more efficiently than the other computer nodes. Refer to USGS Open-File Report 01-244 for the original configuration of the EDC cluster (as configured for the first set of timings).

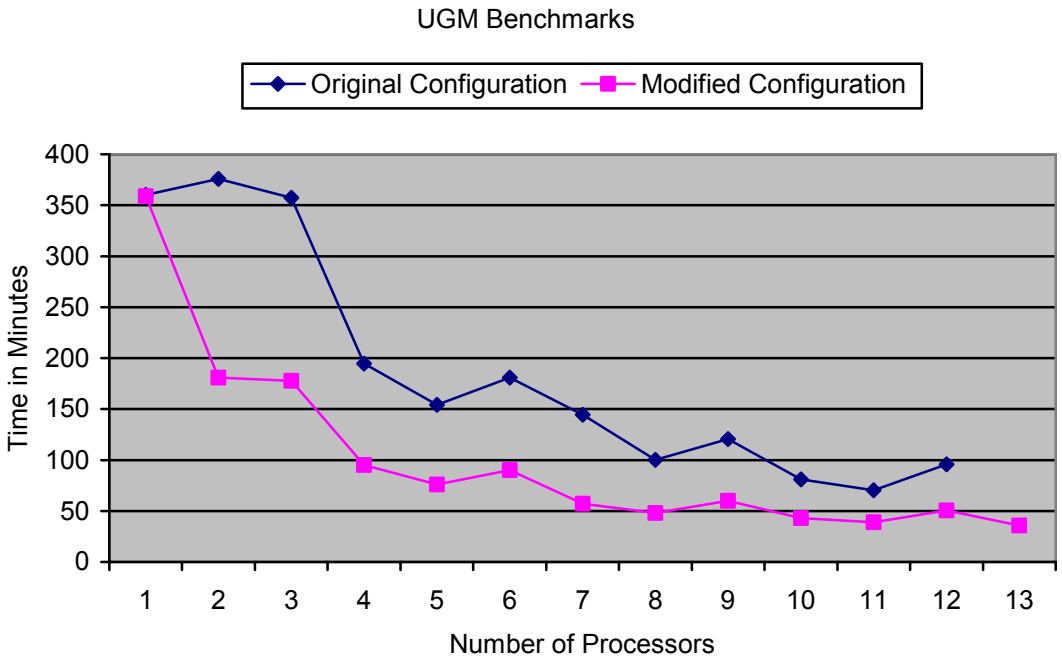

Figure 4. Benchmarks illustrating the factor of 3 problem.

Looking at the runtimes, both before the system upgrades and after, it's apparent that processing speedup is not linear. In fact, for each factor of 3 nodes, there is a significant increase in the amount of time required to process the dataset, as evidenced in figure 4 above.

To investigate this condition further, researchers conducted additional tests with the SLEUTH software on the EDC cluster, the Midas cluster at North Dakota State University (NDSU), and the cluster at South Dakota State University (SDSU). The NDSU cluster is an older machine built with AMD K6 266 MHz computers. The configuration of the SDSU cluster is given later in this report under the section entitled "The Linux Computer as a Beowulf Cluster."

During testing, it was discovered that a flaw exists in the original SLEUTH algorithm. Any time the number of processors is a factor of 3, the early and late stop times of nodes become much larger than if the number of nodes is not a multiple of 3. Figure 5 below gives runtimes and early finish times (when the first node finished) and late finish (when the last node finished) times:

EROS DATA CENTER RUNS

# of Nodes	Total Time (hh:mm:ss)	Early Finish (hh:mm)	Late Finish (hh:mm)	Difference (in minutes)
12	0:49:43	0:30	0:49	**19**
11	0:38:16	0:35	0:38	3
10	0:43:45	0:36	0:42	6
9	1:07:04	0:31	1:07	**36**
8	0:51:47	0:47	0:50	3
7	0:58:57	0:51	0:58	7
6	1:32:26	0:46	1:32	**44**

NDSU MIDAS CLUSTER RUNS

15	1:32:24	0:31	1:32	**61**
14	1:11:15	0:44	1:08	24
13	1:20:12	0:51	1:10	19

SDSU CLUSTER

14	0:41:38	0:34	0:41	7
13	0:52:01	0:35	0:47	12
12	1:07:41	0:29	1:07	**38**

Figure 5. Runtimes, early finish times, and late finish times for three Beowulf clusters.

Notice that in all cases a multiple of three nodes leads to an increasing difference between the earliest and latest finish. Notice also that, in some cases, decreasing the number of nodes makes the job run faster!

It was first thought that the random-number generator used in the algorithm was the cause of this periodicity. The original algorithm uses the linear congruential method (Knuth, 1997), and Knuth's subtractive method was tried next. Both algorithms can be found in chapter 7 of "Numerical Recipes" (Press, et.al., 1992). Substituting the random-number generator shows no improvement in the code, leading the researchers to believe that the division of work between the cluster nodes is to blame. See figure 6 below:

SDSU Cluster

Number of Nodes	Total Time (hh:mm:ss)	Early Finish (mm:ss)	Late Finish (mm:ss)	Difference (in minutes)
14	0:44:45	00:33	00:44	11
13	0:46:51	00:35	00:47	12
12	1:07:55	00:29	01:08	**40**
11	0:59:13	00:44	00:59	15
10	1:03:55	00:47	01:04	17
9	1:24:30	00:38	01:24	**46**
8	1:11:34	01:04	01:12	8

Figure 6. SLEUTH benchmark with Kernal 2.4 and a new random-number generator.

Further research is required to determine the real cause of this 3x problem; however, this was considered by the researchers to be beyond the scope of this investigation. It was agreed, however, that a load-balancing scheme that also provides some fault tolerance is necessary to achieve the maximum obtainable speedups with this algorithm.

Map Projection Change Resampling -- A "Process Parallel" Approach

A new method for resampling categorical data was developed for another NMD prospectus project. Because of the high number of numeric computations, this algorithm became a candidate for implementation on the EDC cluster and, thus, a part of Beowulf clustering investigations.

Commonly available methods for resampling categorical data (class or binned data, not signal-based, remote sensing data) typically consist of nearest neighbor-like resampling methods. These methods are chosen because their alternatives - cubic convolution, bilinear interpolation, etc. - are interpolating methods, which do not maintain categorical values. Furthermore, the geometric distortions present in the projection change of data of global extent are far greater than those that occur in moderate to high-resolution remote sensing data. Indeed, most of the software tools available today were designed for single-scene, signal-based remote sensing image data, where the image extent usually extends only a few hundred kilometers, rather than for datasets of global or continental extent.

The typical nearest neighbor algorithm for categorical resampling takes the center (or upper-left corner) of a pixel as point data and reprojects that coordinate into the new projection space. Because the resulting coordinate is often not at an exact pixel location, it is rounded to the nearest pixel position, and that pixel's value is used to populate the resulting output image. Although this

method is computationally efficient, it can result in imagery that is not representative of the original image owing to varying amounts of geometric distortions present in the transformation.

The new resampling method treats pixels not as points, but as areas. In summary, the new algorithm maps the corner coordinates of the image pixels between the two projections and determines the number of input image pixels that go into making the resulting output image pixel. If one pixel (or less than one complete pixel) goes into the making of the output image pixel, the nearest neighbor approach is used. More often, however, multiple input image pixels go into one output image pixel. In this case, simple statistical methods (min, max, average, etc.) are used to determine the output pixel value. This many-to-one condition is present when the output image pixel size is greater than that of the input. It is also present in transformations of similar pixel sizes where geometric distortions are great (for example, an image in a projection where the pole is represented as a line to a projection where the pole is represented as a point.) The algorithm also keeps track of pixel values that were present in the input area but were not used. These are written to a statistical image that can assist the end user in determining the extent of data loss/misrepresentation.

The two images below illustrate the output of the algorithm. The image in figure 7 was processed using nearest neighbor resampling; the image in figure 8 was processed with the maximum occurring pixel method.

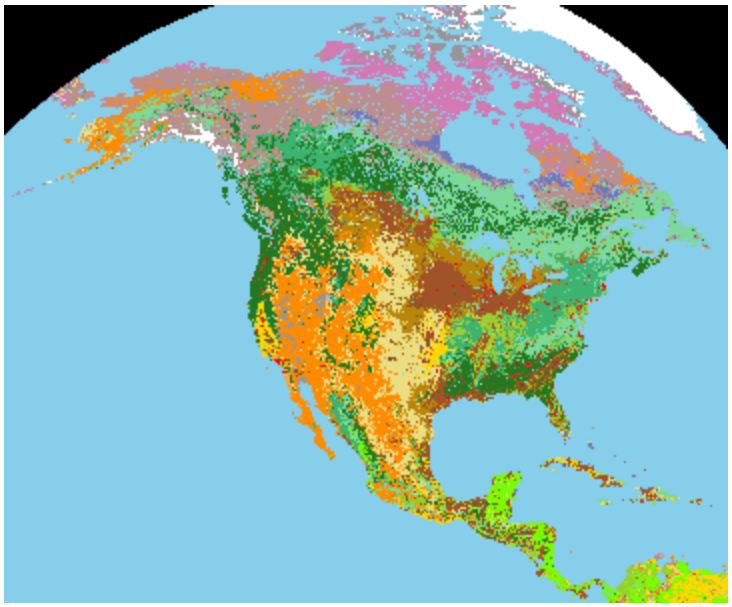

Figure 7. Nearest neighbor resampling method.

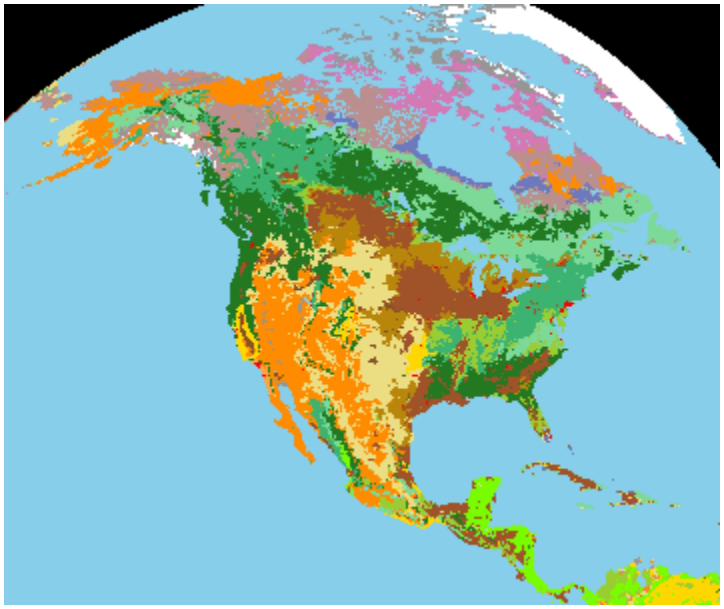

Figure 8. Maximum occurring pixel resampling method.

The new resampling algorithm was implemented as a C-language image processing routine that reads the entire input image into memory and processes the reprojection with an inverse-mapping algorithm, writing each output image line as it is processed. With an input image of just under 1 GB, this routine has a run time of approximately 25 minutes on the master node for a global map with a 25-km output pixel. Run times with imagery closer to the input resolution (approximately 1 km) take significantly longer.

This algorithm can be easily parallelized in a "process-parallel" fashion (Sterling et al., 1999). The parallel algorithm takes advantage of the inverse-mapping style of the original algorithm and simply uses the serial reprojection program (with no rewriting) by specifying subsets of the output as parameters, running the subsets on separate nodes, and joining the imagery at the end of the process. This type of parallelism is typically not as efficient as an algorithm that is rewritten with MPI, but the parallel algorithm can be implemented in a few hours.

The intent of the reprojection study was to process an image to a number of different projections with varying pixel sizes and resampling methods while creating the ability to measure and record output pixel sources. Because the same input image is used for the processing of different algorithm parameters, the input image was copied to each computing node. This is a test of the Data Resident Computing model for which the EDC cluster was designed. Thus, only the results are sent back to the master node, greatly reducing network traffic.

The theory of image segmentation for the case of map projection resampling is beyond the scope of this paper. A Perl script that performs this task was constructed; its output is a C shell script that provides “process-parallel” job control. A partial example of the produced script follows:

```
#!/usr/bin/csh
printf "Start time: "

printf "resampling 00goode25...\n"
rsh -n node0  "/home/test/bin/resample  /data1/inputImg  /home/test/00goode25
     /data1/histo 0 o > /dev/null"&
printf "resampling 01goode25...\n"
rsh -n node1  "/home/test/bin/resample  /data1/inputImg  /home/test/01goode25
     /data1/histo 0 o > /dev/null"&

[similar for additional nodes…]

rsh -n node11  "/home/test/bin/resample  /data1/inputImg  /home/test/11goode25
     /data1/histo 0 o > /dev/null"&
printf "Waiting for processes to complete..."
wait

/bin/butter goode25.part goode25 255
printf "End time: "
```

For a 25-km output pixel size and the Goode’s Interrupted Homolosine projection, a run time of just under 12 minutes was observed. Although this speedup is not impressive, the amount of time needed to parallelize this algorithm was minimal. Speedups for datasets with pixel sizes closer to 1 km are expected to be more significant.

This part of the project also considered graphical user interfaces (GUIs) for the cluster. In this case, a GUI based on the Tool Command Language/Toolkit (Tcl/Tk) was used. This scripting language-based GUI again added very little effort to the project and resulted in a good-looking, easy to use product (see fig. 9). A tutorial on adding Tcl/Tk GUIs to C or C++ programs is presented in appendix A.

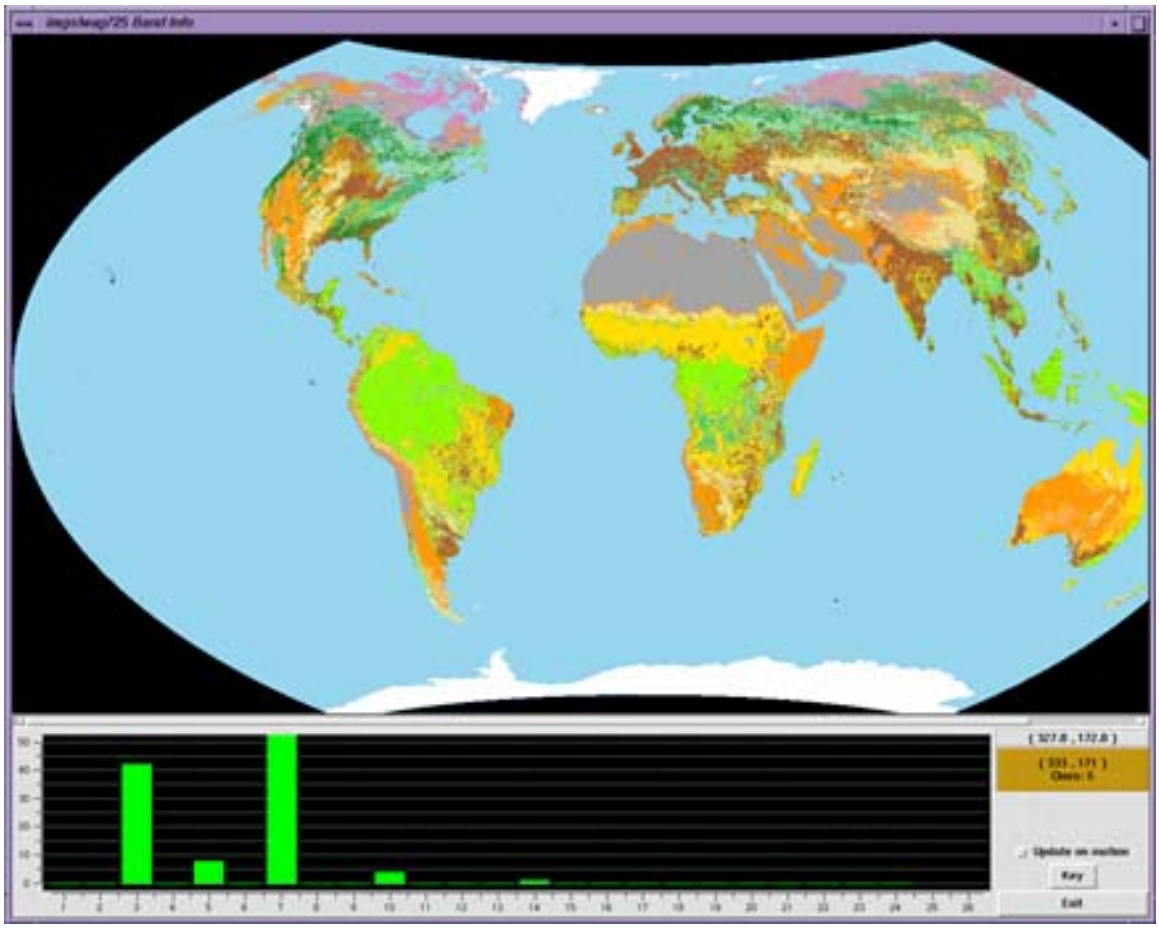

Figure 9. Example of Tcl/Tk-based GUI on the EDC Cluster.

Beowulf Tile-Based Image Resampler

The goal of implementing an image resampler for a Beowulf cluster was to determine whether a significant performance improvement could be achieved over a similar algorithm that runs on a single-processor computer. The most significant obstacle to implementing a resampler on a cluster is the limited network bandwidth between the processors of the cluster. It was unknown whether commodity networking hardware (in this case, 100-Mbps Ethernet) and standard cluster software tools, such as MPI, could be used to speed up the processing.

The Beowulf Image Resampler is based on previous implementations of image resamplers used at the EDC. Previous resampler implementations were optimized to efficiently use a machine with shared memory architecture and several processors. Shared memory architecture provides memory bandwidth between the processors that is several orders of magnitude greater than an Ethernet network. Previous resampler implementations were not designed to account for potential bandwidth limitations. Instead, the hard drive I/O bandwidth

and processor speed were the limiting factors of the design. It is possible to design software that works well on both shared memory architecture and a cluster if it is designed with bandwidth limitations in mind.

This algorithm discussion only covers the parallel implementation of the program: it does not describe the serial version of the program or any algorithms of the program except where it affects the parallel implementation. It was assumed that MPI would be used for the parallel implementation of the resampler. MPI has worked well for other applications on the cluster, and it seemed a natural fit for this program as well. The EDC Land Analysis System (LAS) package is needed to build grid model files and weight tables for the table-based resampling method.

It is assumed that the resampler will be running in a bandwidth-limited environment.

The LAM/MPI implementation was used for developing the resampler. It should work with other implementations of MPI but has not been tested with any other implementations.

The LAM team also provides a tool called XMPI. XMPI is used to monitor the message passing that takes place between the nodes in the cluster. XMPI will also collect traces of the communication activity between the nodes. Although the XMPI documentation needs improvement, once the output is understood, the XMPI trace capability has proved a very valuable tool for analyzing where communication between the master and slave nodes can be improved.

To make the source code easy to maintain and test, the developer's goal was to keep the serial and parallel code versions maintained in the same set of source code. This was accomplished by isolating the code that differs between the versions to a pair of modules.

In this application, one of the nodes acts as the master and the others act as slaves. The role of the master node is to assign work to the slave nodes and collect the results from each slave. The master node also acts as the image server in the current design. The slave nodes accept work from the master node, perform the image resampling, and return the results to the master node.

One of the key issues to parallelizing a program is determining how the work that needs to be completed can be efficiently split up between different CPUs (in this case, nodes in the cluster). To achieve the greatest speedup possible, the work packets chosen should be as independent from other work packets as possible. This reduces the communication overhead between the nodes and lost time waiting at synchronization points between the nodes.

The granularity of the work is also important. If each work packet takes a long time to complete on a node, it can adversely affect the overall run time of the

algorithm. For an extreme example, assume that a given job takes 5 hours to run in a serial implementation. Running an ideal parallel implementation of that job on a four-node system could reduce the run time to 1.25 hours. However, if that job is split into five work packets that each take an hour to run, the job will take 2 hours to complete. Obviously, the granularity of the job size is too large to make efficient use of the system.

For the image resampler, the work is divided between the slaves by assigning tiles in the output image to each slave node. Each output tile is independent of other output tiles, and the size of the output tile can be easily adjusted.

The effective use of the network band can sometimes be a critical factor in the performance of an application on a Beowulf cluster. The nodes in the EDC cluster are networked together on a single subnet using a standard 100-Mbps Ethernet. This translates into a theoretical maximum bandwidth of approximately 11 MB/s for a single Ethernet connection (11 MB/s in both directions if the network is configured in full duplex mode). This can quickly become a limiting factor on the performance of an application if a large amount of data needs to be passed over the network. This proves to be true for an image resampling application.

Consider this example. If a 100-MB input image is to be resampled to a different projection, that means at least 100-MB of data need to be transmitted from the node, with the input image to the nodes doing the resampling work. Assuming 100 percent of the network bandwidth can be used for transmitting the input image data to the slave nodes means that it will take approximately 9 seconds to send the data to the slave nodes (100/11 is approximately 9). This establishes the lower limit on the time it will take the image resampler to complete its work. There is no way that a 100-MB input image can be resampled in less than 9 seconds, regardless of the number of nodes present in the cluster, using a standard 100-Mbps Ethernet.

The example above is an ideal case. In most cases, there is rotation or warping between the input image and output image. Consider the case where the output image is a 45-degree rotation of the input image. Creating a 100x100 pixel tile of the output image, will require a 141x141 block of the input image. So instead of transmitting 10,000 bytes of the input image, you need to transmit 19,881 bytes. Extrapolating this to the entire image (ignoring that the corners of the output image will be fill and not need any of the input image), nearly twice as much data will be transmitted, and the lower limit on the time it will take to resample the image will have grown to approximately 18 seconds.

Returning the output image tiles from the slave nodes to the master node encounters the same limitations. However, since our network operates in full duplex mode, its bandwidth requirements are independent from the input image requirements. One option for making the input image available to all the nodes is

to place the image data on a disk that is NFS mounted to all the nodes in the cluster. However, NFS access is not a good method for this application. Accessing data on an NFS mount transfers data in blocks from the image file. The size of the data block is configurable, with block sizes typically ranging from 1-KB to 8-KB blocks. Although this is a good idea when complete files need to be read, it is inefficient for an application that is sparsely reading tiles from an image. For example, if the NFS mount is configured to transfer data in 1-KB blocks and a node needs to read a 100x100 tile from an image that is 8,000x8,000, the result will be 100 lines of the image read in 1-KB blocks for a total of 100 KB of data being sent. This increases the lower limit on the time to 90 seconds instead of 9 seconds.

This simple algorithm ignores the effect of disk caching. A second run of the application with the same input image would most likely be much faster if the slave nodes had a disk cache large enough to hold the entire image.

In an MPI application, the cluster is initialized with a pair of MPI API calls. The first is the MPI_Init call. This routine initializes the MPI environment and must be the first MPI routine called by an application. The second MPI call is the MPI_Comm_rank call. This call returns a number indicating which instance of the process this is. A zero indicates that this instance is the master node. Any other number indicates that the instance is running on a slave node.

Master Node Process Flow

The first thing the master node does is call the MPI_Comm_size routine to determine the number of slave nodes that are available to perform work. It then allocates memory for any communication buffers it will need based on the number of slave nodes. From there, it drops into a loop over all the image bands requested by the user. In that loop, it initializes the image I/O routines for the current band and calls a routine to assign and collect work from the slaves for the current band.

MPI provides two communication modes for sending and receiving messages: blocking and nonblocking. When the blocking mode is used, a call to send or receive a message will not return until a message has been completely sent or completely received. The standard MPI calls to do blocking communication are MPI_Send and MPI_Recv. When the nonblocking mode is used, a call to send or receive a message schedules the operation to occur and returns immediately. A handle to the communication request is returned at the same time. This handle needs to be retained since it is later used to confirm that the communication has been completed. When nonblocking communication is used, the buffer of data can't be modified until the send or receive operation is completed. So using the nonblocking mode requires the program to maintain a buffer for each outstanding operation. The set of functions needed to use the nonblocking mode is larger than the blocking mode since a function must be

called to both start and complete a message. Common functions for this API include MPI_Isend, MPI_Irecv, MPI_Wait, and MPI_Waitsome.

The master node implementation for the resampler originally used the blocking mode of communication. After the original version was functional, some performance testing was done and communication traces were gathered using XMPI. The traces showed that the master node was spending a lot of time waiting in the blocking calls. The master node also waited while the slave nodes were waiting to send results back to the master node, and the full Ethernet bandwidth was not being utilized. Modifying the master node to use nonblocking communication eliminated the time spent waiting for communication to complete and doubled the throughput of the application. It also allowed the Ethernet bandwidth to be fully utilized.

Initially, the master node sends each of the slave nodes a packet of work to do. This packet includes the output image tile location assigned and the corresponding location of the tile in the input image. The work is sent using the MPI_Isend routine, specifying the particular slave node to which the work should be assigned. The master node keeps track of the output tile assigned to each slave node so it knows what output tile the slave node is sending back when the work is complete.

Immediately following the packet containing the location of the tile in the output and input images, another packet is sent that contains the actual image data for the input image tile. This packet is sent separately from the first since the slave node needs the first packet to determine how large the buffer needs to be for receiving the input image data. The input image data are also sent using MPI_Isend.

The request handle for each of these messages is kept so that the communication can be closed later.

After the initial assignment of work, the master node calls the nonblocking MPI_Irecv to prepare for each slave to return the output tile assigned. After all the slave nodes have been assigned work (or there is no more work to assign), the master node waits for any of the slaves to return the results. This waiting period is implemented by calling the MPI_Waitsome routine. MPI_Waitsome accepts an array of outstanding request handles and returns when at least one of the requests has been completed. The resampler passes an array of the outstanding requests initiated with MPI_Irecv. One special feature of MPI_Waitsome is that it will return a list of all the requests that have been completed. Initially, MPI_Waitany was used to wait for a single completion. It was found that MPI_Waitany was very biased to return the lower numbered slaves if they had completed requests. In the bandwidth-limited scenario of the resampler, this was leading to some of the slaves being starved – a condition in which they are waiting for work assignments.

When results are received, the master node determines which output tile has been received by checking which output tile the sending slave was assigned. The output tile is then written to the output image in the correct location. After the tile is written, the MPI_Wait routine is used to complete the original requests for assigning the tile to the slave node and sending the input image data to the slave. Both of these requests are guaranteed to be completed because the resulting output tile was already received.

After the output tile has been written, another output tile is assigned to the slave.

If the MPI_Waitsome call reports more than one of the output tiles to be complete, the additional output packets are dealt with in the same manner before MPI_Waitsome is called again.

After the master node has sent the last output tile for the current band to a slave node, it continues collecting output tiles from the slave nodes until all the output tiles for the band have been returned. If there are more bands to resample, the rest of the bands go through the same process.

When all the bands have been completed, the master node sends a message to all the slave nodes to tell them it is time to terminate. All the nodes call the MPI_Finalize to shut down the MPI environment and then exit.

Slave Node Process Flow

The slave nodes initialize the MPI environment by calling MPI_Init and MPI_Comm_rank to learn their roles in the cluster, much like the master node. The slave nodes know they are slaves since the return value from MPI_Comm_rank is nonzero for a slave.

Each slave node allocates a buffer for its output tile and immediately calls the warp function. The warp function is the same function that performs image resampling for the serial version of the resampler.

For the slave nodes to function in the cluster, two routines are replaced from the serial version: the get_tiles and complete_output_tile functions. The get_tiles function returns the output tile location, input tile location, and input image tile for the next area to resample. In the MPI version of the resampler, this function calls MPI_Recv twice: first, to receive the location of the output and input tiles; second, to receive the actual input image tile data. The slave nodes use the blocking MPI calls. The slave nodes could be modified to use the nonblocking MPI calls. However, for this to make a difference, the master node implementation would need to increase in complexity to have multiple tiles assigned to each slave node. It has been determined that the table-based resampling method is sometimes processor limited and not bandwidth limited. It has also been determined that running multiple slave processes on each node

can gain the same benefits as adding the complexity of having the slaves use nonblocking MPI communication calls.

To easily support 8-bit, 16-bit, 32-bit, and floating-point resampling, the input tile data are always converted to floating-point data. The data are not converted to floating point until they reach the slave node, to save network bandwidth. If the input image is 8-bit data, converting it to 32-bit floating point at the master node would result in performance being decreased by a factor of 4, since 4 times as much data would be sent over the network.

After the slave node resamples the assigned tile, it calls complete_output_tile. Complete_output_tile converts the output image data to the user-requested output data type and sends it back to the master node using MPI_Send.

Currently, the resampler requires a common NFS-mounted drive that is visible to all the nodes. There are three reasons for this:
(1) The program needs to be available to all the nodes.
(2) Any associated model file needs to be readable by all nodes.
(3) The input image .geoinfo and .proj files need to be readable by all nodes.

The second two requirements could be eliminated by having the master node read these data and send them to the slave nodes, but no noticeable performance increase would be obtained.

Improvements

Improvements that could be made to the Beowulf resampler implementation include the following:

- Remove the requirement for the geoinfo and .proj files to be available through an NFS mount to the slave nodes.

- Only read the command line parameters on the master node and send the relevant parameters to the slave nodes by means of an MPI message.

- Modify the error-reporting mechanism to have the slave nodes send error messages to the master node instead of sending them to standard out.

- Add fault tolerance to allow the master node to send work to a different slave node when the results are not returned at the end of the run.

- Add support for allowing a different node to serve the image to the slave nodes.

- Complete the pixel-by-pixel model.

The Beowulf resampler was run many times using different resampling methods and numbers of slave nodes. The resulting execution times were recorded for performance comparisons.

Several identical runs of the resampler were done. Any timing results that differed significantly from the nominal time were eliminated from the results; this was to eliminate the impact of other processes running on the system and disk cache from causing variation in the results. Typically, this resulted in the first run being eliminated, since it loads the input file into the disk cache. Basically, this was done so that the run times would be testing the parallel speedup and not the time used to read a file from disk. The results are shown in figure 10.

All testing was done with executables built with the GNU gcc compiler using -O optimizations.

It is worth noting for the results that the master node of the cluster is a dual-processor machine, which affects the results when more processes are started than there are nodes in the cluster.

Processes	Time (in seconds)	Speedup
1 (Serial)	42.3	N/A
2	1838.2	0.023
3	62.5	0.67
4	31.5	1.34
5	21.9	1.93
6	16.8	2.52
7	13.8	3.07
8	12.2	3.47
9	12.0	3.53
10	11.6	3.65
11	10.8	3.92
12	10.8	3.92
3	10.9	3.88
14	9.6	4.41
15	9.95	4.25
26	13.8	3.07

Figure 10. Grid model and bilinear resampling with a 13-node cluster.

Several observations can be made from the data displayed in figure 10:
- Bilinear resampling is not very CPU intensive.
- Network latency has a large negative impact when a small number of nodes are applied to the problem.
- Bilinear resampling quickly becomes bandwidth limited since the maximum speedup obtained is roughly four times the serial implementation.
- It appears that, with a large number of processes, the setup time can make the overall performance worse when bandwidth is the limiting factor.

Some additional tests were run without compiler optimization turned on. In these tests, the quickest run times were limited to about 11 seconds. The speedup was larger since the nonoptimized serial version took nearly 60 seconds to complete. This also indicates that the application was bandwidth limited.

Processes	Time (in seconds)	Speedup
Serial	394.8	N/A
7	73.3	5.39
10	47.1	8.38
13	35.0	11.28
24	31.6	12.49
26	31.0	12.74

Figure 11. Grid model and weight-table-based resampling with a 13-node cluster.

Observations that can be made from the data in figure 11 include:
- Weight-table resampling is CPU intensive.
- Weight-table resampling is not bandwidth limited since it attains a nearly linear speedup on the cluster.

Processes	Time (in seconds)	Speedup
Serial	30.4	N/A
3	38.9	0.78
4	20.5	1.48
6	11.0	2.76
9	6.8	4.47
13	5.7	5.33

Figure 12. Polynomial model and nearest neighbor resampling with a 13-node cluster.

Figure 12 above shows the general trend of a bandwidth-limited job.

Processes	Time (in seconds)	Speedup
Serial	30.4	N/A
3	20.9	1.45
6	12.0	2.53
9	10.2	2.98

Figure 13. Polynomial model and nearest neighbor resampling with a three-node cluster.

A near-linear speedup is shown in figure 13 for nearest neighbor resampling. Note that the master node has two processors and is causing a slightly better performance than expected for the nine-process case.

Processes	Time (in seconds)	Speedup
Serial	60	N/A
3	40.2	1.49
6	22.4	2.68
9	18.9	3.17

Figure 14. Grid model and cubic convolution resampling with a three-node cluster.

Figure 14 also shows a near linear speedup for cubic convolution resampling. Note that the master node has two processors and is causing a little better performance than expected for the nine-process case.

Using an NFS mount instead of MPI

For comparison purposes, a modified version of the resampler was generated that used an NFS mount for reading the input image tiles instead of sending the input image tiles through MPI. Figure 15 presents the results for a grid model and bilinear resampling on a 13-node cluster:

First Run (in seconds)	Second Run (in seconds)
57.7	10.9
47.5	11.3
46.4	11.2

Figure 15. Grid model and bilinear resampling on a 13-node cluster.

The first run is when the disk cache on the slave nodes does not contain the input image. The second run immediately follows the first. This shows that an NFS mount is a poor choice for a single run of an image. However, if the same image is used multiple times and it fits in the disk cache of the slave nodes, an NFS mount can provide performance comparable to the MPI implementation.

Image Processing with the Land Analysis System on the Cluster

The Land Analysis System (LAS) is fully implemented on the EDC Beowulf cluster. The LAS software provides a complete image processing environment for remote sensing data, including a full-featured geometric registration system, Advanced Very High-Resolution Radiometer (AVHRR) radiometric and geometric processing, image enhancement software, and a batch processing/scripting language. The package also contains X-windows-based large-image display functionality and the ability to select image control and tie points from a variety of sources. The LAS software is required to create input datasets for the Beowulf resampler.

Implementation of the CENTURY model on the EDC Beowulf Cluster

Local-scale processes, such as land use and land cover change, are closely

related to phenomena such as global climate change with regional to global consequences. Simulation of the impact of human activities on biogeochemical processes on the Earth's surface, with the capability of incorporating local variability of state and driving variables, relies heavily on computing power. Owing to the lack of adequate algorithms for assimilating detailed local data and computing power, the simulated interactions between the biosphere and the atmosphere are most likely biased (Reiners et al., 2001). Liu and others (1999; 2000), and Reiners et al. (2001) have developed a stochastically based ensemble simulation system to simulate nitrous oxide (an important greenhouse gas that contributes to global warming) emissions from soils into the atmosphere, with applications in Costa Rica. At present, a generic model for simulating carbon dynamics in terrestrial ecosystems over large areas (COLA) is being developed using the ensemble approach (Liu et al., 2001). Our final goal is to apply COLA to simulate the spatial and temporal changes of CO_2 exchange between the terrestrial biosphere and the atmosphere over large areas, such as the conterminous United States.

The underlying ecosystem model of COLA is CENTURY, which is a well established model for simulating biogeochemical cycles in various ecosystems, including crops, pastures, forests, and savannas, at various spatial and temporal scales (Parton et al., 1987). Although a predecessor of COLA has been implemented successfully on the Unix Sun workstation, it took 50 hours to finish one run for an area of 0.4 million hectares in Costa Rica. Obviously, it is necessary to speed up the simulation if COLA is to be applied to an even larger area, such as the conterminous United States. To test the feasibility and efficiency of implementation of the COLA on the EDC Beowulf cluster system, we installed a prototype (for the prediction of nitrous oxide emissions from soils in Costa Rica) in 2001. Results indicated that the Costa Rican simulations could be done in about 2.5 hours using the parallel processing of the Beowulf system: a reduction of 95 percent of the time required with the Sun workstation version.

We would like to implement a full version of COLA on the EDC Beowulf cluster to simulate the impact of land use and land cover change on the carbon cycle in the conterminous United States for the past 200 years. The preliminary results obtained have two implications. First, more complex model simulations could be performed by installing the model on parallel systems to incorporate fine-scale information, which has previously been a daunting task for earth scientists. This could lead to important findings on the optimization of up scaling algorithms of various processes and phenomena in the earth sciences. Second, fast and detailed simulations of the interactions between the biosphere and the atmosphere are necessary for climate and weather prediction. The successful implementation of COLA will have implications for developing similar systems to facilitate and improve additional detailed coupling of Earth surface models and climate models in general.

The Linux Computer Lab as a Beowulf Cluster

During the summer of 2001, a Linux cluster was constructed at SDSU in Brookings, SD, as part of the research conducted in the EDC Beowulf project. The Linux operating system is used in many courses taught at SDSU; the university has computer labs that contain Linux-only machines. The goal of this investigation was to see if a computer lab could function in a dual-purpose mode: as stand-alone student workstations and, at the same time, as a Beowulf cluster.

Approximately 2 weeks were required to get the cluster running. Fourteen Pentium II 500-Mhz machines, each with 128 MB of RAM, were used to build the cluster. It was built following the model of the cluster at the EDC. The machines can boot one of two operating systems: Red Hat Linux 6.2 running the 2.2.14 kernel and Linux Mandrake 8.0 running the 2.4.3-20 kernel. At summer's end, the cluster was redone so that it is now a mixed student lab and cluster computer. The students can sit at any node in the cluster and do their classroom assignments while parallel jobs run on the cluster in the background. This required making the network public, as opposed to the private networks used in Beowulf clusters.

MID-CONTINENT MAPPING CENTER

As the new millennium dawns, we are witnessing an explosion in the use of geospatial data. Not only are more people interested in using the information, but new technologies and research are producing more and better information than was previously available. NMD has also embarked on a new program, called the National Map, that promises to produce and make available even more geospatial information.

Greater amounts of information also require greater computing power to process. Datasets that were previously measured in megabytes are now measured in gigabytes. There is a need to find ways to process these data quickly, as they are being used increasingly in hazards mitigation and emergency response. Unfortunately, owing to the enormous quantity of data, desktop machines can no longer provide the capability to process this information, either in a time-critical manner or at all.

Distributed processing systems can provide a solution to this processing requirement. Unfortunately, most of the computer science research in distributed processing has been directed at numerically intensive processing, not at databound applications. Much is known about how to design algorithms for distribution, but less is known about how to design a system that can effectively process large amounts of data.

To expand the realm of knowledge in this area, MCMC scientists have focused their research on methods to efficiently process large amounts of data in a distributed manner. This work is a continuation of an FY 2000 Center-funded

research project that involved finding ways to quickly process gigabyte sized datasets. Results from the FY 2000 project guided MCMC researchers in the selection of concepts and theories to study in FY 2001. MCMC researchers focused on several methods that could be used to process data quickly and efficiently. The first was the use of a distributed file system, where parts of a file were spread across several nodes. The second was to try a new networking technology that could give a machine the ability to communicate with multiple machines at the same time. The third method was a theory developed at MCMC for using processing partitions to divide the work among the machines.

The Beowulf cluster at the MCMC is housed in the Geographic and Cartographic Research and Applications lab. This lab is attached to the computer center lab at MCMC and is environmentally controlled. The cluster was upgraded in FY 2001 to support research for this prospectus project. As it currently stands, the cluster comprises 16 upgraded machines and 2 dual-processor Dell workstations (see fig. 16 below).

Figure 16. At 18 nodes, the MCMC system is the largest of the Beowulf clusters.

The 16 upgraded machines were constructed during an FY 2000 project at MCMC. They consist of various machines that were put on surplus by the Center and upgraded for the project. Each of these machines uses a VIA-503+ Super

Socket 7 motherboard with AMD K6-2 550-MHz processors. This configuration was chosen because it could fit into the Advanced Technology form factor cases of the surplus machines. Each of these nodes also has 256 MB of SDRAM physical memory installed. Tekram DC-390U2W SCSI controllers are installed in each machine, and each node has a single 9-GB IBM U2W SCSI hard drive. Each machine has a generic ISA video card that is only used for major system upgrades or when problems arise and the node needs to be pulled. The rest of the parts, such as floppy drives and CD-ROMs, are parts that were reused from the original equipment.

A Dell workstation was purchased in FY 2000 as the original head node. This machine is a dual 733-MHz Pentium III with 256 MB of RAMBUS memory. An Adaptec Ultra160 SCSI adapter is installed and connected to two 18-GB SCSI drives. The machine includes an IDE ATAPI CD-ROM drive. The machine also has an nVidia GeForce 2 GTS video card that is only used for upgrades or system problems.

The current head node was purchased in FY 2001. It is a dual 800-MHz Pentium III Dell workstation that is outfitted with 512 MB of RAMBUS memory. This node also has an Adaptec Ultra160 SCSI controller with two 18-GB SCSI drives. It has an IDE ATAPI DVD drive so that it has the ability to load data that may be stored on DVDs. An IDE CD writer is included so that data can be transferred from the cluster to an outside machine, as the cluster has no direct connections to the MCMC network. It has an nVidia GeForce 2 GTS video card that is connected to a 21-inch flat screen monitor for output. The nVidia card was chosen because nVidia provides Linux drivers that allow hardware-accelerated three-dimensional video displays. In fact, the nVidia Linux drivers are taken from the same source code base as the Windows drivers, making nVidia one of the few video cards that can perform as well under Linux as it can under Windows.

Each of the nodes in the MCMC cluster is running the Red Hat 7.1 distribution, with current updates applied. The Linux 2.4.10 kernel has also been installed on each machine, with the stock Red Hat version removed. The networking has been hand modified to route on an FNN configuration. The cluster nodes may be upgraded to the Alan Cox experimental kernel series in the near future.

The MCMC cluster is on a self-contained network, meaning that it has no connections to the outside world. This was done both for security purposes and for network testing, because the network needs to be "clean" in order to obtain accurate data during testing. In the current flat network neighborhood configuration, the cluster is connected through three Netgear FS516 fast Ethernet switches. The FS516 provides either sixteen 100-mBit capable connections or fifteen connections with an uplink port. They are used in the cluster in their 16-port configuration. Each node has two 3Com 100-mBit Ethernet cards installed for the flat network neighborhood configuration. Power is provided to the cluster by means of three APC Smart-UPS 2200XL

uninterruptible power supplies (UPS). These units each have eight power connectors in the back and are connected to the Center's power supply by heavy- duty cables. The units power all of the equipment in the cluster, including the nodes themselves, the switches, and the monitor.

Parallel Virtual File System

The Parallel Virtual File System (PVFS) was chosen for researching distributed file systems. This technology was created specifically to resolve the problem of data starvation in a distributed processing environment. In a distributed environment, this starvation may become severe as many processing nodes are all trying to get data at the same time.

PVFS is designed to address these concerns by providing parallel input/output (I/O) to the cluster. This parallel I/O increases data bandwidth on the cluster by using multiple machines to create a large virtual disk drive. When files are written to this virtual drive, they are broken into multiple parts that are stored on the various file-serving nodes. The advantage of this setup is that instead of every machine in a cluster trying to get data from a single file server, multiple machines are simultaneously serving different parts of a file to different processing nodes. This increases the overall data throughput on the cluster, as every processing node does not have to wait in line to communicate with a single file-serving node. Instead, there are multiple data streams that can be active at any given time. These multiple data streams equate to much more data being accessed at any given moment.

PVFS operates by means of a metadata manager and various operating system modifications. The operating system modifications allow some of the standard Unix utilities, such as cp and ls, to function correctly. The package also provides a library so that applications can bypass the operating system to communicate directly with the PVFS system. To request a file, a client first contacts the metadata manager. This manager is responsible for telling the client which file server nodes contain the parts of the file that were requested. The client then contacts the file server nodes and requests the various parts of the file. Figure 17 from the PVFS Web site illustrates the logical design of this system.

This system can be compared with the more traditional method of file serving that involves a single server. In this scheme, every machine on the network must contact the single server when an application requires some data from permanent storage. The problems in a distributed processing system begin here, as each node must contend for a network connection to this single server. This

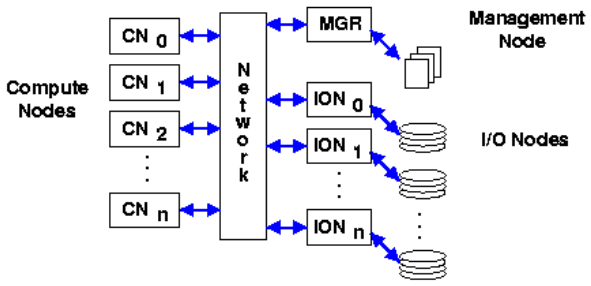

Figure 17. PVFS logical system design (from the PVFS Web site).

contention can cause the nodes to wait for a period of time, and they must keep retrying until they get connected. Once connected, the single server can then transmit only as much information during a given time period as its networking hardware allows.

Flat Network Neighborhood

The Flat Network Neighborhood (FNN) is a new networking topology that was developed at the University of Kentucky and the Australian National University at Canberra's Bunyip cluster. This topology, or connection arrangement, is designed to maximize bandwidth between large numbers of nodes. In a traditional setup, machines in a distributed processing cluster are connected through a single Ethernet card to a switch. The switch is a piece of hardware that creates connections between machines so that they can communicate with each other. Switches have a limited number of physical connections and are typically connected to other switches by an uplink port to create a larger network of machines. The problem with this uplink port is that all machines on one switch must communicate with machines on another switch through this single connection. This can cause a significant amount of network congestion as machines must wait in line to use the uplink port to connect to the other switch.

FNN setups have several characteristics that distinguish them from other network setups. First, nodes in an FNN system have multiple Ethernet cards in each machine connected to multiple switches. The interfaces in each machine have different Internet Protocol (IP) addresses. Instead of connecting switches via the uplink ports, the FNN has network connections arranged so that each machine can communicate with any other machine on the network by going through exactly one switch. To accomplish this, the network routing on each machine is set up so that different machines can "know" a given machine by different IP addresses. This differs greatly from a conventional network, where each machine is known by a single IP address. The routing in each machine in an FNN is configured so that communication is sent through the appropriate Ethernet interface card. Figure 18 from the FNN Web site illustrates how machines are connected.

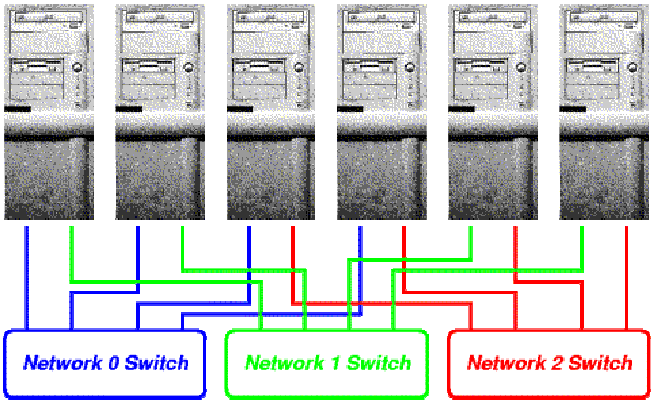

Figure 18. Diagram of a simple FNN (from FNN Web site).

The FNN architecture, although similar in some ways to channel bonding, is a different concept. Channel bonding is a method of increasing bandwidth between machines by using two network cards at the same time. Channel bonding treats both interfaces as one virtual interface that has almost double the network bandwidth that a single interface can provide. While channel bonding can only communicate with one machine at a time, FNN setups allow a machine to communicate with multiple machines simultaneously. This can be advantageous in the case of file servers, because it can help to increase the amount of data that can be simultaneously transmitted to different machines. FNN systems provide a minimum of latency between machines by bypassing the use of uplink ports. This can be important to clusters with a large number of machines, because switches have only a finite number of ports that can be used to connect nodes. FNN setups also seek to provide a higher bisection bandwidth than other methods, where bisection bandwidth is the product of the individual link bandwidths and the number of links cut when the network is cut in half. This measurement can provide information on the worst-case workings of the network topology.

Processing Partitioning Method

The Processing Partitioning Method (PPM) was developed at MCMC in late 2000 as a possible means to increase network throughput. Some background is necessary to understand this method. Parallel-processing time compared with that of a single-threaded system takes the common form of $T = \frac{T_1}{P_{eff}} + c$, where T is the parallel-processing time for a given task, T_1 is the single-threaded time for the task, P_{eff} is some coefficient of parallelism, and c is an overhead constant (Salmon et al., 2001). In distributed systems, c represents the overhead time from tasks, such as the time it takes to acquire a connection to another machine, the time it takes to transmit data, the time it takes for the operating system to move data from permanent storage to the networking system, and so on. Some of these tasks, such as data transmission time, are constants themselves, being limited by the maximum speed of the networking hardware installed in the

system. Others, such as the time it takes to acquire a connection to another machine, can be minimized by reducing the number of machines trying to make the same connection.

The PPM was designed to try to minimize the effects of some of these time delays by working in conjunction with the PVFS system previously described. In this setup, each of the processing nodes is grouped into *n* processing partitions. There are also *n* PVFS file-serving nodes, and each processing partition contains one of these PVFS nodes. Data to be used by the application are split evenly into file partitions across each of the *n* PVFS nodes, where each node contains the file partition of data that will be used mainly by the processing nodes in its partition. This setup is illustrated in figure 19 below.

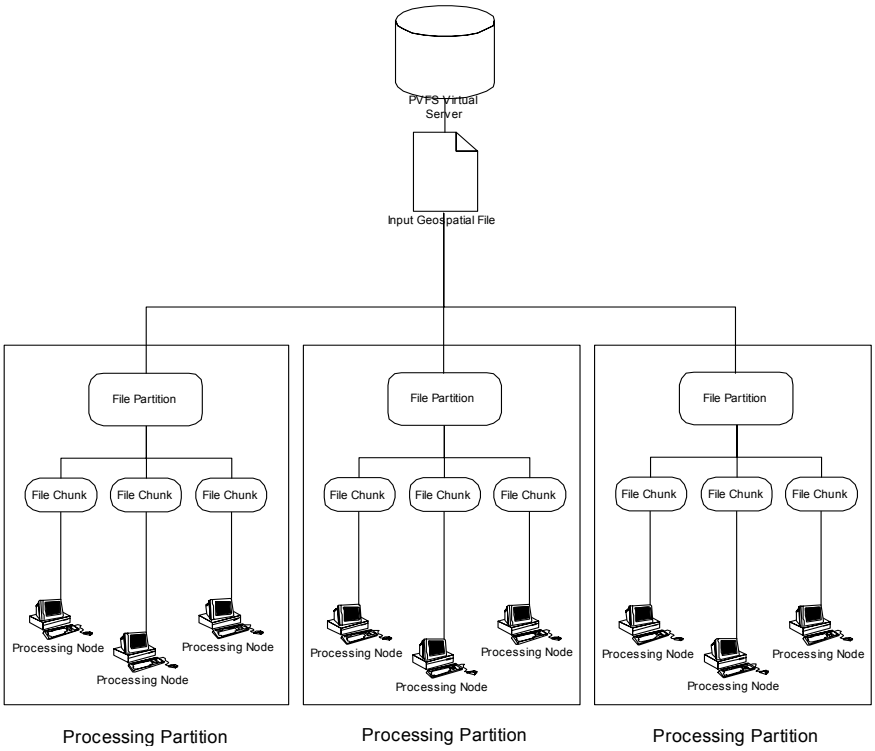

Figure 19. Processing Partitioning diagram.

When data are processed, the work is first broken into *n* pieces, one piece for each of the processing partitions. Within each processing partition, the work is further subdivided by the number of processing nodes in that partition. The last processing partition will typically have a smaller workload than the others, as the data and work may not be evenly divisible by *n*.

The setup offers several techniques to reduce the overhead constant in the parallel-processing time equation. First, as the data needed in a partition are kept in that partition's file-serving node, the number of nodes needing access to

that data is reduced. The main accesses will be from the processing nodes in the same partition, not the cluster as a whole. There can be some accesses from other partitions, but this would be mainly for data processing that is near the beginning or end of a file partition segment. This can reduce network congestion at the file server, because fewer nodes are trying to get access to the partition's server at any given time. This leads to a reduction in the time it takes for a node to acquire a network connection to the server. Second, as each partition contains a PVFS file-server node, more data can be accessed at any given time than with traditional methods. This can lead to more data throughput as at least *n* parts of the file can be accessed simultaneously.

Fiscal Year 2001 Research

At MCMC, a software system was designed and implemented during the FY 2000 research projects to perform projection system conversions on digital data. This was chosen because it met a need to find ways to quickly project large amounts of digital data quickly for disaster needs. It also directly met NMD's needs for large-scale geospatial data handling. It was an ideal application for testing, as it is extremely databound when used for gigabyte-sized datasets. This software was also used in the FY 2001 research.

This software is modified each time a new experimental method is tested. This involves designing additional functionality and testing for accuracy. For testing, digital orthophoto quarter-quadrangle (DOQQ) datasets are used that have verified control points. The input file for all of the testing in FY 2000 and FY 2001 is a merged area of these DOQQs that is roughly $1^1/_2$ GB in size. These files were merged with the GRASS GIS and then converted into GeoTIFF format.

All of the following charts represent the average test run time based on either a certain number of nodes or a certain data size. The runs were performed three times before being averaged. The software calculates the run time internally by making system calls to get the current time before processing starts and after processing is complete. The times are rounded to the nearest second for each run.

The first part of the FY 2001 research focused on continuing some of the FY 2000 research; this involved processing data in work units instead of small individual pieces. The goal of the experimentation was to see if the work units could be increased in size to improve processing efficiency without congesting the network. The is that when an operating system reads data from permanent storage, it usually caches more of the file in memory than requested in case the application needs more data. The goal is to have the application read data from the memory cache instead of always having to read it from the much slower hard drives. Sending data in larger work units may be more efficient, as data already in memory can be more quickly sent over the network. Sending larger work units also allows the processing nodes to keep more data they may need locally. The

research was also performed to see if the Linux 2.2 series kernel being used had any problems with this type of processing.

The research found that there was a limit to how much data could be sent at once without degrading performance. Past that point, the network began to suffer from too much congestion and processing began to slow. With the Linux 2.2 series kernel, this point seemed to be reached whenever each work unit comprised of approximately 5.5 MB of data. Anything past this point resulted in the projection software suffering from data starvation, as the increased network congestion slowed down data reads from the file server.

Other interesting issues were noted when the number of processing nodes and the work unit size were varied. The most interesting was that the time it took to process data actually increased past a certain number of nodes. This was observed as the software was run on a varied number of nodes in order to produce a predictive time versus the number of nodes function. Figures 20 and 21 illustrate this feature.

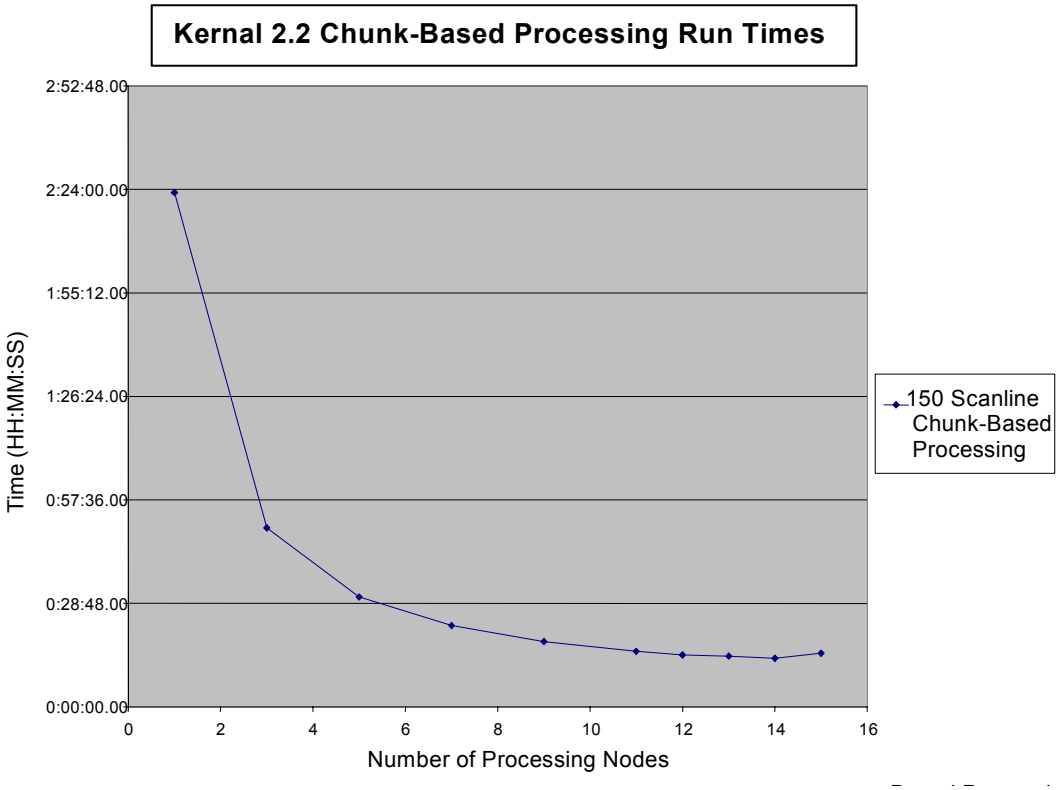

Figure 20. Chart of the minimum runtime theory.

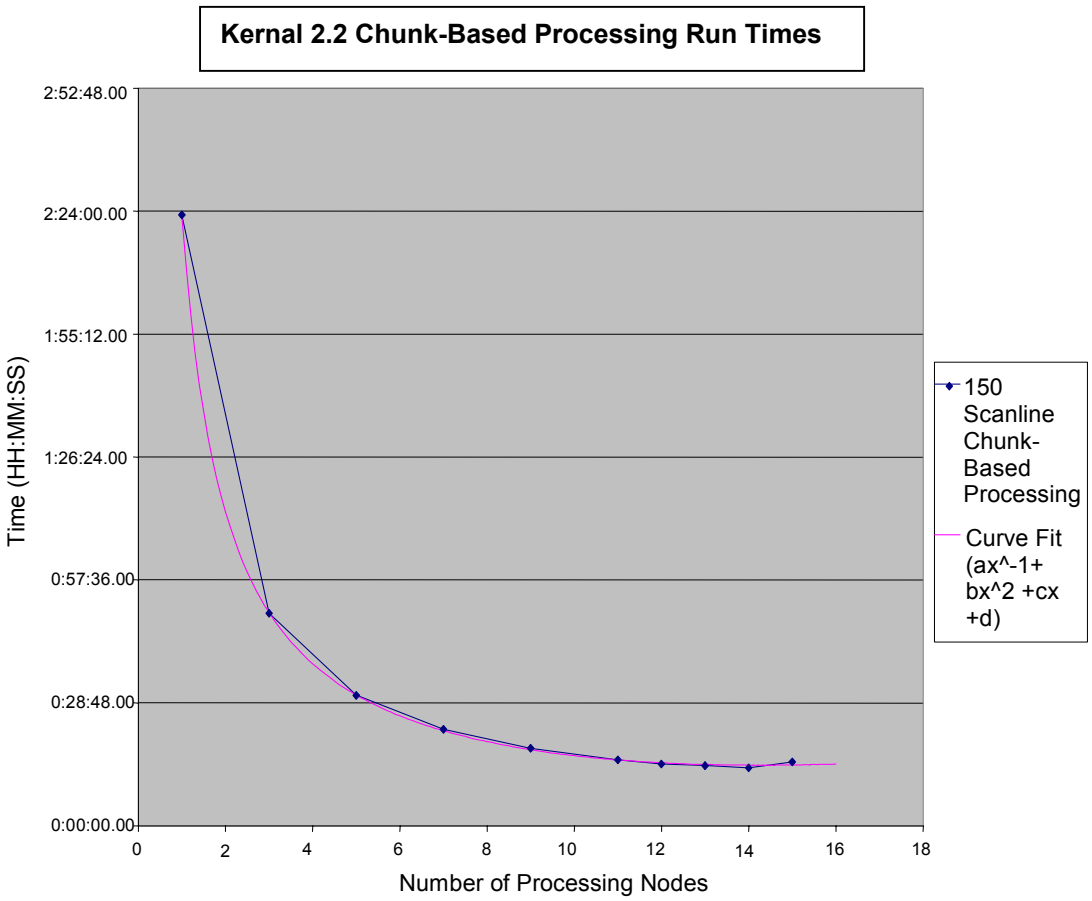

Figure 21. Minimum runtimes with curve fit.

This feature was then researched to provide an explanation. Up to a point, as the above curve illustrates, the run times decrease in a fairly predictable manner. But, once a minimum is reached, they begin to increase again. Some testing was done with network congestion, and some additional networking information was researched. An initial theory was then developed to explain these observations: Adding nodes to a distributed-processing cluster will cause databound applications to reach a minimum runtime. Additional nodes past this point will cause the runtime to increase.

To understand this theory, one must again consider the parallel-processing time equation. As previously discussed, the function has a term that is equal to the overhead in the system. As more nodes are added to a system, the coefficient of parallelism takes on larger values, making the first term of the equation smaller and smaller. But the overhead constant also takes on larger values, owing to the fact that there is more network traffic on the cluster that causes more congestion. However, this is usually insignificant when compared with the speedup from the added nodes. When a certain point is reached, the overhead portion begins to overtake the speedup gained. The overhead then takes on such large values that it begins to increase the run times. This causes the run time curve to take

on a parabolic shape, as is shown in the above figures. It is theoretically possible that this can also be the case with processor-bound applications. However, because process-bound applications transmit so little data, it would take a much larger number of processing nodes to reach a minimum value than it does with databound applications.

The first task begun in FY 2001 was to upgrade all of the nodes on the cluster to the Linux 2.4 kernel and the Red Hat 7.1 distribution. The Linux 2.4 kernel offered some new features, such as zero-copy networking and improved virtual memory performance. Red Hat 7.1 also came with a newer version of the GNU compilers that offered better C++ compatibility and performance.

The zero-copy networking feature was developed to improve network efficiency. In previous versions, when an application required sending data over a network, it had to first store the data in its own memory buffer. The operating system then had to copy this buffer into a buffer of its own before it could send it over the network. This additional copying of data added a time delay that reduced the amount of data that could be transmitted during a given time period. The zero-copy feature eliminates this extra data copy, thereby increasing network transmission efficiency.

The 2.4 kernel also offered improvements to some problems that were directly caused by the older Linux kernels. The 2.2 kernel was observed to have some problems in its virtual memory and disk cache handling. It is common for an operating system to read in more data than what is requested by an application. This can help improve I/O because any additional read requests can be done from the faster random access memory (RAM) instead of the slower hard drives. Modern operating systems also use hard drive space as additional memory by moving less often used programs and data to the hard drive to free up more room in RAM for running programs. This is known as the virtual memory mechanism. Normally, the disk cache is reduced in size when running applications need more memory. The 2.2 kernel, however, seemed to never really reduce the size of the hard drive cache. This resulted in a smaller than normal amount of memory available for running applications. The operating system responded by moving more programs in and out of virtual memory, a process that slowed down processing in some cases, because the operating system was busy moving data back and forth between the hard drive and RAM.

Once some of the existing nodes were upgraded and the 18th node was brought online, several problems occurred during cluster operation. The first significant problem was that the default SCSI driver for the Tekram DC-390 U2W controller cards used in some of the original nodes caused the operating systems to lock up. After some work, it was discovered that the default SCSI controller driver was actually the incorrect one, and another had to be used. This required hand modifying the remaining Red Hat 7.1 installs to use the correct driver. After this was done, all of the nodes functioned properly when accessing the hard drives.

The second problem was actually caused by the zero-copy networking mechanism being a victim of its own success. The zero-copy mechanism was quite successful in increasing the amount of network traffic during a given time period. The problem arose when using a single file-server node configuration. As the processing nodes could generate more traffic, they began to access the file-server node quite frequently. In fact, the increased network traffic resulted in so much network congestion at the file server that it caused a massive, distributed denial-of-service attack, much like the ones used by hackers when they want to bring down a Web server. In this case, the congestion prevented the file-server from handling requests quickly enough. The file server would either stop responding for a period of time or lock up completely. Several days of work went into diagnosing this problem.

The problem was finally narrowed down to the network card being reset. Within the network card driver, a mechanism called a watchdog timer tracks how long it takes for the machine to respond to an incoming network packet. If the response takes too much time, the watchdog timer resets the network card. This was causing either a significant wait, as it took some time to reset the card, or a complete system lockup because the watchdog timer was tuned to a normal network, not one that has a large amount of network traffic, such as a distributed-processing cluster running a databound application. The solution was to increase the amount of time that the watchdog timer waited before it tried to reset the card. An optimal time was finally found that would keep the system running smoothly.

When these problems were fixed, the timings were rerun under the new kernel to see what effects it had. The software was recompiled under the newer compiler system, because it better supported the C++ source code in which the projection software was written. This testing again consisted of running the software on a varied number of nodes to determine a predictive function. The new run times are represented in figure 22.

As can be seen, the addition of the zero-copy networking, improved virtual memory handling, and cache performance did have an effect on runtimes. The kernel improvements decreased the run time significantly in some cases. The zero-copy networking allowed more network data in a given time period, and the improved memory performances stopped the excessive virtual memory accesses that had previously had such a negative impact on performance. These results are significant, as most Beowulf clusters use the Linux operating system and care should be taken as to what kernel version these clusters are running.

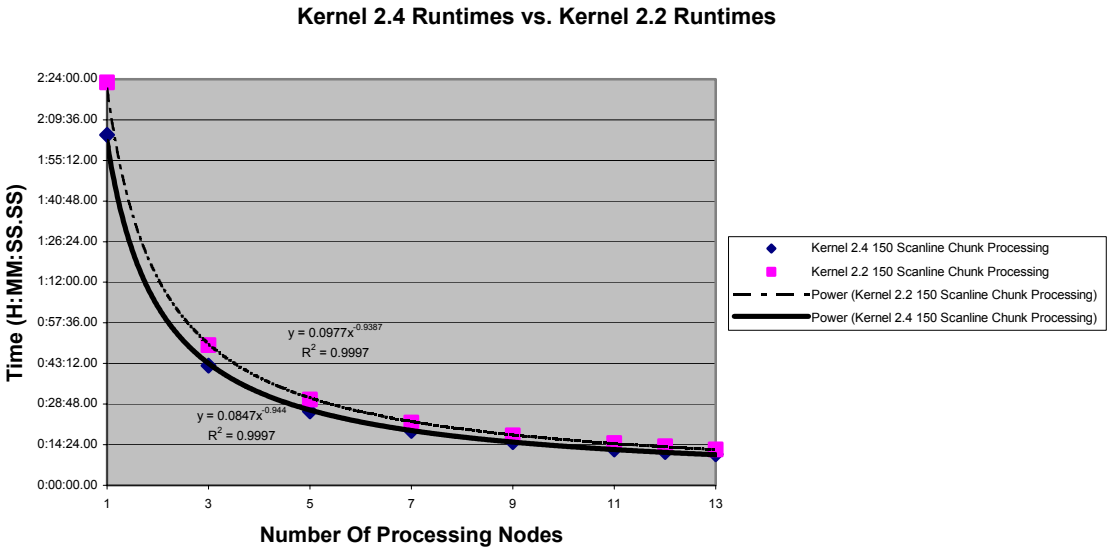

Figure 22. Runtimes – Kernel 2.4 versus 2.2.

Once the upgrades and testing were complete, moved on to PVFS setup and PPM experimentation. The initial PVFS conversion consisted of converting three of the processing nodes into I/O servers, striping the test data across all three of the servers equally, and then testing the unmodified projection software using the striped image as input. In order to correctly stripe the input data across the I/O servers, we had to modify the PVFS copy program, because it originally created zero-length files if the stripe size was set up to be too large. However, once this modification was made, the PVFS installation was completed without problems.

Initial testing using this setup produced results that were much slower than those obtained just using NFS data access. It was determined that the degradation in performance was caused by the PVFS kernel module doing unnecessary reads when asked to perform simple searches within a file. Additionally, using the kernel module requires four context switches for each I/O operation. A context switch is when the operating system has to change the processor state from running user applications to a special mode where the operating system can run its own tasks. This occurs because in order to perform an I/O operation, the client program must switch to the kernel mode and then run the PVFS daemon, and vice versa, to return the results of the operation. Both these issues caused an extreme slowdown in the processing rate. A solution to this problem was to modify the projection software, specifically the libtiff library, to use the PVFS minilibrary instead of the PVFS kernel module. This minilibrary did not suffer from the same problems because it bypasses the kernel module on file operations and has faster file position seeking.

After successfully installing PVFS, we modified the projection software to follow a PPM paradigm. This was accomplished by having the master node logically partition processing nodes by I/O servers. Every time a processing node asked for work, the master would only assign it work from that node's logical partition. An additional feature was added that allowed processing nodes of completed partitions to be added to active partitions so that no processing node could be inactive during the projection operation. Testing results showed this to be a slight improvement over the straight NFS access, as indicated in figure 23.

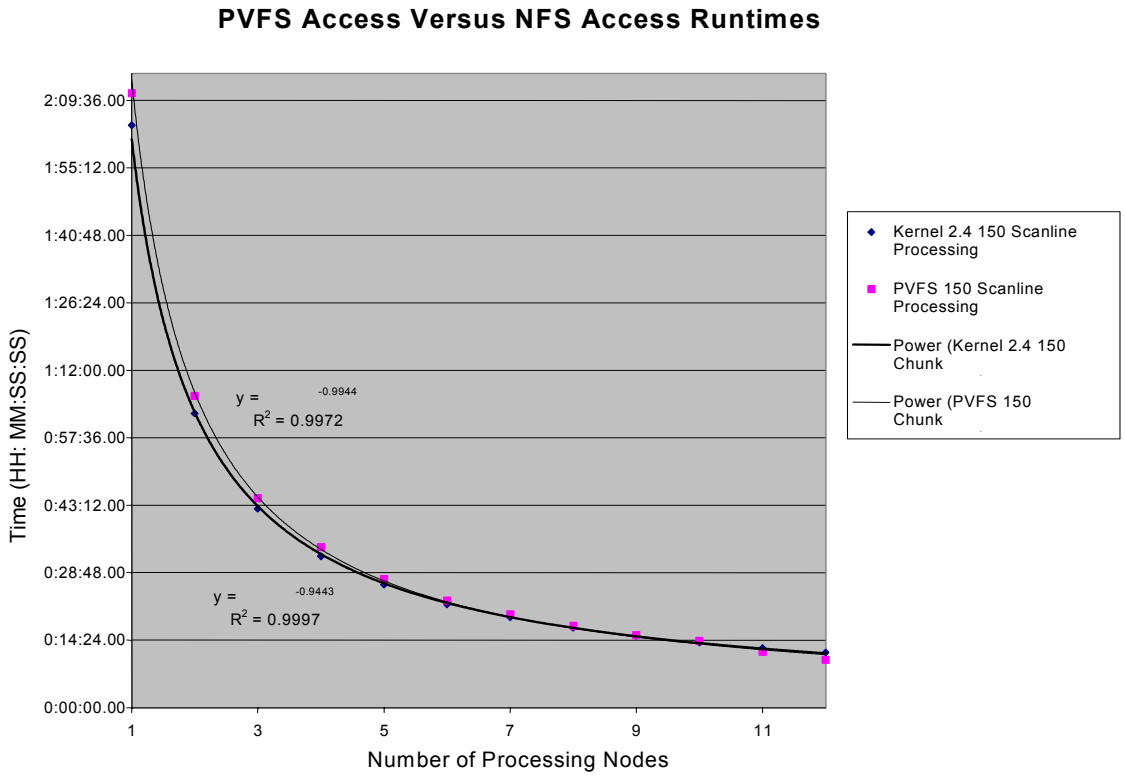

Figure 23. PVFS access versus NFS access times.

Even though PPM with PVFS did much to decrease the network congestion due to data requests, there was still significant network activity of processing nodes in each group "talking" to their I/O server and all the processing nodes trying to talk to the master node. It is thought that the bandwidth to the I/O servers and the bandwidth to the master could be improved by combining the PPM with an FNN cluster design.

The FNN testing came last, which was fortunate considering the troubles with implementation. This eventually proved to be the most troublesome part of the FY 2001 testing and also led to several discoveries about preexisting libraries for distributed processing.

The initial design for the MCMC FNN came from the University of Kentucky's Web site. This Web site has a Web form that can be used to generate a wiring diagram. The user enters in certain parameters, such as the number of ports on an Ethernet switch, number of Ethernet cards in a node, and total number of machines. The Web site then uses a version of a genetic algorithm developed at the university to design a wiring configuration that is optimal for peer-to-peer communication patterns. MCMC researchers slightly modified the design, and adopted a more balanced approach where every switch was connected to the same number of nodes. This was done to optimize peer-to-peer communications and to ensure that each switch could provide the maximum bandwidth possible to the machines connected to it.

The Web site only directed which machines should be connected to certain switches. The users are then left to determine how to connect the machines. To facilitate this, we arranged the machines in a grouping structure that comprised three groups of six nodes each. This allowed MCMC researchers to determine which Ethernet cards should be connected to the three switches. Once this was done, the machines were taken offline and the second Ethernet cards were installed. The entire cluster was then rewired on the basis of the FNN design chosen. The MCMC FNN setup is shown in this diagram, where the solid lines represent the connection to the first Ethernet card in each machine, and the dashed lines the second. There is currently no node 19, but it is included in the diagram to illustrate how it would be connected in the system. A 19th node would serve as the new head node and would be connected to each of the three switches.

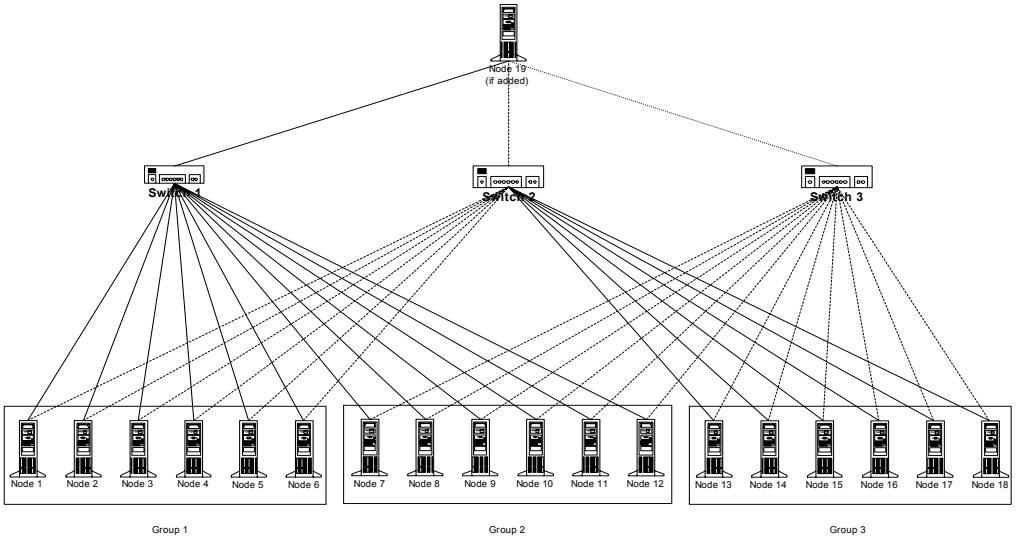

Figure 24. The MCMC FNN setup.

After the physical rewiring work was done, each node had to be visited to create the specialized routing that an FNN requires. For this, the routing on each node was created by hand, as no tools existed to automate the process. Great care was taken at this step, because each machine had to have a rather large and complex routing table so that it could contact the other machines properly. When this was done, the system as a whole was brought back online and tested extensively to make sure all of the network routings were correct. Once this was verified, we began testing to determine the effects on processing.

Testing quickly came to a halt when it was found that the PVFS libraries are by default incompatible with an FNN setup. As previously stated, in an FNN setup, a machine is known by several different IP addresses. The routing in this case is resolved by modifying the host table on each machine with the IP addresses it can use to communicate with the other machines. The problem with PVFS was that the metadata manager would send back an IP address instead of a host name. This caused problems, since the IP address that was passed back was the IP address the metadata manager used to connect to the file-serving nodes, not the IP address that the client machines would use to connect. This meant that the clients could never connect to the file servers to get any data. The PVFS library was modified so that it would send back a host name instead of an IP address. This allowed the client machines to resolve the proper IP address to get to the file-serving nodes.

Further research found that this method (sending only the IP address) is fairly common in distributed processing libraries. One of the main reasons that libraries do this is to try to save time. In a normal system, where each machine has only one IP address, passing the IP back to client machines keeps the clients from having to look up the IP address themselves. The lookup on the client machine takes two forms. It can be stored locally in what is known as a hosts file, or it may have to connect to an Internet server and ask the server what the IP address is for the host. The local method can be done very quickly, but the remote method can be very time consuming, as the Internet itself may be congested or the remote server may be busy fulfilling other host-name conversion requests. The FNN is a fairly new and specialized network topology, so most libraries do not explicitly offer support for it. This was the case for PVFS, which had to be modified at MCMC so that it would function properly.

The biggest problem came when the Parallel Virtual Machine (PVM)-based experimental software was to be tested. The PVM exhibited the same problems as the PVFS software library. In this case, however, an examination showed that the PVM would have required far more time to modify than the PVFS library. Considerable time was spent deciding what options were available. In the end, it was found that the FNN researchers at the University of Kentucky had submitted modifications to the LAM project that enable it to work on FNN architectures. LAM is an implementation of the MPI standard. Without these modifications, LAM would have the same problems that PVFS and PVM have.

The decision was then made to use the LAM libraries and switch to MPI. This presented a new difficulty: the projection software had to be modified to use LAM/MPI instead of PVM. Fortunately, the function calls and methods of using LAM and PVM are similar, so porting the software could be done much more quickly than modifying the PVM source code. The projection software was then ported to LAM and tested to ensure accuracy.

These problems, as well as the problems with the Red Hat 7.1 upgrades, slowed completion of the FNN testing until October 2001. However, initial results with the FNN suggest that the databound processing is indeed finishing faster than a normal network setup. One of the reasons for this is that, with the FNN setup, a file server can communicate with two client nodes simultaneously. These dual paths of communication can effectively double the bandwidth going out of the node. This allows two machines to get data at the same time from the file-server node.

Future Work

FNN technology will be further investigated at MCMC in FY 2002. This technology is still in its infancy, and research to find other benefits can be used to expand computer science knowledge about it. It is currently showing benefits for databound applications and can possibly be used to find other means of efficiently processing large amounts of data in a distributed fashion.

We may need to develop different distributed processing algorithms to take advantage of FNN topologies. Most algorithms now assume only a single network connection to a machine. Algorithms could be modified to take advantage of the extra bandwidth or the extra connectivity provided by the FNN architecture. For example, searching algorithms could be modified to use the extra connections, with the request going through one interface and the response routed through another.

Another possible modification could be made to the Linux kernel itself. As can be seen from the MCMC FNN diagram, some nodes in FNN configurations actually share more than one switch with another machine. A modification to the kernel could be made to check if a current Ethernet interface is congested when trying to connect to a machine. If such congestion occurs, we could make the operating system try to use the other interface to connect with a machine. This process is not without problems, because it would require some drastic changes to the Linux networking core. It would also introduce extra latency as the operating system switches between Ethernet interfaces when trying to connect to a machine.

These modifications could require yet another network topology that makes sure that each machine is connected to each other machine by multiple switches. Such topologies are possible but are generally quite complex and have problems

of their own. The complexity of designing such a network system, combined with the complexity of modifications to the Linux kernel, could make it next to impossible to do this research.

The methods studied this year can also be combined next year to lead to more research. One idea is to take advantage of the improved peer-to-peer communication capability of the FNN by combining it with the parallel I/O of PVFS. In this case, every node in the cluster could be used to create one giant virtual PVFS drive. When a file is written to this drive, it would be split across all of the nodes in the cluster. Since the FNN setup provides multiple network channels per machine, each node could serve its piece of a file to two other nodes at once. The processing-partitioning method could also be used to group nodes logically into the partitions discussed previously. This logical grouping could be used to spread processing evenly across all of the nodes in a cluster. Network contention for a node would be reduced, because the file parts would be served on so many machines that fewer nodes would be connecting to any given machine at any given time.

This method would probably yield better results than the previous research into processing partitioning. In the case of the MCMC setup, there are still at least six nodes trying to connect to a PVFS file-server node at any given time. Although this is an improvement over the single file-server model, further development could make it even more efficient. Distributing the data over all nodes in the cluster could lead to situations where only one to three nodes try to access any given node for data. The ability to communicate with multiple machines simultaneously could greatly increase file-processing throughput in this case. However, the partitioning would have to be crafted and experimented with in order to produce the best results.

ROCKY MOUNTAIN MAPPING CENTER

The RMMC has two Beowulf clusters that are situated in Building 810 on the Denver Federal Center in Colorado. The initial Beowulf cluster consists of a 16-node system that was created from castoff components. Seven of the machines had Pentium II 200-MHz processors, with five of these having 128 MB of EDO SIMM memory, one having 96 MB of EDO SIMM memory, and one equipped with 64 MB of memory. Each of these seven machines contained a single 3COM 3C905-TX NIC, with the master node having two NICs. Each of these computers contained a 4-GB EIDE hard drive. The other nine computers were equipped with Pentium 133-MHz processors and 64 MB of EDO RAM. Seven nodes had 2-GB EIDE hard drives, whereas the other two nodes had 4-GB EIDE hard drives. Each PC contained a single 3COM 3C595-TX NIC, and all were connected to a network by means of a 3COM OfficeConnect dual-speed unmanaged 16-port switch. All machines had a 3.5-inch floppy drive and a CD-ROM of undetermined speed. This cluster has experienced continuous replacement of equipment because of the age of the components. Funding to

set up this system has been minimal and entailed the purchase of a switch and of memory for some of the machines.

The original operating system that was loaded on the cluster was Red Hat Linux version 6.1. The computing nodes had the standard workstation installation of the software, and the master node was configured with server packages: Printer Support, X Windows System, GNOME, Mail/WWW/News Tools, Graphics Manipulation, Networked Workstation, NFS Server, SAMBA Server, Web Server, DNS Server, PostgreSQL Server, Authoring/Publishing Tools, Emacs, Development Package, Kernel Development, Clustering, and Utilities. Other software that was loaded onto the cluster included TripWire, IP Chains, XV, LAM/MPI 6.2.3, MPICH 1.2.0, and PVM 3.4.

Although the cost of setting up this castoff system was minimal, maintenance of the hardware proved costly. The equipment was approximately 5-6 years old, and intermittent failure of individual nodes occurred frequently enough to cause delays in timetables for deliverables. Multiple failures occurred on more than one occasion. These coincidences may be attributed to the procurement process at RMMC, where multiple machines are purchased at the same time, resulting in system parts having a similar lifespan. Although putting a system together with spare parts is usually cost effective at first, such a system may end up costing more in the long run owing to downtime and costs for repairs, man-hours involved with maintenance, and limited potential for application development on older machines.

The second system that resides at the RMMC was procured with USGS NMD Prospectus money (fig. 25). On the basis of the type of architecture available at the time of procurement, a conscious effort was made to match the hardware requirement with a specific application that was of interest at RMMC. At that time, the application was being executed on single CPU Silicon Graphics, Inc. (SGI), and Data General (DG) machines. The first choice was to select alpha processor machines, but that proved to be cost prohibitive. Other USGS Centers associated with the Beowulf project were procuring Pentium III systems, so RMMC decided to try a different processor to support clustering. Technical articles favored the ASUS A7V motherboard with the AMD Athlon /Duron processor for mathematical computations, and that is the motherboard/processor combination selected by RMMC. The second cluster is also a 16-node system, consisting of 1 master node and 15 computing nodes. The master node is a single CPU machine; a dual processor was not available with the ASUS motherboard. The master node contains an Athlon KT 133 chipset, 200-MHz bus speed, an AMD 750-MHz Thunderbird CPU with L2 on-Die Cache and 1.5-GB PC-133 registered ECC SDRAM (3 DIMMS). The master node also has a 30-GB IDE hard disk, a Viewsonic 21” SVGA monitor, an AGP

Figure 25. The 16-node Beowulf cluster at RMMC.

Visiontek video card with 32 MB, 2 – 10/100 NICs, a 104-key keyboard, a 3- button mouse, an HP9200I 32x/8x/4x rewriteable SCSI CD drive, a Sony 48x IDE CD-ROM, and one 3.5 inch floppy drive. The 15 computing nodes contain the same motherboard and chipset as the master node. Seven of the computing nodes contain three 512 MB of registered ECC PC-133 SDRAM DIMMs, while the remaining eight nodes contain three 512 MB of nonregistered, non-ECC, PC-133 SDRAM DIMMs. The different type of memory was a result of investigations on obtaining additional memory. The original memory that came with the cluster had ECC memory installed; however, it was later learned that the motherboard does not support ECC memory DIMMs, so the additional memory that was added to the cluster was of the non-ECC type. Each of the computing nodes also contains a Sony 48x IDE CD-ROM, a 15.3-GB IDE Seagate hard disk, a Kingston Etherx 10/100 NIC, and an ATI Expert 98 8-MB AGP video card. The system is networked using a NetGear Fast Ethernet 16-port unmanaged switch. An additional Belkin OmniView PRO KVM switch was added, which enables the user to control the 16 nodes from 1 keyboard, mouse, and monitor.

The operating system currently running on the second cluster is Red Hat LINUX version 6.2. RMMC intends to upgrade to Red Hat 7.1 when the load on the cluster permits. The kernel that is currently executing on the cluster is version

2.2.17. The server node has printer support, an X windows system, graphics manipulation tools, a networked workstation package, authoring and publishing tools, emacs, a kernel development package, a clustering tool, and standard development and utility packages. The master node has the TripWire security package installed along with Open Secure Shell, TCP Wrappers, and IPChains as its firewall. The cluster originally installed LAM/MPI version 6.2.3, MPICH version 1.2.0, and PVM version 3.4 on all nodes using NFS. The system has since been upgraded to LAM/MPI version 6.5.1 and MPICH version 1.2.2.

The computing nodes are running the standard installation of Red Hat 6.2 for workstations with the upgraded kernel. The computing nodes access the applications, utilities, and file system using the NFS mounts from the server node.

Some configuration issues are unique to the AMD architecture, in contrast to Intel- based systems. For example, it was necessary to rebuild the system during the course of the year. Because the system originally came factory installed, the system administrator was unaware of flags that needed to be placed in the startup scripts to allow the nodes to boot up. The general installation that comes on the Red Hat disks functions properly for Intel-based systems, as was the experience with the initial cluster that was comprised of all Pentium machines. After LINUX was successfully installed on the Pentium machines, the systems booted up using Red Hat 6.1 and then also when upgrading to Red Hat 6.2. This was not the case for the cluster composed of AMD hardware. Within the startup routines that boot the node, flags must be added to commands within the boot files to tell the system that the architecture is AMD. If this is not done, the LINUX Loader (LILO) will fail when attempting to load the individual packages of the system, causing the system to lock up.

Hardware and System Software Enhancements

With the construction of a LINUX server to use in the parallel-processing research project came issues that needed to be addressed strictly from a system administration aspect. A major concern at RMMC has been the need for increased network security. With cyber terrorism frequent throughout the internet, affecting standalone machines and servers alike, the requirement for a good security model was critical. Decisions had to be made regarding where the cluster would reside with respect to the network security already in place at the Center. It was decided that the cluster would reside outside the firewall. That decision required some investigation, prototyping, and implementation of a number of security measures. Secure Shell (SSH) was implemented, which allows secure TCP communications between any two systems regardless of which distrusted systems might be listening on each systems communications network. The actual version of SSH that was implemented was Open SSH version 2.9. Open SSH supports the SSH 1 and SSH 2 protocols. SSH encrypts all communication in an attempt to limit network-level attacks. SSH is

predominantly used for replacing telnet, rsh, rcp, and rlogin, some of which are used in the cluster environment when communicating between nodes. When RMMC upgraded to Red Hat LINUX 6.2, the decision was made to incorporate the firewall layer of the LINUX operating system, called IPChains. This has been the cluster's main application for network security and has performed well. IPChains is a basic packet-filtering firewall. Packets that pass through the LINUX cluster will first pass through the IPChains subsystem. Rules were added to the IPChains system checking to see which packet matches the rule and how the packet is handled on a match. The rules were organized into two categories, an input category and an output category, with the understanding that there is a third category, the forward category, which was not addressed on the RMMC cluster. The server has a filter that denies all traffic to and from the firewall system. Then, with the individual commands, control was given to individual IP addresses at the network level. Nmap version 2.54 was then loaded onto the server to scan for open ports to validate the successful implementation of the firewall.

RMMC's cluster initially ran only TCP Wrapper as its security system on the original configuration. TCP Wrapper is an interface that provides data security. TCP Wrapper controls which remote systems and users have access to the individual services. This application worked as the server security until a hole was recognized in a specific port under Red Hat LINUX 6.2. Compromise of the system necessitated a rebuild of the system, and at that time IPChains and SSH were implemented, along with TCP Wrapper.

SLEUTH Urban Growth Model

The main incentive for the RMMC to become involved in the investigation of Beowulf clusters has been a requirement to run Keith Clarke's SLEUTH Urban Growth Model. The model is very memory and CPU intensive. The original platform that the RMMC used to execute the model was a single CPU DG with 64 MB of memory. Data input required for the model includes temporal urban images, road images, excluded growth zones, and a "percentage-of-slope" image. Optional input includes temporal land use images and a shaded-relief image (required for the predictive phase of the model). After porting the model to the DG computer, we ran a test dataset. The array size of the input images was 1,100 by 1,400-pixels, with each image cell representing 30 m^2 on the ground. RMMC began the calibration part of the model, only to realize that completion of the run would take an inordinately long time. Hence, the array size was reduced by half and then halved twice more. The ratio of the images that were used for testing consisted of full resolution, $^1/_2$ resolution, $^1/_4$ resolution, $^1/_8$ resolution, and $^1/_{16}$ resolution. The $^1/_{16}$ resolution model was used to test the successful porting of the source code to the DG. The calibration phase of the model took approximately 28 hours to complete, which included 3,125 "single simulations" (or iterations of the code) of land cover change during the run. Execution of the predictive phase of the model required input of the results from the calibration run. The predictive run took approximately 24 hours to complete. For

benchmarking purposes, RMMC used the 1,100 by 1,400-pixel image on the calibration phase once again. The application had run for 7 weeks when there was a spike in the electrical current that caused the system to fail. Observing the log files up to that point showed that the model was only 76-percent complete. Similar results were noted when the model was moved from the DG to an SGI Origin 2000 server. In fairness, these machines were both running the model in a single CPU mode.

The code was given to the EPA, which subcontracted with Lockheed Martin to rewrite SLEUTH as a parallel-processing application. When the code was available to run on parallel processors, the original cluster was established, followed closely by the procurement of a second. SLEUTH was ported from the CRAY Operating System to Red Hat Linux 6.1 on the original cluster, and to Red Hat 6.2 on its second cluster. The SLEUTH package now comes with test images that were used to determine if the port of the model was successful.

Following the successful porting of SLEUTH, the same resolution images that were used on the DG and the SGI were run through the model on the cluster made from surplus machines. The results showed that SLEUTH running in a distributed environment was 7 to 10 times faster than in the single-CPU mode. What had taken days to complete was now taking hours on machines that had been deemed expendable by the USGS. Results on the procured system, which had faster CPUs and more memory, were even more impressive (see fig. 26).

The model has been executed numerous times throughout the year. RMMC collaborated with the New York City Environmental Protection Department, Water Quality and Control Unit, in executing the SLEUTH 03.00.03 Beta release for the Delaware River watershed (Delaware County). Urbanized area forecasts were modeled for the year 2040, enabling New York City to evaluate the potential total maximum daily loading (TMDL) of nutrients attributed to urbanization. In another application of the model, growth in urban land throughout the Front Range of Colorado was forecast to the year 2020. These results have provided valuable information to decision makers affiliated with the Denver Regional Council of Governments and the Colorado State Legislature. SLEUTH has also been used to generate urbanized area forecasts for the horizon years 2025 and 2050 for the Seattle, WA area and for an urbanization prediction to 2050 for Albuquerque, NM.

Platform	Array Size of Images	Accumulated Clock Time (days:hours:minutes)	Number of Iterations
Data General	87 x 72	00 : 10 : 39	2160
SGI	87 x 72	00 : 06 : 50	2160
Cluster	87 x 72	**00 : 01 : 58**	2160
Data General	175 x 145	00 : 17 : 01	1024
SGI	175 x 145	00 : 09 : 09	1024
Cluster	175 x 145	**00 : 02 : 54**	1024
Data General	234 x 193	10 : 05 : 54	3125
SGI	234 x 193	09 : 01 : 05	3125
Cluster	234 x 193	**01 : 16 : 23**	3125

Figure 26. Benchmarks based on data from the Isleta, NM, quadrangle.

ALASKA GEOGRAPHIC SCIENCE OFFICE

The Alaska Beowulf cluster is housed in an environmentally conditioned computer room at the USGS AGSO in Anchorage, Alaska. Currently, the six nodes making up the main cluster are organized on a modular two-shelf stand, as shown in figure 27. All of the equipment in this room is protected by an uninterruptible power supply. This central location provides easy access for USGS and cooperator staff. The master node is also connected to the USGS Wide Area Network for remote access to offsite scientists.

Hardware and Systems Software Enhancements

The Beowulf cluster at the AGSO consists of one master node and five computing nodes connected on a dedicated 100-mBit local area network. The master node is a Dell Precision 420 workstation configured with two Pentium III processors running at 733 MHz, 1,024 MB of RAM bus memory, 50 GB of EIDE hard disk capacity, and 70 GB of SCSI Disk. This computer has two 100-mBit NICs; one connected to the local area network and one connected to the dedicated cluster network by means of a 100-mBit switch. The second Pentium processor and the 70-GB SCSI disk drive were added to the master node as part of a system upgrade.

The computing nodes are five identical Dell Optiplex GX110 desktop computers. Each computing node has a Pentium III 733 MHz processor, 512 MB of memory, and a 50-GB EIDE hard disk, to which a 40-GB EIDE hard disk has been added. A 100-mBit NIC is connected to the dedicated cluster network.

Figure 27. At six nodes, the AGSO Beowulf cluster makes a compact and space-efficient layout.

All of the nodes are connected to a single 19-inch monitor, mouse, and keyboard with a 16-port OmniPro KVM switch. The dedicated network consists of a 3Com OfficeConnect 10/100-mBit 16-port switch connected to each node.

A second cluster has been created from four castoff 75-MHz Pentium computers with limited disk space. This cluster facilitates testing major system upgrades and changes without affecting the functionality of the main cluster. When the process and suitability of the change is determined to be safe, the updates are made to the main cluster. The test cluster is networked together with a 3Com OfficeConnect 10/100-mBit switch.

The software operating system has recently been updated to run under Red Hat Linux 7.1 and LAM/MPI version 6.5.4.

Applications

Applications are just reaching the implementation stage and work is progressing with application scientists running tests of multivariate nonhierarchical statistical clustering of remotely sensed spectral data. The initial test of this application will evaluate the speed and efficiency by which the Beowulf cluster can perform multivariate, nonhierarchical clustering on a series of Alaska area datasets. Another test will evaluate image classification and data analysis techniques using the Beowulf cluster to classify and delineate wetland and upland cover types within a 3.2-million-acre study area encompassed by land in the Kodiak Island archipelago and will assess the classification accuracy by using an analysis of ground reference data. Data for this test will come from multitemporal MRLC 2000 Landsat 7 ETM+ imagery and ancillary datasets such as slope, aspect, and elevation. Other data such as NDVI, illumination, and radiation, may be used if initial analysis finds the information useful.

Work has begun with application scientists at other Western Region centers to port existing modeling applications to the Alaska cluster for testing and to provide these scientists with extra computing power in their model analyses.

FUTURE DIRECTIONS AND ACTIVITIES

The objective of this research is to develop a thorough understanding of computer clusters and the class of geospatial problems that are best solved on these systems; this includes investigating system hardware, network topologies, system-level software, and applications.

Summary of FY 2001 Accomplishments

FY 2001 constituted Phase 1: Investigate Cluster Technologies. Project objectives and milestones as defined in the proposal were met and include the following:

- Held the **first team meeting** at the EROS Data Center. Team members from EROS, RMMC, AGSO, and MCMC attended.
- Published **Open File Report 01-244** entitled “A Parallel-Processing Approach to Computing for the Geographic Sciences,” which details the project’s progress to the midyear point, including system building and configuration at each site, systems-level software, networking setups, and details on applications software. A second Open File Report will be created in September 2001 documenting results achieved during the second half of the year.
- Integrated and tested the **Clark Urban Growth Model** in the cluster environment.
- Implemented the **CENTURY Carbon Model** in the cluster environment.
- Performed research on the efficient distributed processing of **large-image**

datasets and developed initial methods of processing partitioning to increase I/O throughput.
- Implemented an image **reprojection and resampling** algorithm for large datasets.
- Created a prototype tile-based **image resampling** algorithm for large satellite image sets.
- Started investigations into the **Cluster-of-Clusters** (the "Grid") experiment.
- Identified Mapping Applications Center **(MAC) participants** to be included in the project in FY 2002.
- Began an investigation of **OpenDX** for the visualization of results.

Proposed Fiscal Year 2002 Work Plan

In FY 2002, the project enters Phase 2, which encompasses the evaluation of prototype clusters and the refinement of modeling processes. The focus in this phase is on applications-computationally intensive models, image processing of remotely sensed data, and visualization, along with cluster configuration modifications necessary to perform them efficiently.

The following tasks have been identified for the project in FY 2002. Task 1 is an implementation necessary to accomplish this research at the MAC: the remaining tasks are hypothesis-driven research.

Task 1: Establish an Operational Beowulf Cluster at the MAC (MAC, EDC)

Replace the existing MAC Beowulf cluster composed of old and unreliable PCs. Procure a new system and install the same operating environment that is running on the EDC cluster. This Beowulf cluster is needed to provide a parallel processing capability at the MAC that will be used initially to model and predict future urban growth and associated impervious surfaces for the environs of Tampa Bay and Chesapeake Bay as high-priority input to water-quality studies.

Milestones/Deliverables:
- Technical specification and procurement of the cluster (early first quarter FY 2002).
- Cluster configuration and testing (late first quarter FY 2002).
- Installation of the SLEUTH Model and test installation (early second quarter FY 2002).

Task 2: Feature Extraction and Computational Intelligence (MCMC)

Hypothesis: Distributed processing technologies may finally provide the computational power necessary to enable the extraction of specific features in the landscape from remotely sensed imagery in an automated fashion using computational intelligence techniques.

Develop an approach to the automated extraction of features from remotely sensed imagery using aspects of computational intelligence (neural networks and fuzzy logic). For example, the ability to "scan" satellite imagery to determine median tidal boundaries and identify roads, buildings, orchards, prairie dog colonies, and other features is a capability that directly relates to revision tasks in support of Cooperative Topographic Mapping, as well as the Landscape Analysis Program.

Milestones/Deliverables:

- A literature review of state-of-the-art techniques for feature identification and extraction, including those that are based on computational intelligence techniques.
- Preliminary research for developing a cluster-based algorithm for computational-intelligence-based automated feature extraction.
- Publication of an Open-File Report and a peer-reviewed journal article.

Task 3: Processing Large Image Data Sets (MCMC, RMMC, EDC)

Hypothesis: Techniques and methodologies can be developed to allow a Beowulf cluster to process very large remote sensing datasets efficiently.

Conduct further research into the handling of very large remote sensing image datasets in a clustered environment. Comparisons between PVFS and other parallel file systems, such as the GPFS (General Purpose File System), will be performed. A hardware solution--Myrinet--will be investigated. The capability of Myrinet to execute a control program that interacts directly with the application, thus bypassing the cluster's operating system for low-latency communication could be one resolution to enhancing internodal communication, involving parallel applications on clustered systems. Channel bonding and the use of a dedicated data node will also be investigated as low-cost solutions to scalable I/O. This study will benefit programs that work with large datasets: Land Remote Sensing, Land Surface Analysis, and Cooperative Topographic Mapping.

Milestones/Deliverables:

- A report comparing PVFS and GPFS configurations (MCMC, RMMC).
- A report on research into the distributed processing of databound problems, including the use of distributed file systems and FNN

technology (MCMC).
- A report on the effectiveness of channel bonding (EDC, RMMC).
- A report on the effectiveness of a data node with dedicated network (EDC).
- Publication of an Open-File Report and a peer-reviewed journal article (EDC, MCMC, RMMC).

Task 4: Prototype the "Cluster of Clusters" Concept (EDC, RMMC)

Hypothesis: A computationally intense problem (too large for any single cluster in the USGS system) can be divided among multiple linked clusters in the USGS constellation and provide a prototype for the "USGS Compute Grid."

Conduct research into the Cluster-of-Clusters concept, also known in the industry as Grid Computing, using electric-power grids as an analogy to computing-power grids. This concept has potentially broad application across the USGS.

Milestones/Deliverables:
- A literature review of state-of-the-art techniques in Internet computing, collaborative computing, and networking methods that support the computing grid concept.
- Develop a method to tie together many of the clusters in this project via the Internet into a single, distributed compute system.
- Development of an algorithm to test this concept: Analyze the BRD-MESC Invasive Species Model as currently implemented with stepwise linear regression and replace the stepwise regression with an exhaustive regression (all possible combinations) algorithm. This is complementary to the application under study in the USGS Internet Supercomputing Venture Capital project and will provide a good comparison of the two approaches to computing.
- A publication in a peer-reviewed journal (EDC, RMMC).

Task 5: Interactive Visualization of Very Large Datasets (EDC, MAC)

Hypothesis: Beowulf clusters, combined with a visualization engine, can enable real-time visualizations of large remote sensing datasets and model results.

OpenDX, an open source version of IBM's Data Explorer product, is a useful visual-programming environment that can be used to model and visualize many physical and cultural geographic issues. When the models (input datasets) are small, this visualization package performs nicely on

single-CPU workstations. However, when data sizes are large, as is the case for many USGS datasets and study areas, the computational requirements of the two and three-dimensional modeling processes are too great for a single modern CPU. The latest release of OpenDX has parts of its modeling engine implemented for Beowulf clusters. This task seeks to evaluate and, if necessary, enhance this MPI-based implementation for very large remote sensing image datasets.

Milestones/Deliverables:

- A demonstration of OpenDx/Beowulf interactive two and three-dimensional modeling capabilities (third and fourth quarter FY 2002).
- An Open-File Report and a peer-reviewed journal publication (EDC, MAC).

Task 6: Model Investigation and Applications (MAC, MCMC, RMMC, AGSO)

Hypotheses: There are few models that can be practically applied to predict growth in urban areas, and the ones that can be applied are difficult to implement successfully. The USGS has experience using primarily one model, the SLUETH, developed by the University of California at Santa Barbara (UCSB), but is in the process of expanding its knowledge and application of other models. Using large datasets, we will examine comparisons between different predictive models; predictions will not be limited to urban feature changes. Performance between models will be compared in parallel-process balancing, accuracy of results, speed, and other areas.

Results from the following two endeavors will be incorporated: (1) collaborative research between RMMC and UCSB on modeling comparisons, and (2) modeling inventory released by the EPA (2000, EPA/600/R-00/098).

Deliverables (Milestones):

- Comparison of single-thread versus distributed-processing performance (first quarter FY 2002).
- Modification and testing of the algorithms to handle large datasets more efficiently (second quarter FY 2002).
- Evaluation of performances (third quarter FY 2002).

Task 7: Seismic Activity Modeling (AGSO)

Hypothesis: The Beowulf cluster can be used successfully to support and enhance seismic modeling capabilities within the USGS. Implement and test a stress-field model application on the Beowulf cluster. The stress-

field operating in the upper crust of southcentral Alaska is largely unknown. This is at least partly due to computational limitations that up until the last few years have made such a project unfeasible. Modern programs use orientations of earthquake fault planes to determine the optimal orientation of the stress tensor for a given region in the crust by means of a grid search method. These programs are highly CPU intensive and can take several hours (depending on the size of the dataset) to finish a single run. Analysis of the stress field in a single region usually requires days of processing on current systems. Performing such analysis on a series of subregions in southcentral Alaska would take weeks. It is believed that this study could benefit enormously from parallelization and implementation of the model on the Beowulf system.

Milestones/Deliverables:

- Porting the model to the Beowulf cluster.
- Evaluating the performance of the model application on the cluster.
- Application of model to the study areas.
- Evaluating the performance of the model application on the cluster.
- Publication of results.

Task 8: Web Mapping on the Fly (AGSO)

Hypothesis: Beowulf clusters can be used as an engine for on-the-fly processing of large, high-resolution datasets supporting the National Map, Web mapping, Open GIS, and Gateway to Earth test projects.

Investigate using parallel-processing techniques for on-the-fly geoprocessing of high-resolution image datasets. Integrate with Open GIS Web mapping services for real time Web mapping application.

Milestones/Deliverables:

- Literature review of current parallel-processing techniques and applications in image processing.
- Review of current Open GIS and Web mapping standards and protocols.
- Development of a prototype application using high-resolution data for Alaska, including DOQ, DEM, Landsat, and IFSAR.
- Incorporation of the prototype into the Web Mapping Testbed project.
- Publication of results.

BIBLIOGRAPHY

SELECTED REFERENCES

Anderson, P., 1999, *The Texas Tech tornado cluster: A Linux/MPI cluster for parallel programming education and research*, *in* Crossroads, Association for Computing Machinery, Issue 6.1, p. 28-32.

Brown, R.G., 2000, *Maximizing Beowulf performance*: Durham, NC, Duke University Physics Department.

Brown, R.G., 2000, *Engineering a Beowulf-style computer cluster*: Durham, NC, Duke University Physics Department.

Byars, Bruce, 2001, *Welcome to the official GRASS GIS homepage*, http://www.baylor.edu/grass/index2.html.

Crane, M., Steinwand, D., Beckmann, T., Krpan, G., Haga, J., Maddox, B., and Feller, M., 2001, *A parallel-processing approach to computing for the geographic sciences*: USGS Open-File Report 01-244.

Decyk, V.K., Dauger, D.E., and Kokelaar, P.R., 1999, *How to build an AppleSeed: A parallel Macintosh cluster for numerically intensive computing*: presented at the International Topical Conference on Plasma Physics, Faro, Portugal, September 1999.

Dedkov, Anatholy F. and Eadline, Douglas J., 1995, *Performance considerations for I/O dominant applications on parallel computers*: Paralogic Corporation, ftp://www.plogic.com/pub/papers/exs-pap6.ps.

Geist, A., Beguelin, A. et al., 1997, *PVM: A user's guide and tutorial for networked parallel computing*: MA, MIT Press, Cambridge.

Gropp, W., Lusk, E., and Skjellm, A., 1999a, *Using MPI: Portable parallel programming with the message-passing interface*: MA, MIT Press, Cambridge.

Gropp, W., Juss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., and Snir, M., 1999b, *MPI: The Complete Reference: Volume 2, The MPI Extensions*: MA, MIT Press, Cambridge, MA.

Hargrove, W.W., and Luxmoore, R.J., 1998, *A new high-resolution national map of vegetation ecoregions produced empirically using multivariate spatial clustering*: http://www.esd.ornl.gov/~hnw/esri98/.

Hargrove, W.W., and Hoffman, F.M, 1999a, *Using multivariate clustering to characterize ecoregion borders*: Computing in Science & Engineering, v. 1, no. 4, p. 18-25.

Hargrove, W.W., and Hoffman, F.M., 1999b, *Multivariate geographic clustering using a Beowulf-style parallel computer*: http://research.esd.ornl.gov/~forrest/pdpta-1999/.

Hargrove, W.W., and Hoffman, F.M., 2000, *An analytical assessment tool for predicting changes in a species distribution map following changes in environmental conditions*: Fourth International Conference on Integrating GIS and Environmental Modeling, Banff, Alberta, Canada.

Hoffman, F.M., and Hargrove, W.W., 1998, *Making soup from stones*: Troubleshooting Professional, v. 2, issue 5.

Hoffman, F.M., and Hargrove, W.W., 1999, *Parallel computing with Linux*: Crossroads, v. 6, no. 1.

Hoffman, F.M., and Hargrove, W.W., 1999, *Multivariate geographic clustering using a Beowulf-style parallel computer*: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99), v. III, CSREA Press, p. 1292-1298.

Hoffman, F.M., and Hargrove, W.W., 1999, *Cluster computing: Linux taken to the extreme*, Linux Magazine, v. 1, no. 1, p. 56-59.

Kittel, T.G.F., Rosenbloom, N.A., Painter, T.H., Schimel, D.S., Fisher, H.H., Grimsdell, A., 1999, *The VEMAP Phase I Database: An integrated input dataset for ecosystem and vegetation modeling for the conterminous United States*, http://www.cgd.ucar.edu/vemap/user_guide.html.

Knuth, D.E., 1997, *The art of computer programming, Volume 2, Seminumerical Algorithms, 3rd Edition*: Addison-Wesley.

Ligon, W.B., and Ross, R.B., 2000, *An overview of the parallel virtual file system*: Parallel Architecture Research Lab, Clemson, SC, Clemson University.

Liu, S., Reiners, W.A., Keller, M., and Schimel, D.S., 1999, *Model simulation of changes of N_2O and NO emissions with conversion of tropical rain forests to pastures in the Costa Rican Atlantic Zone*: Global Biogeochemical Cycles, v. 13, p. 663-677.

Liu, S., Reiners, W.A., Keller, M., and Schimel, D.S., 2000, *Simulation of nitrous oxide and nitric oxide emissions from tropical primary forests in the Costa Rican Atlantic Zone*: Environmental Modeling & Software, v. 15, p. 727-743.

Liu, S., Bliss, N.B., and Loveland, T.R., 2001, *A generic model for simulating carbon dynamics in terrestrial ecosystems over large areas (COLA) with data assimilation capability. (in preparation)*.

Mahinthakumar, G.F., Hoffman, F.M., Hargrove, W.W., and Karonis, N.T., 1999, *Multivariate geographic clustering in a metacomputing environment using globus*: Proceedings of the ACM/IEEE SC99 Conference, Portland, OR.

Matthew, N., and Stones, R., 2000, *Professional Linux programming: Databases, PostgreSQL, MySQL, LDAP, security, device drivers, GTK+, GNOME, Glade, GUI, KDE, Qt, Python, PHP, RPC, diskless systems, multimedia, internationalization, CORBA, PAM, RPM, CVS, Flex, Bison, Beowulf, Clustering, ORBit, MPI, PVM, and XML*: Wrox Press.

Parton, W.J., Schimel, D.S., Cole C.V., and Ojima, D.S., 1987*, Analysis of factors controlling soil organic matter levels in Great Plains grasslands*: Soil Science Society of America Journal, v. 51, p. 785-809.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1992, *Numerical recipes in C—the art of scientific computing, Second Edition*: Cambridge University Press.

Red Hat, Inc., 2000, *The official Red Hat Linux getting started guide: Linux 6.2*: Durham, NC, Red Hat, Inc.

Reiners, W.A., Liu, S., Gerow, K.G., Keller, M., and Schimel, D.S., 2001, Historical and future land use effects on N_2O and NO emissions using an ensemble modeling approach: Costa Rica's Caribbean lowlands as an example: Global Biogeochemical Cycles (submitted).

Salmon, J., Savarese, D. F., and Sterling, T., 2001, *Managing high-volume astronomical data with heterogeneous Beowulf clusters*: Center for Advanced Computing Research, http://www.cacr.caltech.edu/Publications/techpubs/PAPERS/cacr176/cacr176.html.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., 1998, *MPI: The complete reference: Volume 1, The MPI Core*: Cambridge, MA, MIT Press.

Spector, D.H.M., 2000, *Building LINUX clusters*: Sebastopol, CA, O'Reilly and Associates.

Sterling, T., Cwik, T., Becker, D., Salmon, J., Warren, M., and Nitzberg, B., 1998, *An assessment of Beowulf-class computing for NASA requirements: Initial findings from the First NASA Workshop on Beowulf-class clustered computing*: IEEE 1998 Aerospace Conference, Snowmass, CO.

Sterling, T.L., Salmon, J., Becker, D.J., and Savarese, D.F., 1999, *How to build a Beowulf: A guide to the implementation and application of PC Clusters*: Cambridge, MA, MIT Press.

Thakur, R., Gropp, W., and Lusk, E., 2000, *MPI-OI: A standard, portable API for high performance parallel I/O*: Argonne, IL, Argonne National Laboratory.

_____, 2001, *Bunyip (Beowulf) project*: Canberra, Australia, Australian National University at Canberra, http://tux.anu.edu.au/Projects/Beowulf/.

_____, 2001, *FNN: Flat neighborhood* networks: University of Kentucky, http://aggregate.org/FNN.

_____, 2001, *LAM/MPI parallel computing*: LAM Team, http://www.lam-mpi.org/.

_____, 2001, *Message passing interface forum*, http://www.mpi-forum.org/.

_____, 2001, *PVM: parallel virtual machine*: Oak Ridge National Laboratories, http://www.epm.ornl.gov/pvm/pvm_home.html.

_____, 2001, *The parallel virtual filesystem project*: Clemson, SC, Clemson University, http://parlweb.parl.clemson.edu/pvfs/.

Wang, P., Cwik, T., and Green, R., 2000, *Benchmarks for AVIRIS algorithms on a Beowulf parallel computer system*: AVIRIS Earth Science and Applications Workshop, Pasadena, CA.

Warmerdam, Frank, 2001, *GeoTIFF*, http://www.remotesensing.org/geotiff/geotiff.html.

Welles, Mike, 2001, *TIFF software*, http://www.libtiff.org.

Welsh, Dalheimer, and Kaufman, 1999, *Running LINUX*: Sebastopol, CA, O'Reilly and Associates.

RELEVANT WEB SITES

http://www.beowulf.com – provides a history of the Beowulf project, a partial list of other Beowulf Web sites to check, cluster-related projects, and commercial Beowulf Web sites.

http://www.epcc.ed.ac.uk/epcc-tec/documents.html – the Web site for the Edinburgh Parallel Computing Centre provides a list of free publications and lecture notes and a calendar of upcoming events.

http://clusters.top500.org - this is an electronic newsletter that contains articles about benchmarks, Beowulf clusters, hardware and software, press releases, and editorials.

http://www.euroben.nl/ - provides benchmark programs for scientific and technical computing to assess the performance of computers for these fields.

http://www.nrel.colostate.edu/projects/century/ - provides a description of the Century Model and its applications.

APPENDIXES

Appendix A: Quick and Easy GUIs With Tcl/Tk

This document examines Tcl/Tk as a C programmers' tool for creating easy graphical user interfaces. Brief primers are presented, first on Tcl and then on Tk. This is followed by a discussion of the three primary methods for interfacing C with Tcl: (1) using a Tk GUI to collect arguments and call the C utility, (2) writing new Tcl commands in C, and (3) using Tcl's C API, with a focus on the API method. This document assumes that the reader has little to no experience with Tcl and that the reader knows C or C++ and Perl. It's provided as a starting point for creating simple GUIs for cluster programs. References are given at the end of this appendix for those who wish to use TCL/TK in their projects.

Tcl Basic Syntax

The Tcl interpreter is named tclsh. The interpreter with Tk commands is wish. Therefore, the first line of a Tcl script is

```
#!/usr/bin/tclsh
```

and the first line of a Tcl/Tk script is

```
#!/usr/bin/wish
```

Like other interpreters, these can be run from the command line and execute commands as they are entered.

set saves a value into a variable.

This is correct: set histring "Hello world"
This is not: histring = "Hello world"

puts writes a string to stdout. It automatically appends a newline character.

```
puts "Hello"                    Hello
set histring "Hello"
puts $histring                  Hello
```

Notice the **$** above. Tcl treats words as string literals unless the substitution is explicitly made.

```
puts Hello                      Hello
set histring "Hello"
puts histring                   histring
puts $histring                  Hello
```

{ Braces **}** group things together.

```
puts {Hello World}              Hello world
set histring {Hello world}
puts $histring                  Hello world
```

The end of the line symbolizes the end of the command. To continue a command onto the next line or put two commands on the same line, special characters are needed.

```
set histring "Hello world" ; puts $histring
puts \
"Hello world"
```

Using braces, as with **for** and **if**, is a special case. The opening brace must be on the first line, otherwise the rest is interpreted as a second command.

```
For { set x 0 } { $x < 10 } {incr x} {
        if {x == 5 } {
          puts “x is five”
        } else {
          puts “x is not five”
        }
}
```

It can be done this way instead:

```
for { set x 0 } { $x < 10 } {incr x }\
{ \
  if …
```

Remember the **** after the opening brace is necessary.

Tcl parses the comment symbol (**#**) as a command, not like comments in other languages.

```
#This is a comment. If it is continued \
onto the next line, it is still a comment.

puts $histring # This is an error
puts $histring ;# this is correct
puts\
#this is also an error
$histring
```

[Brackets **]** tell the interpreter to evaluate what is inside before continuing.

puts expr 1 + 1	Error: too many args
puts **{**expr 1 + 1**}**	expr 1 + 1
puts **[**expr 1 + 1**]**	2

Miscellaneous information

- Tcl is case sensitive
- White space separates a command’s arguments; commas do not
- C-style escape characters are used in strings: \n, \a, \t, \”, …
- ( )’s are only used for array subscripts and in regular expressions
- Tcl arrays are hash tables, not conventional arrays; any string is a legitimate subscript.

Tk

Tk widgets and widget options are explained in most of the Tcl/Tk books available, so this is a very brief overview.

To create a generic widget:

```
widgettype .widgetname –option1 value –option2 value
```

widgettype is a Tk command; for example button, frame, label, etc.
.widgetname is the name of this widget, with complete path.

The main window is named '**.**', and all other widgets, even toplevel widgets (separate windows), are its children (or children of its children).
For example:

```
frame .aframe
label .aframe.alabel –text "This label widget is in a frame"
```

To make a widget appear in the main window, it must be packed.

```
pack .aframe –anchor center
pack .aframe.alabel –side top
```

The order in which widgets are listed to pack is significant; besides changing the way a window looks, the wrong packing order can cause errors. A child widget cannot be packed before its parent unless the correct parameters are given. pack options are well explained in most Tcl books. There are other "geometry managers" available with Tcl: for example, grid makes it easy to align widgets in both rows and columns; frames are needed to do this with pack.

Tk and Other Scripting Languages

Perl and Python also have Tk implementations. The Perl community ported Tk. The Python interpreter just passes any Tk commands to the Tcl interpreter. Tk commands are the same in all implementations and widget parameters vary only slightly, but syntax varies quite a bit between Perl and Tcl.

Tcl/Tk:

```
label .hilabel -text "Hello World in Tcl/Tk"
button .exitbutton -text "Exit" \
        -command exit
# Tcl's pack is its own command; packing is done after creation
# items with the same parameters may be packed in one call
pack .hilabel .exitbutton –side top
#Don't explicitly run the main loop in Tcl script
```

Perl with the Tk module:

```
$mw = new MainWindow;  #must declare main window
$mw->title("Hello World with Perl's Tk module");
$hilabel = $mw->Label(-text "Hello World")->pack(-side => top);
# '=>' is equivalent to ',' but makes option pairs easier to read
$exitbutton = $mw->Button(-text => "Exit",
        –command => sub {exit})->pack(-side => top);
#Perl's packer is a member function of each object
#packing may be done at creation, or later:
#       $hilabel = $mw->Label (-text => "Hello World");
#       $hilabel->pack(-side => top);
MainLoop;  #must explicitly run the main loop in Perl script
```

Interfacing Tcl with C

There are three main ways to mingle C and Tcl/Tk: write a Tcl wrapper to a C program, create a new Tcl command with C, or use Tcl's C API.

Tcl Wrapper

Write a script with the GUI specs, get parameters via the GUI, and have the script call the C program, passing those parameters. This is probably what is needed most of the time, and most importantly, it's simple.

To do this, write a Tcl script with the GUI layout. Make sure it has an "OK" or "Go" button with the parameter "-command { some_proc_name }." The procedure some_proc_name should check the correctness of parameters then run the program.

Example calling procedure:

```
proc some_proc_name {} {
#set the argument variables as global here or outside proc.
global arg1 arg2 arg3
#check for legitimate values
#execute the program:
exec myCprog $arg1 $arg2 $arg3
}
```

Using the exec command and creating procedures in Tcl are covered by all Tcl books.

Tcl Commands

The second way to get Tcl and C to interact is to write a new Tcl command. This technique seems to be aimed at users who write everything in Tcl, want to do something Tcl can't do (or doesn't do well), and who happen to know C. Tcl command writing is documented in all Tcl books that advertise a section on integrating with C, but it is not covered here.

Tcl's C API

The third method is the C API. Tcl was written in C, and the Tcl people made an interface for C/C++ programs. Almost anything possible in Tcl can be accessed through this interface: variables in the interpreter can be changed or created, Tk event bindings can be made, the GUI can be modified, all through the API.

This is the basic algorithm for main() with the C API, with examples of relevant procedures listed below.

- create a Tcl interpreter object
  - Tcl_Interp *interp = Tcl_CreateInterp();
- initialize Tcl and Tk
  - Tcl_Init, Tk_Init
- do any other Tcl setup: execute external scripts, set up variable traces, etc.
  - Tcl_EvalFile, Tcl_TraceVar
- do C initialization
- display the GUI and start processing events
  - Tk_MainLoop()

This can be complicated, so main.c for the BandInfo utility, the GUI developed to display the results of the "Process Parallel" programming example, will be dissected and explained.

The files tcl.h and tk.h are always included. Since this program uses the BLT package's C API, blt.h is also included.

```
//---------- global vars ----------//
// Misc. vars for the C-only part are omitted
Tcl_Interp *interp;     // The conventional Tcl interpreter name
Blt_Vector *tcl_Yvec;

int main(int argc, char *argv[])
{
//---------- initialize C-only things ----------//
```

The next three lines are needed by all C/Tcl/Tk programs. Tcl or Tk must be initialized. If not, the program may or may not run, but it will not run properly. The BLT package also has a C interface with its vector type; this must also be initialized.

```
interp = Tcl_CreateInterp();    // Create a new Tcl interpreter
Tcl_Init(interp);               // inititalize Tcl
Tk_Init(interp);                // initialize Tk
Blt_Init(interp);               // initialize BLT
```

Tcl uses an argv similar to C's. The parameter to this C program was the filename to load. The Tcl script also needs this parameter, so argv must be set in the interpreter before the script is run. The third parameter to Tcl_SetVar is for flags. None are needed here, so null is passed.

```
Tcl_SetVar(interp, "argv", argv[1], 0);
      //copy argv[1] from C to argv in Tcl interpreter
if(Tcl_EvalFile(interp, SCRIPT) != TCL_OK)
{ printf("Tcl Error with %s\n",SCRIPT);exit(-1); }
```

The C part of the program needs to know when the mouse coordinates change so it can read the corresponding data from files. A trace is put out on the Ycoord Tcl variable, which will call proc when a value is written to Ycoord.

```
Tcl_TraceVar(interp, "Ycoord", TCL_TRACE_WRITES, Proc, (ClientData)NULL);
      // When Ycoord changes, read new data from file
```

To update Yvec with new data, there must be a pointer to the vector.

```
Blt_GetVector(interp, "Yvec", &tcl_Yvec);
      // Get pointer to Yvec in interpreter for use later
```

All the preparation is done. The script with GUI specifications has been run, a trace has been placed on the proper variable, and a pointer to the BLT vector has been created for updating the data. Now the GUI can be displayed and events can be processed.

```
Tk_MainLoop();          // Display GUI and process events
} // End of main
```

When the coordinate variables are updated in the Tcl interpreter, this procedure is used. ClientData is a Tcl data type for passing other information; it isn't used in this procedure. name1 is a Tcl variable name. name2 is a subscript if name1 is an array, null otherwise.

```
char *Proc(ClientData data, Tcl_Interp *interp,
char *name1, char *name2, int flags)
{
```

```
char *tempstr;
double tempvec[26];
```

The Tcl variables need to be copied into C variables. First Tcl_GetVar is called, which returns a string. Then the string is converted to an integer.

```
tempstr = Tcl_GetVar(interp, "Ycoord", flags);
sscanf(tempstr, "%d", &line);
tempstr = Tcl_GetVar(interp, "Xcoord", flags);
sscanf(tempstr, "%d", &samp);
//----- call the band query routine -----//
//----- BLT vector stuff -----//
// Copy unsigned char data into double array tempvec
```

tcl_Yvec is the BLT vector pointer found earlier, and tempvec is the data copied into a double array; the other three parameters are explained in Tcl documentation.

```
if(Blt_ResetVector(tcl_Yvec, tempvec, filecount, filecount,
TCL_VOLATILE) != TCL_OK)
      printf("Problem resetting vector...\n");
```

One way to update the background color of a label on the fly is to create the Tcl command as a string and call it via the C API. Tcl uses dynamic memory for its strings, as one would expect, but it needs to change command strings internally. Some C compilers (the Gnu compiler is one) make constant strings read-only. There are compiler options to change this; however, this example allocates memory for the string manually.

```
tempstr = (char *)malloc(256);      // 256 bytes is sufficient
sprintf(tempstr,".status configure –background $colors(%d.0)",
            (int)tempvec[0]);
if(Tcl_Eval(interp, tempstr) != TCL_OK)
            {printf("Tcl Error, fix me\n\n");exit(-1);}
// Update the coordinates displayed on screen
sprintf(tempstr,"set statusmsg \"( %d , %d )\nClass: %d\"",
            samp, line, (int)tempvec[0]);
if(Tcl_Eval(interp, tempstr) != TCL_OK)
            {printf("Tcl Error\n\n");exit(-1);}
free(tempstr);
```

Since there were no errors, a null string is returned.

```
return "";
}
```

```
#!/usr/bin/wish
package require BLT

#############################################################
# The bar.tcl script to be used with the Bandinfo utility #
#############################################################
```

```
########## set "constants" ##########
set bands 26
set barchartheight [expr [winfo screenheight .] / 5 ]
set barchartwidth  [expr [winfo screenwidth .]  * 5 / 6 ]
set imgfile "$argv.gif"
puts "Opening $imgfile..."
tk appname "$argv Band Info"
global Yvec
set originalstatus "click for\nmore information"
set statusmsg $originalstatus
set coordmsg  "( 0.0 , 0.0 )"
set togglemsg "Click"
blt::vector create Xvec($bands)
blt::vector create Yvec($bands)
Yvec notify always
Xvec set { 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 }
Yvec set { 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 }
#array subscripts are strings, not numbers. BLT vectors are floats, so the subscripts need to be
set colors(0.0)  #000000;  set colors(1.0)  #FF0000
set colors(2.0)  #A05228;  set colors(3.0)  #865228
set colors(4.0)  #9F3600;  set colors(5.0)  #B8860B
set colors(6.0)  #9ACD32;  set colors(7.0)  #EADD82
set colors(8.0)  #FF8C00;  set colors(9.0)  #FFA343
set colors(10.0) #FFD700;  set colors(11.0) #3CB371
set colors(12.0) #698A41;  set colors(13.0) #7AFC00
set colors(14.0) #227922;  set colors(15.0) #7DD998
set colors(16.0) #87CEEB;  set colors(17.0) #889BD1
set colors(18.0) #6D79BC;  set colors(19.0) #969696
set colors(20.0) #CB82C7;  set colors(21.0) #BC8F8F
set colors(22.0) #D579B4;  set colors(23.0) #FDBAF2
set colors(24.0) #FFFFFF;  set colors(25.0) #404040
set colors(100.0) #404040

########## Create Widgets ##########
button .key -text "Key" -command { show_key }
button .exit -text "Exit" -command { exit }
checkbutton .toggle -text "Update on motion" -variable togglevar -command { toggle_binding }
label .status -textvariable statusmsg -justify center -borderwidth 2 -relief groove -height 3
label .coordbox -textvariable coordmsg -justify center -borderwidth 2 -relief groove
frame .f
image create photo img1 -file $imgfile
canvas .f.bigimg -cursor {crosshair gray50}  \
            -borderwidth 0

########## configure canvas for displaying image ##########
set img1_w [image width img1]
set img1_h [image height img1]
### wish won't do variable substitution within the scrollregion
### parameter, (perhaps something with the braces?) but this does work.
set command ".f.bigimg configure -scrollregion { 0 0 $img1_w $img1_h }"
eval $command
unset command

grid .f.bigimg -sticky news -row 0 -column 0
```

```
if { $img1_h > [ expr [winfo screenheight .] - $barchartheight ] } {
   scrollbar .f.yscroll -orient vertical -command [list .f.bigimg yview] -width 10
   grid .f.yscroll -row 0 -column 1 -sticky nes
   .f.bigimg configure -height [expr [winfo screenheight .] - $barchartheight - 10]\
                    -yscrollcommand [list .f.yscroll set]
} else {
    .f.bigimg configure -height [expr $img1_h -1]
}

if { $img1_w > [winfo screenwidth .] } {
  scrollbar .f.xscroll -orient horizontal -command [list .f.bigimg xview] -width 10
  grid .f.xscroll -sticky sew -row 1 -column 0
  .f.bigimg configure -width [expr [winfo screenwidth .] - 50 ]\
                    -xscrollcommand [list .f.xscroll set]
} else {
    .f.bigimg configure -width [expr $img1_w -1]
}


########## Create & Configure barchart ##########
blt::barchart .bchart \
 -plotborderwidth 0\
 -plotbackground black\
 -width  $barchartwidth\
 -height $barchartheight

.bchart legend configure -hide yes
.bchart axis configure x -min {0.6}
.bchart axis configure y -min {0} -max {50}

.bchart element create e -xdata Xvec -ydata Yvec -fg green -relief flat

########## Pack widgets ##########
pack .f -side top
pack .bchart   -side left -anchor w
pack .coordbox -side top -fill x
pack .status   -side top -fill both
pack .exit     -side bottom -fill x
pack .key      -side bottom
pack .toggle   -side bottom -fill x

########## Bind events ##########
bind all <k> {show_key}
bind all <q> {exit}
focus .f.bigimg
bind .f.bigimg <Button-1> {
      set Xcoord [.f.bigimg canvasx %x]
      set Ycoord [.f.bigimg canvasx %y]
}
bind .f.bigimg <Motion> {
      set coordmsg "( [.f.bigimg canvasx %x] , [.f.bigimg canvasy %y] )"
}

.f.bigimg create image 0 0 -image img1 -anchor nw
#############################################################################
```

```
########## toggle_binding ########
# toggle between update barchart #
# on click and update on motion  #
####################################
proc toggle_binding {} {
 global togglevar statusmsg originalstatus
 if { $togglevar } {
  bind .f.bigimg <Motion> {}
  bind .f.bigimg <Motion> {
   set coordmsg "( [.f.bigimg canvasx %x] , [.f.bigimg canvasy %y] )"
   set Xcoord [.f.bigimg canvasx %x]
   set Ycoord [.f.bigimg canvasy %y]
  }
  bind .f.bigimg <Button-1> {}
 } else {
  bind .f.bigimg <Motion> {
   set coordmsg "( [.f.bigimg canvasx %x] , [.f.bigimg canvasy %y] )"
  }
  bind .f.bigimg <Button-1> {
   set Xcoord [.f.bigimg canvasx %x]
   set Ycoord [.f.bigimg canvasy %y]
  }
  set statusmsg $originalstatus
 }
}

########## show band key ######################
# Show the key for bands in a separate window #
###############################################
proc show_key {} {
global Xvec colors bands

bind all <k> {
  bind all <k> { show_key }
  .key configure -command { show_key } -relief raised
  destroy ".color key"
}

.key configure \
 -relief sunken \
 -command {
   .key configure -command { show_key } -relief raised
   bind all <k> { show_key }
   destroy ".color key"
 }

toplevel ".color key"
button ".color key.exit" -text "Close" -command {
            focus .f.bigimg
            bind all <k> { show_key }
            .key configure -command { show_key } -relief raised
            destroy ".color key"
   }
pack ".color key.exit" -side bottom

set classtr(0)  "(0) Interrupted areas"
```

```
set classtr(1)  "(1) Urban and Built-Up Land"
set classtr(2)  "(2) Dryland Cropland and Pasture"
set classtr(3)  "(3) Irrigated Cropland and Pasture"
set classtr(4)  "(4) Mixed Dryland/Irrigated Cropland and Pasture"
set classtr(5)  "(5) Cropland/Grassland Mosaic"
set classtr(6)  "(6) Cropland/Woodland Mosaic"
set classtr(7)  "(7) Grassland"
set classtr(8)  "(8) Shrubland"
set classtr(9)  "(9) Mixed Shrubland/Grassland"
set classtr(10) "(10) Savanna"
set classtr(11) "(11) Deciduous Broadleaf Forest"
set classtr(12) "(12) Deciduous Needleleaf Forest"
set classtr(13) "(13) Evergreen Broadleaf Forest"
set classtr(14) "(14) Evergreen Needleleaf Forest"
set classtr(15) "(15) Mixed Forest"
set classtr(16) "(16) Water Bodies"
set classtr(17) "(17) Herbaceous Wetland"
set classtr(18) "(18) Wooded Wetland"
set classtr(19) "(19) Barren or Sparsely Vegetated"
set classtr(20) "(20) Herbaceous Tundra"
set classtr(21) "(21) Wooded Tundra"
set classtr(22) "(22) Mixed Tundra"
set classtr(23) "(23) Bare Ground Tundra"
set classtr(24) "(24) Snow or Ice"
set classtr(25) "(100) no data"

frame ".color key.l" -borderwidth 5
pack ".color key.l" -side top

label ".color key.l.0" -background black -fg gray -text $classtr(0) -justify left
pack ".color key.l.0" -side top -fill x -anchor w

for {set i 1} { $i < $bands } {incr i } {
 label ".color key.l.$i" -background $colors($i.0) -text $classtr($i) -justify left
 pack ".color key.l.$i" -side top -fill x -anchor w
}

focus ".color key.exit"

}


########## Override error handler ###########################################
# For whatever reason, if a variable is used in a binding and a C trace, #
# Tcl gives a "can't set var" error when it comes back from the C call.  #
# Instead of discovering and actually fixing the problem, I overrode the #
# Tk error handler to silently ignore "can't set..." errors, but still   #
# write other errors to stdout.                                          #
##########################################################################
proc bgerror { whatever } {
set temp [regexp {^can't set} $whatever ]
if {!$temp} {
          puts $whatever
  }
      return ""
}
```

```
/* main.c                          *\
\*---------------------------------*/
/* Interfaces with Tcl to display data *\
\* on screen as a bar graph           */

#include <stdio.h>
#include <tcl.h>
#include <tk.h>
#include <blt.h>
#define SCRIPT "bar.tcl"

//---------- prototypes ----------//
char *Proc(ClientData, Tcl_Interp *, char *, char *, int);
void point_query(FILE **, unsigned char*, long, long, long, int);

//---------- global vars ----------//
FILE *files[256];
int filecount;
long line, samp;
long nl, ns, nb;
double ulx, uly;
float pixsiz;
unsigned char vector[256];
Tcl_Interp *interp;          // The obligatory Tcl Interpreter name
Blt_Vector *tcl_Yvec;

int main(int argc, char *argv[])
{
int flags[256]={0};                  // flags for file creation
char filename[256];                  // filename storage
int i;                               // loop counter
double tempf;                        // temp float var
long tempi;                          // temp integer var


if(argc != 2)
{ printf("Usage: data family\n\n"); exit (-1); }

get_geoInfo(argv[1],&nl, &ns, &nb, &ulx, &uly, &pixsiz);

//---------- Read Histogram Information ----------//
sprintf(filename, "%s.histo", argv[1]);
files[0] = fopen(filename, "r");
if(!files[0])
{ printf("Error opening %s\n\n", filename); exit(-1); }
for(i = 0; i < 256; ++i)
{
   fscanf(files[0],"%d %d %f%%", &tempi, &flags[i], &tempf);
   if(flags[i])
           filecount++;
}
fclose(files[0]);

//---------- Open Files ----------//
```

```
sprintf(filename, "%s.img",argv[1]);
files[0] = fopen(filename, "r");
if(!files[0])
        { printf("Warning: Error opening %s\n",filename); }
tempi = 1;          // opening "the" output as files[0]
i = 0;
while(tempi < filecount)
{
 if(flags[i])
 {
  sprintf(filename, "%s.%03d.img",argv[1],i);
  files[tempi] = fopen(filename, "r");
  if(!files[tempi])
        { printf("Warning: Error opening %s\n",filename); }
  tempi++;
 }
 i++;
}

//---------- Tcl Goo ----------//
interp = Tcl_CreateInterp();        // Create a new Tcl interpreter
Tcl_Init(interp);               // inititalize Tcl
Tk_Init(interp);                // initialize Tk
Blt_Init(interp);               // initialize BLT
// Tcl_SetVar(interp, "argv", argv[1], (int)NULL);
Tcl_SetVar(interp, "argv", argv[1],0);
        //copy argv[1] from C to argv in Tcl interpreter

if(Tcl_EvalFile(interp, SCRIPT) != TCL_OK) //load the Tcl script
{ printf("Tcl Error with %s:\n%s\n\n",SCRIPT,interp->result);exit(-1); } // exit if error

Tcl_TraceVar(interp, "Ycoord", TCL_TRACE_WRITES, Proc, (ClientData)NULL);
        // When Ycoord changes, it's time to read stuff from file
        // we don't need to trace Xcoord-- if one is changed, both are

Blt_GetVector(interp, "Yvec", &tcl_Yvec);
        // Get pointer to Yvec in interpreter for use later

Tk_MainLoop();              // Display GUI and start grabbing events
}

/*-----------------------------------------*\
\* Proc                                    */
/* procedure called when coordinates change *\
\*-----------------------------------------*/
char *Proc(ClientData data, Tcl_Interp *interp, char *name1, char *name2, int flags)
{
int i;
char *moo;
double tempd[26];
moo = Tcl_GetVar(interp, "Ycoord", flags);
sscanf(moo, "%d", &line);
moo = Tcl_GetVar(interp, "Xcoord", flags);
sscanf(moo, "%d", &samp);
if(line >= nl)
  {printf("%d invalid line number\n",line);return ""; }
```

```
if(samp >= ns)
  {printf("%d invalid sample number\n",samp);return ""; }

//----- call the band query routine -----//
point_query(files, vector, line, samp, ns, filecount);

//----- the BLT vector stuff -----//
for(i = 0; i < filecount; ++i)
  tempd[i] = (double)vector[i];
if(Blt_ResetVector(tcl_Yvec, tempd, filecount, filecount, TCL_VOLATILE) != TCL_OK)
            printf("Problem resetting vector...\n");
moo = (char *)malloc(256);          // 256 bytes should be enough...

sprintf(moo, ".status configure -background $colors(%d.0)",(int)tempd[0]);
if(Tcl_Eval(interp, moo) != TCL_OK)
{printf("Tcl Error, fix me\n\n");exit(-1);}

sprintf(moo, "set statusmsg \"( %d , %d )\nClass: %d\"",samp, line, (int)tempd[0]);
if(Tcl_Eval(interp, moo) != TCL_OK)
{printf("Tcl Error, fix me\n\n");exit(-1);}

free(moo);
return "";
}

/*----------------------------*\
\* point_query                */
/* Reads band info from files *\
\*----------------------------*/
void point_query(FILE *files[], unsigned char outvec[], long line, long samp, long ns, int count)
{
int i;
long offset = line*ns+samp;

for(i = 0; i < count; ++i)
{
 if(files[i])
 {
  if(fseek(files[i],offset,0) != 0) { outvec[i] = 0; }
  else
  {
   if(fread(&outvec[i],1,1,files[i])!= 1)
    {
//           printf("Error reading from file %d\n",i);
            outvec[i] = 0;
    }
  }
 }
} // for i
} //point_query
```

References

Flynt, Clif., 1999, *Tcl/Tk for real programmers*: Academic Press.

Foster-Johnson, Eric., 1997, *Graphical applications with Tcl & Tk*: M&T Books.

Harrison, Mark., 1997, *Tcl/Tk tools*: O'Reilly & Associates, Inc.

Osterhout, John K., 1994, *Tcl and the Tk toolkit*: Addison-Wesley Publishing Company, Inc.

Raines, Paul, and Tranter, Jeff, 1999, *Tcl/Tk in a nutshell*: O'Reilly & Associates, Inc.

Welch, Brent B., 2000, *Practical programming in Tcl and* Tk: Prentice-Hall, Inc.

Appendix B: A User Guide to the Beowulf Resampler

NAME

resample - create a new image by resampling the input image. *Note that this information is for the single-processor implementation. These options also apply to the Beowulf resampler that is discussed later in this document.*

SYNTAX

resample in_img out_img model_file resampling_type [-b "band_list"]
[-a cc_alpha] [-t weight_table] [-f fill_pixel]
[-d output_data_type]

where in_img is the input image name (including filename extension)
out_img is the output image name (including filename extension)
model_file is the mapping model file (including filename extension)
resampling type is the type of resampling from the following list:
CC = cubic convolution
BI = bilinear
NN = nearest neighbor
TABLE = user-provided weight table

OPTIONS

-a Cubic convolution alpha value. Defaults to -0.5 if not specified. Commonly used values are -1.0 and -0.5. Values between -3.0 and 3.0 produce reasonable results.

-b Comma or space delimited list of band numbers to process. Defaults to all the bands in the input image if not specified.

-d The data type of the output image. Can be:
BYTE (8-bit unsigned integer)
I2 (16-bit signed integer)
I4 (32-bit signed integer)
R4 (single precision floating point)
If not specified, the output data type defaults to be the same as the input data type.

-f Pixel value to use when an output pixel cannot be mapped back to the input image.

-t Name of the resampling weight table to use. Required when resampling_type is TABLE, and ignored otherwise.

DESCRIPTION

The resample application remaps an input image into an output image using a mapping model that defines the relationship to go from an output pixel location to an input pixel location. The resample application uses a single CPU.

MAPPING MODELS

Two models are supported to map output pixels to their corresponding input pixels. The two models are the grid and polynomial models.

The grid model defines the mapping function using a grid of evenly distributed output pixel coordinates and their corresponding input coordinates. This array is used to interpolate the location of all the output pixels in the input image. The grid file can be created using several applications available in the LAS package. These applications include ROTRNSCL, GRIDGEN, POLYFIT, REMAP, FINITE, and GRIDFORM.

One of the most commonly used gridding applications is REMAP. See the LAS documentation for the details on the gridding applications and the grid file format.

The polynomial model defines the mapping function using a polynomial equation to map an output pixel to a particular input pixel coordinate. Up to a fourth-degree polynomial with 15 terms is supported for both line and sample directions. The polynomial model file consists of a five-line ascii file. The first line simply reads "polynomial" to identify the file as a polynomial definition file. The second line contains a single number defining the degree of the polynomial. The third line contains the terms of the equation to calculate the line coordinate of the input pixel for a given output coordinate. The fourth line contains the terms of the equation to calculate the sample coordinate of the input pixel for a given output coordinate. The fifth line contains two numbers defining the quantity of lines and samples in the output image.

The coefficients on lines three and four of the file define these terms in order:

term	multiplier	highest order
1	1	1
2	x	1
3	y	1
4	x^2	2
5	xy	2
6	y^2	2
7	x^3	3
8	(x^2)y	3
9	x(y^2)	3
10	y^3	3
11	x^4	4
12	(x^3)y	4
13	(x^2)(y^2)	4
14	x(y^3)	4
15	y^4	4

Where x is the output sample number and y is the output line number.

An example of a simple polynomial file is:

```
polynomial
1
0 0 1
0 1 0
1000 2000
```

This defines a first-degree polynomial that maps each pixel in the output image to the exact same coordinate in the input image.

input_y = 0 + 0 * x + 1 * y, and
input_x = 0 + 1 * x + 0 * y

RESAMPLING METHODS

Four resampling methods are supported: cubic convolution (CC), nearest neighbor (NN), bilinear interpolation (BI), and user-supplied table (TABLE).

The only method that should need explaining is the user-supplied table method. The table should be created using the LAS RTABLE application.

IMAGE FILES

The resampler supports images in a binary flat file with the bands stored sequentially. Along with the image file, it expects to find two additional files

with the same base filename and .geoinfo and .proj as the file extensions.

The .geoinfo file contains a single line with seven numbers representing the number of lines in a band of the image, the number of samples in a band of the image, the number of bands in the image, the upper-left Y coordinate, the upper-left X coordinate, the pixel size, and the data type of the image. The coordinates and the pixel size are in whatever units make sense for the given projection. The data type is an integer indicating the data type stored in the image (1=byte, 2 = 16-bit integer, 3 = 32-bit integer, and 4 = single precision floating point).

The .proj file contains information defining the USGS projection of the image on two lines. The first line has four numbers representing the projection number, the zone number (62 if no zone number), the datum number, and a number indicating the units for the projection (0 = radians, 1 = feet, 2 = meters, 3 = seconds, 4 = degrees, 5 = packed degrees/minutes/seconds). The second line has the standard 15 projection parameters.

EXAMPLES

resample in.img out.img grid.grid CC -a -0.5 -b 1,2

Creates out.img by resampling bands 1 and 2 of in.img using the cubic convolution resampling method with an alpha of -0.5 and the grid model defined by grid.grid.

resample in.img out.img same.poly TABLE -t sinc.rwt -f 1

Creates out.img by resampling all the bands found in in.img using the user-defined weight table sinc.rwt and the polynomial model defined by same.poly. The value of 1 is used for fill pixels.

NAME

mpiresample - a version of resample implemented using the MPI library to take advantage of a Beowulf cluster.

SYNTAX

mpirun N mpiresample <...same options as resample...>
mpirun -np # mpiresample <...same options as resample...>

where 'N' indicates one process should be run on each node and "-np #" indicates the number of processes to run (with # being the number). The number of processes can be more than the actual number of nodes in the cluster. The extra processes are evenly distributed across the nodes of the cluster. This can improve run times if the options are CPU bound.

Note: mpirun is the LAM utility to run MPI programs.

OPTIONS

The options available are identical to the resample options.

There are many options for mpirun. See the mpirun manual page that came with your MPI distribution for more information.

DIRECTORY REQUIREMENTS

The current implementation of mpiresample requires the model file, the input image .geoinfo and .proj files, and resampling weight table (if required) to be available to all the nodes in the cluster via an NFS-mounted directory. The actual mpiresample application also needs to be available to all nodes in the cluster by means of an NFS-mounted directory.

EXAMPLES

mpirun N mpiresample in.img out.img grid.grid CC -a -0.5 -b 1,2

Creates out.img by resampling bands 1 and 2 of in.img using the cubic convolution resampling method with an alpha of -0.5 and the grid model defined by grid.grid

mpirun -np 20 mpiresample in.img out.img same.poly TABLE -t sinc.rwt -f 1

Creates out.img by resampling all the bands found in in.img using the user-defined weight table sinc.rwt and the polynomial model defined by same.poly. The value of 1 is used for fill pixels. Twenty processes are applied to the problem.