

EDUCATIONAL BENEFITS OF BEOWULF CLUSTERS

*John Oleszkiewicz, undergraduate
Grand Valley State University
Allendale, Michigan
oleszkij@student.gvsu.edu
Advisor: Professor Greg Wolffe*

INTRODUCTION

A Beowulf supercomputing cluster is an invaluable educational resource. A cluster in a college environment acts as a fountainhead from which interesting ideas, learning and research opportunities spring forth. This paper will give a sampling of the possibilities a cluster offers. Specifically, it provides examples from three categories: special projects and research, classroom learning and system administration. These examples are based on experience gained from constructing, maintaining, and developing special projects on Los Lobocitos, the resident Beowulf cluster of Grand Valley State University's computer science department.

SPECIAL PROJECTS AND RESEARCH

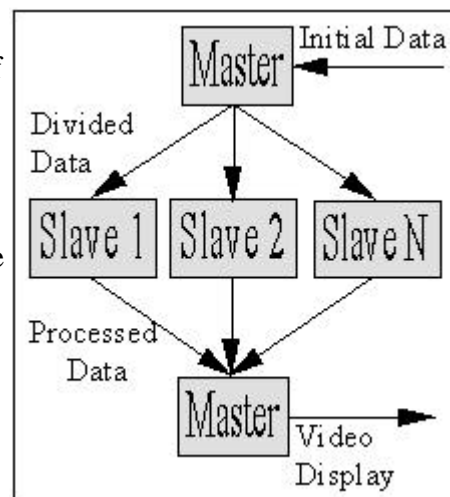
Project Examples

Two projects that used Los Lobocitos as their platform of development and execution serve as examples for the type of projects that can be accomplished on a Beowulf cluster. Each project was executed by an undergraduate student under the direction of a faculty advisor. The first project is a small scale introduction to the concept of parallel programming and its application to computer graphics. The second project is considerably larger and will be discussed in more detail.

Parallel Julia Set Generation

The first project is a parallel Julia set generator. Each node in the cluster receives a set of data consisting of the pixel where the process must start its computation and the number of pixels for which the process must calculate colors. Each node calculates its set of colors, places its results in an array, and ships the array to the master node, where the resulting image is displayed. This process is illustrated in the diagram on the right.

The Julia set problem belongs to a class of problems that are called "embarrassingly parallel" because it can be parallelized very easily. Portions of the data space are handed out to each node and they work on their own share independently. The only communication required is between the master node and the slave nodes for initial distribution of unprocessed data and the subsequent return of calculated colors for the final display. Given a rudimentary knowledge of graphics applications (e.g. OpenGL) and Julia sets, this project is a good introduction to the process of parallelizing serial algorithms and data structures.



Parallel Modeling of Fluid Dynamics

The second project is considerably larger. It is an ongoing research program in collaboration with a physics professor from Western Michigan University. The project models the behavior of solid particles that are subject to hydrodynamic forces. It also provides visualization of the model. This project has applications in paper engineering; diffusion, dispersion, and sedimentation studies; and aerodynamic simulations.

The Problem Domain

The first phase of the project involved becoming familiar with the problem domain. Team members had to read and comprehend an extensive number of papers written on the subject of fluid dynamics and related areas. Because these papers were written for other physicists, a significant amount of discussion and interpretation was required before proceeding on major features of the project. Such discussions were lengthy and took up large portions of time in the design phase. However, they were necessary to write a quality application and educational for all team members.

The Development Model

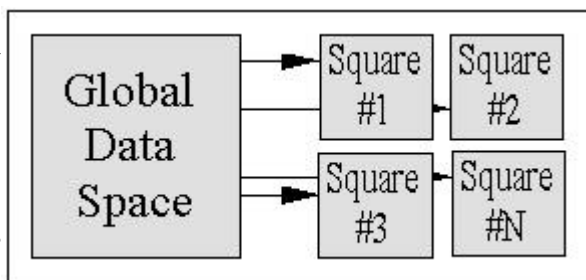
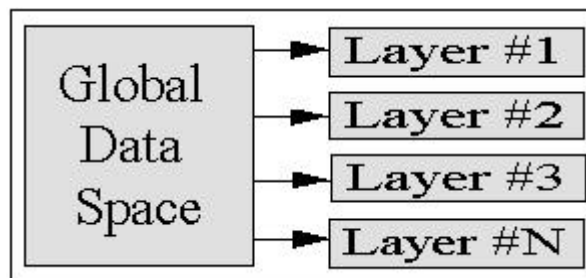
Development followed an iterative, rapid application development model. With each iteration more features were added to the serial implementation of the model so that it could solve more aspects of the problem. When major milestones were reached the serial model was converted to a parallel implementation. An easily understandable serial version was constructed first to help facilitate the construction of the more-complex parallel version. Other parallel projects may follow a different approach. No matter which way it is done, students are introduced to the ideas of design and development models.

Parallelization

Designing parallel data structures and algorithms is not easy. At the time of this writing, there are two different designs for the parallelization of the data structures of our model - each aims for different goals. The first is designed for simplicity, uniformity, and minimal messaging. It comes with the tradeoff of fewer messages, but with a very large amount of data in each message.

In brief, it takes a rectangular data space, divides it into equal sized layers and distributes each layer to a processor. The figure on the right visualizes this division. Each square node of area in the data space must be able to communicate with all of its neighboring nodes (whether they belong to the same layer or not). Each layer must communicate with an adjacent layer to keep track of fluid flow, and each layer must occasionally communicate with the master process for output purposes.

In an alternative design, the data space is divided into roughly equal sized squares. Each square goes to a processor. While this design is more complex compared to the layered method, each processor's data space has a smaller "perimeter". With a smaller



perimeter, a smaller amount of inter-process data needs to be transmitted. This design maximizes the amount of data that is communicated "internally" and minimizes the amount of data that has to be communicated "externally".

These examples illustrate that design work will help students learn about trade-offs. More complex designs can facilitate better performance. Simpler designs make project maintenance easier.

Implementation

Implementing these designs requires knowledge of the parallel environment: MPI/LAM. Knowledge of MPI consists of learning the library of C functions available, becoming familiar enough with the MPI paradigm to use them appropriately, and setting up and using the LAM parallel environment. Most of the implementation work is similar to any large programming project except for some special considerations. For instance, debugging multiple processes on multiple machines is difficult. Parallel debuggers are still developing and traditional serial debuggers can only debug one process at a time. The execution of parallel applications can also be problematic. If one machine is particularly slow in updating its filesystem, it could attempt to execute a slightly older version of the current parallel application while every other node executes the current version. The resulting bizarre behavior is a problem unique to clusters. This kind of implementation challenge will not be found on a traditional serial project.

Miscellaneous

Most large parallel projects require other peripheral tasks to be performed. For instance, our model generated huge volumes of data as output. To save writing time and disk space simple compression routines were written. In order to make sense of all of the data a companion visualization application was created. The former task required the analysis of the output data and how best to condense its storage footprint. The latter meant gaining knowledge of the OpenGL graphics library and designing a visualization scheme to make a rough, useable program. Neither of these tasks are directly involved in parallel programming, but were vital to the overall usability of the project.

Educational Opportunities of Projects

The first project is an example of the type of mildly challenging, small-scale work undergraduates could do to expose themselves to parallel programming and its applications. Such projects could, for instance, be assigned in a computer architecture or graphics class.

The second project demonstrates that developing a major project on a Beowulf cluster exposes students to a wealth of new experiences. Typically there are a multitude of tasks from different domains that have to be done in order to successfully complete the project. While this summer project was primarily the work of one undergraduate and his faculty advisor, the work could have easily been divided amongst several individuals.

Regardless of the scale of a project, parallel programming is a challenge. For instance, there is an ever-present danger of race conditions and deadlock if the sequence of execution is not well thought out ahead of time. Students may neglect planning for traditional, serial programming projects but cannot afford to with a parallel assignment. Planning, they will find, is essential to avoid a nightmare of tangled communications and incoherent process interactions.

The design of parallel data structures draws knowledge not only from software

engineering but other fields such as mathematics. For example, mathematical analysis can be used to determine how to best divide up data for efficient processing. Students may question the usefulness of mathematical knowledge for computer science projects, but the design issues encountered in the data division problem make the need for mathematical skill obvious.

Inter-discipline Collaboration

A cluster is a resource for application development and research, but it is a resource not limited to the computer science department. Anyone who needs serious computing power to conduct research or solve complex problems can use the services of a cluster. This would include, but not be limited to: chemists, physicists, biomedical scientists and engineers.

People from these disciplines are not computer scientists by training and will require help designing, implementing and running their applications. This is a very exciting opportunity for computer science students (and professors) to learn about other domains of knowledge and apply that knowledge to create usable applications. A cluster, and assistance in using it, can establish relationships between departments and start mutually beneficial collaborations.

Interdisciplinary projects, such as the fluid modeling project, demonstrate that computer scientists should not confine their knowledge to their discipline only. They should have the ability to grasp a wide variety of concepts from many domains. Without this ability, students will not be able to write software that is usable by those outside of the computer science community.

CLASSROOM LEARNING

Beowulf clusters are a valuable classroom resource. Architecture, networking, operating system and advanced programming classes all can benefit. A parallel computing architecture provides students with an environment in which they can study network operating systems, parallel programming techniques, and do performance tests and experiments.

Performance

One use of a cluster in the classroom is performance testing. Laboratory exercises can be devoted to the task of comparing the execution of serial code to the execution of parallelized code. Students might study what kind of speedup a parallelized algorithm offers. They could also discover how changes to the algorithm can speed up or slow down execution time. A cluster illustrates important concepts such as "doubling processors does not mean doubling execution speed".

Another classroom exercise might involve measuring the latency, bandwidth, and transmission speed of the cluster's communication network. These types of measurements yield information that has a direct impact on the execution speed of parallel programs running on the cluster. Because the cluster has its own dedicated network, there is a greater amount of control over experimental conditions.

Parallelization

Using a parallel algorithm to implement a solution to a problem can be both new and interesting. Learning to think in parallel terms, parallelizing data structures and designing parallel algorithms are challenging territories for students to explore. Projects such as the Julia set generator previously mentioned give students a feel for the basics of parallel programming and the issues involved.

Construction

Given a small class and enough spare parts, students can use the cluster as a model and construct their own. If the cluster is not a dedicated production or development environment students can deconstruct and reassemble the cluster - perhaps with a different network configuration, more than one master node or some other interesting modification. With a little hand-holding, students learn the issues involved in constructing clusters without having to delve too deeply into the complexities of software setup and configuration.

SYSTEM ADMINISTRATION

A cluster is a resource that must be maintained to remain useful. Its mere existence generates a substantial amount of system administration duties. Maintenance and administration of a cluster is similar to that of a LAN and can give students valuable experience. Four major domains of work are explored here: hardware management, software platform maintenance, automation, and security.

Hardware Management

The machinery of a cluster can not be constructed and then neglected. It breaks down, shifts positions, grows larger, and otherwise changes over time. A student could be appointed the "cluster engineer" to handle these changes. The jobs a cluster engineer can expect are: hardware acquisition and replacement; space, heat, and power management; and most importantly, network management.

When the cluster grows, it must grow intelligently. If it does not, it will become harder to manage and use. Hardware acquisition is a long, involved process that includes research, the prudent use of monetary resources, and tough decisions. Hardware acquisition is also caused by component failure and the need for upgrades. This process occurs more often as the cluster grows and becomes more complex as the cluster becomes less homogenous. Dealing with this process helps future system administrators develop their system design skills.

Space is a valuable resource and it can not be wasted when constructing or adding to a cluster. The cluster engineer will have to plan the location of shelving, machines, wires, and peripherals, all within the constraints of the room available.

Because of space constraints, cluster nodes are packed tightly together. Because these computers are typically originally designed to be stand-alone workstations, the cooling systems of these machines are inadequate when placed in a cluster environment. The cluster engineer will have to devise a plan for cooling all of the machines.

Power management for a cluster consists of more than finding a plug in the wall for a power strip. Before a cluster can be safely "plugged in", finding the total wattage of each machine either through a voltmeter or by consulting documentation is necessary. Only then can the total power requirements of the cluster be reliably calculated.

Networking is a unique problem with Beowulf clusters. It can be divided into two major, related areas: cabling and topology. Cabling is a problem because the large number of wires and their length constrains the space in which the cluster can be placed and limits signal quality. Topology is a consideration because of its effect on performance.

Each computer will use a power, mouse, keyboard, video and potentially multiple network cables. It is difficult to service a cluster when there is a cascading

mess of wires covering everything on the cluster's backside. The cables quickly become unmanageable if a coherent labeling and bundling plan is not thought of ahead of time. Another factor is cable cost. It is cheap and educational for students to create custom cables from a box of cable parts and crimping tools.

When Los Lobocitos was altered from its original configuration, lengthy cables were ordered to maximize flexibility in the positioning of the master node - which introduced the problem of signal degradation. The cables are so long that the video signal degrades as it travels from the master node to the KVM switch and back to the very nice, expensive monitor. Thinking about the immediate wiring needs of the cluster, possible future needs, and the optimal balance between length and performance helps the cluster engineer's management and planning abilities.

Network topology is another major area of focus in clusters. The cluster's topology may change over time depending on communication requirements. For example, a star topology is a simple solution that involves a minimal amount of effort. A hypercube configuration introduces more channels of communication, increasing the number of wires and NICs that have to be managed. Students can research possible topologies, determine the one best suited for the cluster's immediate and future needs, implement it, and test the results.

Software Platform Maintenance

Students get experience in basic configuration of servers in system administration classes, but a Beowulf cluster illustrates the demands of managing a production LAN. Software configurations change over time by upgrading software packages, introducing new ones, removing old ones, and tweaking the existing packages. Every machine in a cluster must be able to work with the other machines; a misconfiguration could potentially bring the entire system to a halt.

Maintaining the software on a cluster consists of administrative work multiplied by N nodes - each of which is potentially dependent on other nodes. A student "cluster configurator" can learn the skills necessary to identify the needs of the users of the cluster (and the needs of the cluster itself) and successfully implement fulfillment of those needs.

For instance, Los Lobocitos' master node required an MPEG player for a project. MPEG players for Red Hat 6.2, the master node's OS, were difficult to find. The MPEG players that were found did not perform adequately. It was decided that upgrading the master node to RedHat 7.1 would be a good idea because 7.1 comes standard with several mpeg players. This was accomplished. However, the SSH (Secure SHell) client/server package was upgraded as a side effect of the installation. The slave nodes ran SSH protocol version 1, and the master node now used version 2. The two protocols are not compatible. Los Lobocitos uses SSH for all of its internal communications, so parallel programming and parallel administration could not be done until the problem was fixed. To remedy this situation, it was decided to upgrade the slave nodes to RedHat 7.1. However, the installer crashed on the slave nodes and they could not be upgraded. Later, it was discovered that the SSH server could be forced to use protocol version 1 via command line arguments given to the client. The SSH client was aliased with those arguments for every user. Later still, it was found that the configuration file of the SSH server could be altered so that version 1 was used by default - negating the need for the alias.

What seemed like a relatively simple task (finding a good MPEG player and configuring it) became an ordeal. This type of experience will challenge the students maintaining the cluster software to acquire advanced system administration skills.

Automation

The volume of administration and maintenance work on a cluster is N times as large as on a single system. It is possible to manually perform the necessary tasks on each node but doing so is time-consuming. On small clusters manual maintenance is an inconvenience. On large clusters it is impractical. Finding ways to automate tasks that need to be carried out on each node is extremely important. Every cluster is different and most automations will have to be custom-fit.

The job of automation encompasses identifying tasks which need to be carried out on each node and designing a way of carrying out those tasks with minimal human interaction. Finding a comprehensive, uniform solution to all of the cluster's automation needs will probably not be possible. What can potentially be automated is without end; the requirements for the automation change over time and better ways to do the same thing can always be found.

One example is the locally-developed "all_ping" script. Los Lobocitos needs to poll nodes at arbitrary times to see which are turned on. It is possible to visually check which nodes are up - but such a solution takes time and requires the operator to be physically present. Our original solution was a bash script that used the "ping" command to determine which nodes are up. Successive versions added new features - but one inefficiency remained: each node was pinged in sequence. The latest version of "all_ping" is written in C and uses forks and execs to ping nodes in parallel.

This example shows that automation is a continuous, evolving process. There are unlimited opportunities in cluster automation programming for interested students. Automation can be done in any language. All computer science students can start automation projects in their favorite language. The more work a student is willing to put into the automation, the better it will perform. Automation programming is a wonderful way to introduce students to real-world, usable programming.

Security

A cluster connected to the internet or a campus LAN presents a good opportunity for students to study and implement computer security. Networked computers are vulnerable by virtue of being connected and a cluster may attract more than its fair share of curious joy-riders and system crackers looking for some serious computing power. Security is a never-ending battle and new developments in the field provide students with an unending supply of real-world security experience.

A student security expert creates, implements, and maintains a security plan. To create the plan, the student will have to answer some questions. How much can each node really trust the others? If one node is compromised, how easy will it be to compromise others via the private network? Is it possible for hostile machines to hook up to the private network? What is preventing people from wandering up to the cluster and playing with nodes (e.g. powering them on or off)?

The implementation of the security plan can prove to be both complex and time-consuming. Aside from understanding the security features of various services, the expert needs to research and choose among various security software packages. Security packages (e.g. tripwire, bastille, etc.) aid in the security rule setup, integrity checking, parsing of log files and generally make life easier for those who are concerned with security.

There are also regular "security maintenance" tasks to perform. A typical day in the life of the cluster security expert could include: examining the log files of each node in the cluster (especially nodes connected to the internet), using an integrity checking system such as tripwire to see if any configurations have been changed without

authorization, searching for strange files (e.g. SUID/GUID executables), investigating any unusual activity, checking BUGTRAQ or CERT advisories for new exploits found in services being run on the cluster and downloading patches that correct problems.

All computer networks require security measures. The need for security in business networks is acute. Students will have a lot to offer potential employers when they are finished with their cluster "tour of duty".

Conclusion

A Beowulf cluster's intended purpose is to solve problems in parallel. But it offers so much more than that to educational institutions. Its mere presence generates opportunities of all kinds: opportunities in the administration and growth of a production system, opportunities for learning in the theory and practice of parallel programming, and opportunities in major research and development. The example projects and learning opportunities listed here are only the tip of a very large iceberg. What one can do with a cluster is limited only by the imagination. What one can learn from a Beowulf cluster is without bound.