Distributed Processing of Projections of Large Datasets: A Preliminary Study

By Brian G. Maddox[1]

Open-File Report 03-117

[1] Mid-Continent Mapping Center, Rolla, MO 65401

U.S. Department of the Interior
U.S. Geological Survey

Contents

Tables

Figures

Key Words

Distributed Processing, Large Datasets, Beowulf, Map Projections

Abstract

Modern information needs have resulted in very large amounts of data being used in geographic information systems. Problems arise when trying to project these data in a reasonable amount of time and accuracy, however. Current single-threaded methods can suffer from two problems: fast projection with poor accuracy, or accurate projection with long processing time. A possible solution may be to combine accurate interpolation methods and distributed processing algorithms to quickly and accurately convert digital geospatial data between coordinate systems. Modern technology has made it possible to construct systems, such as Beowulf clusters, for a low cost and provide access to supercomputer-class technology. Combining these techniques may result in the ability to use large amounts of geographic data in time-critical situations.

Introduction

It is becoming common for geospatial applications to require very large amounts of data. Projects may need multiple-band, remotely sensed data with file sizes measured in gigabytes. Or, they may need hundreds of megabytes of tiled aerial photographs for their study areas. True decision support systems also require enormous amounts of data and processing facilities to perform their activities.

The problem with using this much data is that as file sizes get larger the processing becomes more difficult. This can be especially problematic in hazards-response situations, where digital data need to be quickly and accurately projected between coordinate systems within a bounded time frame. Some scientists working for hazard response during Hurricane Mitch found that large amounts of data often took very large amounts of time to project to different coordinate systems. A problem encountered was that any errors required another period of time to process, and hurricanes do not slow down for processing.

Modern technology now provides low-cost systems with supercomputer-class processing capabilities. Systems such as Beowulf distributed processing clusters can be constructed out of commodity hardware and give users huge amounts of processor and input/output (I/O) handling facilities. Commodity hardware includes standard off-the-shelf components that are less expensive than those required for traditional high-end workstations. Such a system would seem ideally suited for processing large amounts of geospatial data.

This paper describes a preliminary effort in speeding up the projection of gigabyte-sized data while keeping the accuracy within reasonable tolerances. The work concentrated on distributing pure pixel-by-pixel processing and interpolations over a Beowulf cluster. Issues with this type of processing are also discussed for those who may be interested in implementing such a system.

Background

The use of interpolations in projecting geospatial data is nothing new. The mathematics of map projections can be quite complex, and interpolations are often used to find functions that are easier to calculate and close approximations of the original equations. Software today typically uses both one- and two-dimensional mesh-based interpolations. The two-dimensional interpolations have accuracy on their side but can be slow. One-dimensional interpolations can be very fast but can also leave the projected data lacking in accuracy.

The target data for this study had a one-meter resolution, and it was therefore decided to ignore pure one-dimensional and nearest neighbor methods. These interpolation methods generally have problems fitting a surface through multi-dimensional data. To examine the accuracy of two-dimensional methods, several common interpolators were chosen: bilinear, bicubic, least squares plane, and bicubic splines. These interpolations will be

briefly described here, as more in-depth explanations can be found in most numerical methods texts.

Bilinear interpolation is a one-dimensional linear interpolation performed in two directions. Each directional linear interpolation performs its approximation by passing a line segment through known points. It has the benefit of being computationally simple. However, bilinear interpolation generally requires a denser mesh of input points to achieve higher accuracy.

Bicubic interpolation, in contrast to bilinear interpolation, is a pure two-dimensional interpolation that fits a cubic surface through the input points. Although more accurate, bicubic interpolation suffers from increased computational intensity because it requires a grid of points to solve for the surface coefficients. In addition, bicubic interpolation can suffer from stability issues as the mesh of input points becomes denser.

Least squares plane finds the statistical minimum error plane fit to the input points. It is a relatively fast interpolation to compute, but it is inaccurate in this situation as many of the projection functions are nonlinear. It can also suffer the same stability issues of bicubic interpolation with dense meshes.

Bicubic splines interpolation is performed in much the same way as bilinear interpolation, except that instead of linear functions, the interpolation uses cubic splines to interpolate within an area. Cubic splines are functions defined on a smaller interval where "the resulting piecewise curve and its first and second derivatives are all continuous on the larger interval" (Matthews 1992). Cubic splines have the advantage of producing smooth, second-order continuous functions. However, to achieve this continuity, cubic splines must identify the second derivatives at the two end points of the spline. Since these second derivatives are unknown for a given map projection, they must be estimated to perform the interpolation, leading to further inaccuracy. Additionally, though cubic splines are guaranteed to pass through each input point, the spline itself may curve radically between points, often producing significant error in order to satisfy the continuity constraint.
Parallel processing is defined to "specifically refer to the simultaneous execution of concurrent tasks on different processors" (Nichols 1998). It is important to note from the definition that concurrency is a requirement in parallel processing. Concurrent processing is defined as "environments in which the tasks we define can occur in any order. One task can occur before or after another, and some or all tasks can be performed at the same time" (Nichols 1998). In other words, parallel processing involves tasks that can not only run at the same time, but that can run independently of order. This is important, as there are typically no ways to guarantee that the operating system will schedule tasks to run in any specific order. Without the concurrency requirement, an application would run parts of itself at the same time but produce incorrect results.

Distributed processing is a type of parallel processing that involves using multiple computers or processors to run an application. It can be defined as the concurrent execution of multiple tasks on multiple processors over multiple computers. This usually involves taking a large problem and breaking it down into several smaller problems. These smaller problems are then run concurrently on multiple machines with the goal of

finishing the task in a shorter amount of time than it would take a single machine. These machines are usually connected through their own private high-speed network to minimize network traffic that could interfere with processing.

Distributed processing has some advantages over traditional parallel processing in multiprocessor machines. It can offer more memory, processor capability, and permanent storage space than a typical multiprocessor system can. In a multiprocessor machine, each processor can operate concurrently, but they have to share common systems in the machine, such as memory and permanent storage systems. Only one processor or software process can access many of these systems at a time; all of the others must wait until the resource becomes available. In a numerically intensive environment, this could mean a large number of processes are left waiting in line. In a distributed processing environment, the processing nodes have independent processor, memory, and other such systems in each machine. The resource contention in each node is reduced, as there are a smaller number of processes running on each node that may need access to various resources.

The primary disadvantage of distributed processing comes from all inter-node communication occurring over the network. This is because the network can only communicate at speeds that are an order of magnitude slower than the system bus provides. While this may not be a problem for compute-bound applications, it can become a major problem for data-bound applications. Communication over a network is prone to *collisions*, where multiple machines try to transmit at the same time. When this happens, all machines must resend their information over the network. This can happen several times before a computer can successfully transmit its information. Factors such as these tend to force software designs where communication between nodes must be minimal compared with communication between processors in a multiprocessor system.

A type of distributed processing system known as Beowulf clusters has become popular in recent years. Developed by researchers at the NASA Goddard Space Flight Center, a Beowulf cluster is "a kind of high-performance massively parallel computer built primarily out of commodity hardware components" (Sitaker 2000). This has the advantage that equipment costs are minimized, as no specialized hardware is used. It is very common to construct low-cost Beowulf-class clusters out of used or surplus equipment. These clusters also typically use a modified Open Source operating system to handle the interoperability between nodes. Communication is performed by passing messages between nodes on the network. The message-passing systems are set up so that almost any type of data can be encoded in the message packets.

There are two major message-passing systems in use today. The Parallel Virtual Machine (PVM) implementation was developed at the Oak Ridge National Laboratories as "a byproduct of an ongoing heterogeneous network computing research project" (Oak Ridge 2000). PVM is a software library that provides various functions to the user for passing messages in a cluster environment. PVM also provides features such as simple load balancing, transparent hardware access, and dynamic scalability for adding or removing nodes on the system. The other major method involves software libraries that implement the specification set by the Message Passing Interface (MPI) Forum. The MPI Forum is an "open group with representatives from many organizations that define and

maintain the MPI standard" (Message Passing Forum). The MPI is a standard that details how messages should be passed in a clustered environment. There are several libraries that implement this specification. The main difference between these two methods is that PVM provides more facilities to treat the cluster as a single working unit instead of a series of independent nodes.

Method

To test the accuracy of the system, a control had to be chosen for the generated geographic coordinates. The General Cartographic Transformation Package C (GCTPC) package was chosen as the control for this project. This package has been debugged and tested over the years and is used by many software packages. All points generated by the interpolators would be compared with those same points generated by GCTPC to determine the error over the dataset. The requirement for the interpolators was that the average error over the entire dataset must be less than one meter to match the accuracy of the input data.

The method used to increase the speed of the coordinate system transformation processing was to distribute the work of generating the output scanlines of the raster image. This method involved having each node generate an output scanline in the new raster image by performing a reverse projection through GCTPC to determine which pixel in the input image the node needed. This technique would have the greatest accuracy of all as it was performing the full projection equations on each cell in the output image.

Mathematical interpolations were also chosen as a means for generating geographic coordinates. Interpolation, which is used in many geographic information system (GIS) software packages, is an algorithmic method to quickly calculate conversions between coordinate systems. If used properly, interpolation can maintain accuracy within the tolerances of the data products used. The goal of this project was to determine if the combination of interpolations and distributed algorithms could derive a system that could project large amounts of data quickly and accurately. For this study, mesh-based interpolations were chosen as they allow smaller, local-area interpolations on different parts of the data.

Equipment

The equipment used for this study was the Beowulf cluster housed at the USGS Mid-Continent Mapping Center (MCMC). The cluster at the time of study was a refurbished sixteen-node system that was connected by a private 100-megabit Ethernet switch. Each node consisted of an AMD K6-2 500-MHz processor on an FIC VA-503+ motherboard with 128 megabytes of PC-100 system memory. The permanent storage in each node was upgraded to a nine-gigabyte U2W SCSI hard drive connected to a Tekram 395 U2W SCSI controller. Each machine kept its 100-megabit 3Com Ethernet network card and video card, as they had already been upgraded prior to being placed into surplus. They all ran a modified version of the RedHat Linux 6.2 distribution. This distribution contained the necessary PVM message-passing libraries for the cluster environment. The

modifications to the stock distribution included replacing the Linux kernel with the 2.2.17 version and using the vendor-supplied drivers from Tekram and 3Com.

Data

Control data had to be chosen for testing the accuracy of the processing. For the initial testing and debugging, several digital orthophoto quarter-quadrangles (DOQQ) of the Manhattan, Kansas, area were selected. These quadrangles have multiple Global Positioning System (GPS) verified control points in them and are used by MCMC for accuracy testing. These DOQQs were converted into GeoTIFF format and loaded into several GIS packages to confirm the accuracy of the control point data.

Sample data were also needed that would simulate real-world processing of large amounts of data. To fulfill this requirement, the GRASS GIS was used to generate a one-degree by one-degree mosaic of DOQQ data in the area covered by the control data. This mosaic was then converted to a TIFF file, with an accompanying Arc Info world file holding the georeferencing information, and was then converted into a 1.3-gigabyte GeoTIFF for testing. This GeoTIFF was loaded into several packages to verify the coordinates of the control points after merging the files.

Software

Experimental software had to be developed to perform the study. This software was initially designed as a single-threaded system. Debugging is far easier on a single-threaded system than on a distributed system as there is only one memory space that must be examined. The design for the interpolator engine was to use a grid-based system with GCTPC used to generate the grid points. An in-house developed mathematics library provided a series of interpolators to the software system. The software had the ability to either interpolate the chosen points or use GCTPC to project each point directly. This allowed comparisons between the GCTPC-generated points and those generated by the interpolations. It also allowed for time comparisons between distributing pure GCTPC-based projections and distributing interpolation-based projections.

The software was then redesigned to run over a cluster. Although there are several options available, the master/slave design was chosen for this initial study. This design utilizes a master node that will assign work units to be processed and receive the finished data. The slave nodes simply process any data assigned to them. This is a common design in parallel processing and also had the advantage of being easily implemented within the project's time constraints.

In this setup, the master node pre-computes the area of the output image. When it has the geographic and raster bounds of the image, it sends this information to the slave nodes so they can store the coordinates for future processing. The master node then dispatches the work units to the slave nodes. These work units are the y-values of the scanlines of the output image. The slave nodes will then use either interpolation or straight GCTPC calls to generate their assigned scanlines. They do this by performing a series of reverse projections. The coordinates are converted to raster space and then generated from the original image data. When finished, the slave nodes return their processed scanline to

the master node, which outputs it to a GeoTIFF file. If any work remains, the master node will continue to dispatch scanlines until the output projection is finished.

Testing

The first test of any interpolation system is to check its accuracy against several known values. Mesh-based interpolation of geospatial data works by defining a mesh of a fixed set of input points that are obtained by projecting them with GCTPC and then using a subset of these points to interpolate the rest of the values in each local area. Generally, the points within a grid cell are interpolated on the basis of grid-cell corner point values. The number of input points and the geographic size of the mesh then affect the accuracy and stability of the particular interpolator used. To get a good range of accuracy tests for each interpolator, a comparison test was constructed in which control points were projected first with mesh-based interpolation and then compared with the same point projected with GCTPC. After projection, the distance (Euclidean norm) was computed between the interpolator-projected point and the GCTPC-projected point. This test was repeated for each interpolator examined using several map projections, different mesh sizes, and various numbers of input grid points.

Table 1. Average interpolator error on a one-degree by one-degree area

Interpolator Type	Grid Size	Average Error in Meters
Bicubic	4x4	2.17E-04
Bilinear	500x500	2.70E-04
Least Squares Plane	500x500	3.76E-04
Bicubic	10x10	1.35E-03
Bicubic	100x100	1.51E-03
Bilinear	100x100	6.87E-03
Bicubic	500x500	8.81E-03
Least Squares Plane	100x100	9.56E-03
Bilinear	10x10	8.31E-01
Least Squares Plane	10x10	1.16E+00
Bilinear	4x4	7.48E+00
Least Squares Plane	4x4	1.04E+01
Bicubic Splines	4x4	1.24E+06
Bicubic Splines	10x10	1.79E+06
Bicubic Splines	100x100	2.03E+06
Bicubic Splines	500x500	2.03E+06

Table 1. Average interpolator error on a one-degree by one-degree area

Interpolator Type	Grid Size	Average Error in Meters
Bicubic	4x4	2.17E-04
Bilinear	500x500	2.70E-04
Least Squares Plane	500x500	3.76E-04
Bicubic	10x10	1.35E-03
Bicubic	100x100	1.51E-03
Bilinear	100x100	6.87E-03
Bicubic	500x500	8.81E-03
Least Squares Plane	100x100	9.56E-03
Bilinear	10x10	8.31E-01
Least Squares Plane	10x10	1.16E+00
Bilinear	4x4	7.48E+00
Least Squares Plane	4x4	1.04E+01
Bicubic Splines	4x4	1.24E+06
Bicubic Splines	10x10	1.79E+06
Bicubic Splines	100x100	2.03E+06
Bicubic Splines	500x500	2.03E+06

The above table is sorted from best to worst accuracy. It illustrates how well some of the interpolation methods are able to approximate the projection equations. The average error is mathematically calculated by taking the average error of the same area projected to different coordinate systems. The grid size is the mesh size that was used in the interpolation.

To assess the relative speed of each interpolator a similar test was formulated. In this test, a DOQQ from the Manhattan data was projected to different systems using various interpolators with a fixed number of input mesh points. The same DOQQ was then projected using GCTPC, and the time required to complete both operations was compared. These tests used the DOQQ files for the relative speeds as they allowed testing to be completed in a reasonable amount of time.

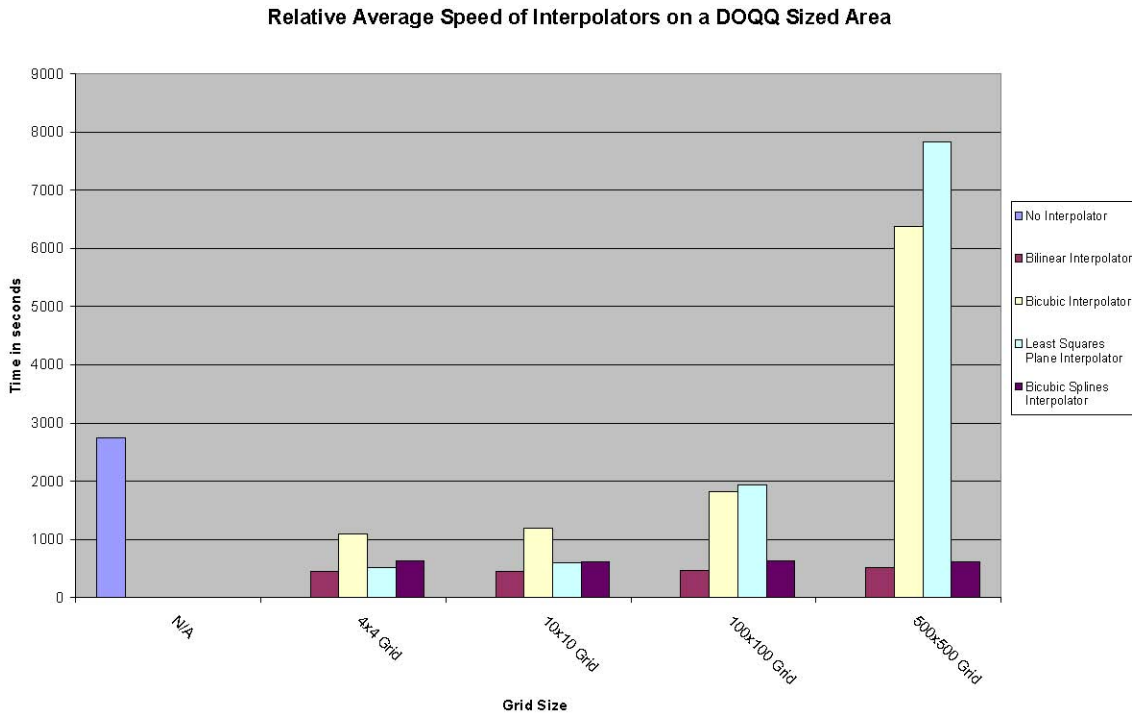

Figure 1. Runtime comparisons for a DOQQ.

Figure 1 shows the differences in the relative speeds of the interpolation methods for various projections. Each interpolation method differs in terms of mathematical complexity that becomes amplified as the numbers of mesh nodes increase. Bilinear interpolation, for example, scales fairly well with increasing grid sizes. Others, such as bicubic and least squares, tend to experience an almost exponential increase in runtime as the number of grid points is increased.

With the error and time results, a determination could be made as to which interpolation method provided the best accuracy and fastest runtime. For accuracy, bilinear and bicubic interpolations provide the lowest mathematical error. The deciding factors, then, are speed and scalability. As can be seen in Figure 1, bilinear interpolation scales very well as the grid size increases. Bicubic interpolation, while producing the lowest error, is also much more computationally intensive than bilinear methods. These issues led to the decision to use bilinear interpolation as the method of choice for all of the distributed time trials.

The next test compared the relative speed increase between the distributed and conventional designs in real-world data processing. In this test, the 1.3-gigabyte composite GeoTIFF of the Manhattan dataset was used as the input to the system. This image was projected to several different coordinate systems, and the time between distributed and conventional designs was compared. However, speed trials performed in this test used only the best (speedup vs. accuracy) interpolator, as well as GCTPC with no interpolation. Additionally, timings on the cluster were done with varied numbers of processing nodes to examine how the addition of processing nodes affected overall runtime. The following charts show the results of the time trials for the cluster, as well as for single-threaded applications.

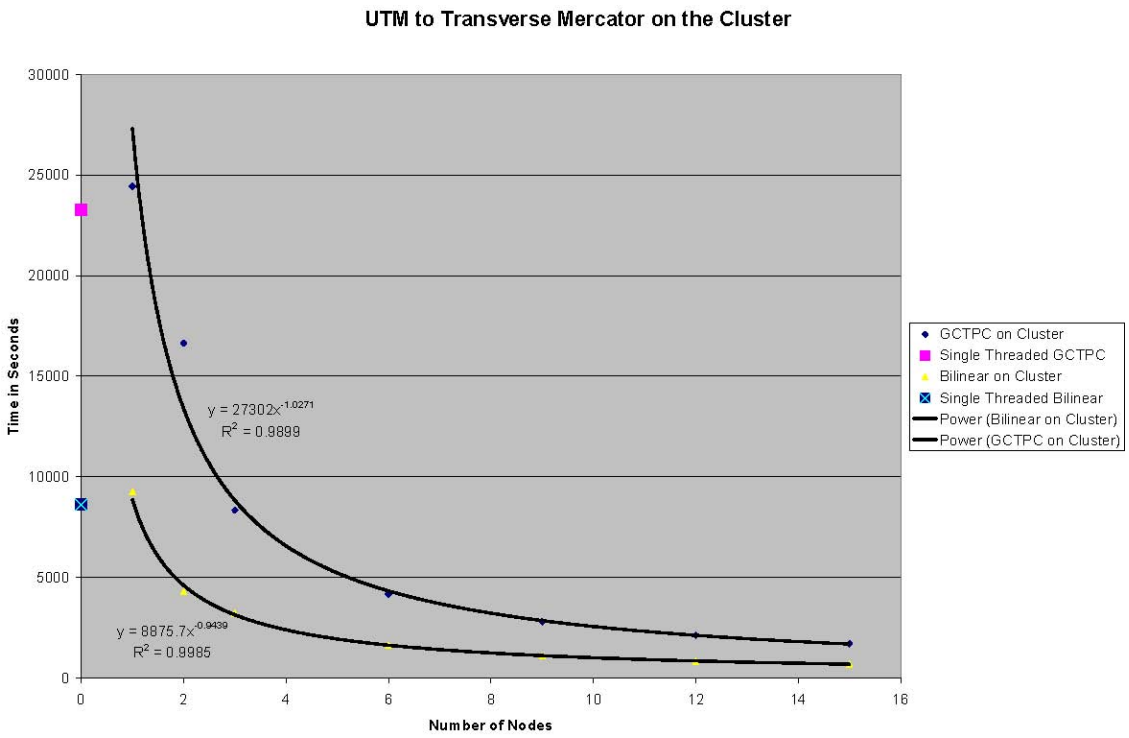

Figure 2. Timings for UTM to Transverse Mercator conversion.

Figure 2 shows the timings for performing a coordinate conversion from the Universal Transverse Mercator (UTM) to a different Transverse Mercator projection for the large GeoTIFF. The data points show the timings for various numbers of nodes, and the curves are power regression curves fit through the data points. Along the left edge of the figure, the timings for a single-threaded version of the conversion software are also shown. The curve fit functions allow for a rough estimate of the speed increase gained by adding a certain number of nodes.

Figure 2 illustrates some common traits of parallelism. Many data-bound parallel applications will follow such a curve, where the speed increases of parallelism begin to level off past a certain number of nodes. This directly follows the Law of Diminishing Returns (Cannan 2001), where the cost of adding more nodes will eventually begin to outweigh the benefits of adding more nodes. The figure also shows that a master node with one processing node will take more time to run than a single-threaded version of the software. This is a direct result of the overhead that is occurred by processing the data over the network and will be further discussed later in this paper.

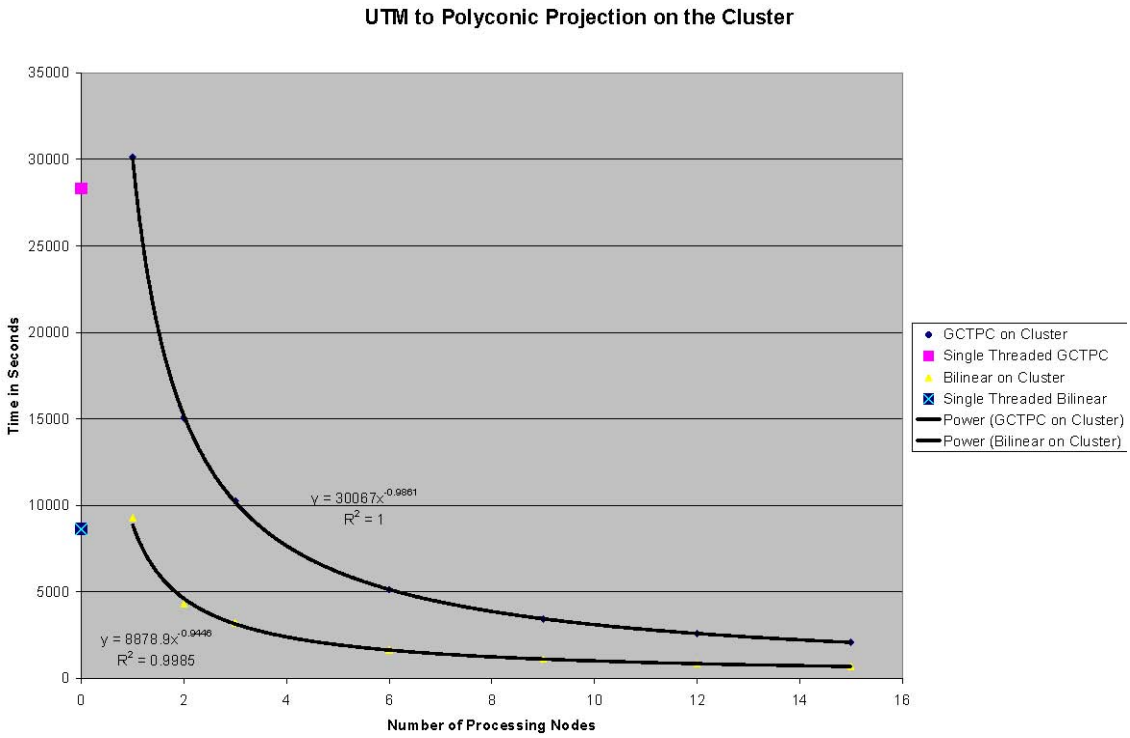

Figure 3. Timings for UTM to Polyconic conversion.

The data points in Figure 3 show that the GCTPC timings are different from those for the UTM to Transverse Mercator projection conversions. This is expected, as the pure GCTPC cell-by-cell conversions have to perform different mathematical equations for the different projection systems. These equations have differing levels of complexity that will affect the amount of time it takes them to run on a processor.

The bilinear interpolation, on the other hand, runs in almost exactly the same amount of time for both the Polyconic and the Transverse Mercator coordinate conversions. This is also expected, as the interpolations are running through the same series of mathematical approximations for both conversions. This is the benefit of function approximation. Instead of performing different equations for each projection system, the interpolator is merely performing a best fit to the input data, regardless of the type of coordinate system.

Discussion

Varying the number of nodes during processing produced power curves that allowed predictions of runtime improvements that were based on the number of nodes added. The next figure presents these data, and there are some interesting observations and comments to be made about increasing the number of nodes.

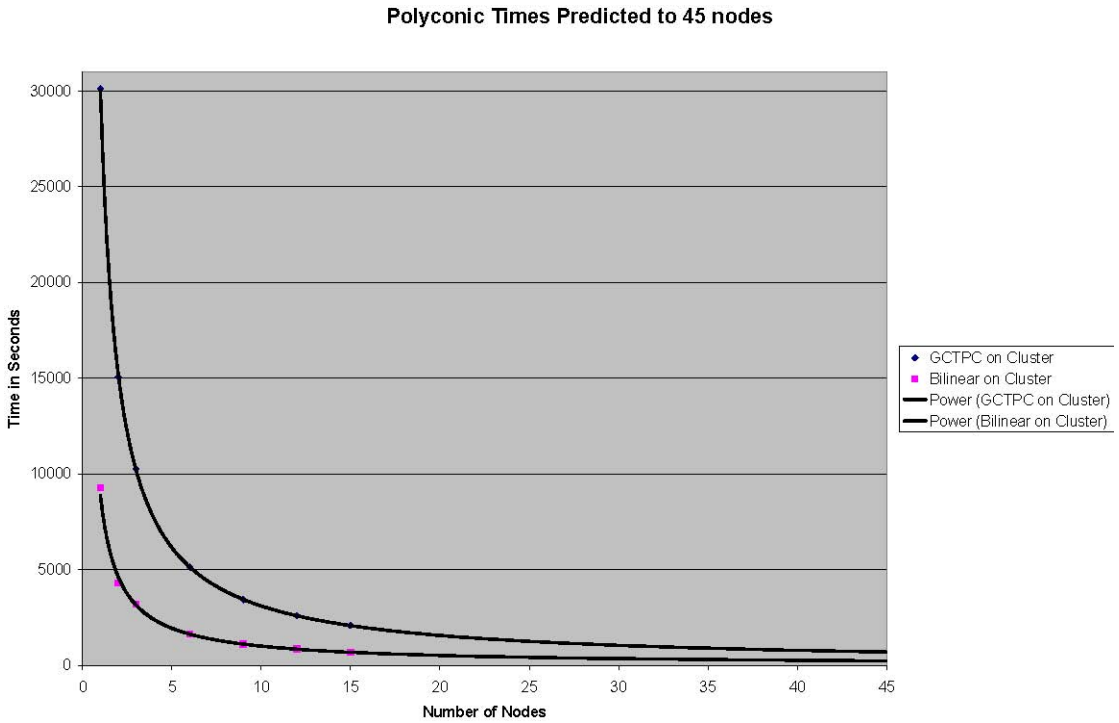

Figure 4. Polyconic runtime predictions.

As more nodes are added to the cluster, they begin to have a diminishing impact on the overall runtime of the system. Figure 4 shows the runtime curve for the conversion from the UTM to Polyconic coordinate systems predicted to forty-five nodes. The curves become nearly asymptotic as they approach a certain minimum runtime. They also both get fairly close to one another as more and more nodes are added. This suggests that given enough nodes, pure GCTPC-based projection conversions may be done in almost the same time that it takes to interpolate them. The curves, however, may well be misleading.

The graph of runtimes per number of nodes is more than likely that of a very flattened parabola than that of a power curve. The reason is that up to a certain point, adding more nodes will approach a minimum runtime and overpower the effects of network overhead. But as more and more nodes are added, the complexity of the system increases. The overhead from transmitting data over the network may begin to have a significant impact on the runtime. When a certain point is reached, the cumulative overhead may actually begin to increase the runtime of the system. A small-scale example of this may be when there is only one processing node combined with the master node. As Figure 3 shows, this setup is actually less efficient at processing than a single-threaded system owing to the increased network overhead.

Several questions arose when designing the distributed version of the software. The primary issue was how to efficiently distribute the processing. The main options were either to develop distributed versions of the map projection algorithms or to distribute the data processing among the nodes. Runtime was a major constraint on what method to use, but consideration of the problem produced other reasons not to distribute the map projection algorithms themselves.

Distributed processing works best when there is a natural partitioning of the problem and each partition can be processed in a relatively independent manner. This is the concurrency requirement discussed earlier. As an example, consider the problem of summing all of the numbers from one to 100 billion. A natural partitioning scheme here would be to break the problem into smaller sums and add all of the sums at the end. Ten nodes could each sum ten billion numbers and add the sums together at the end. The key is that while the nodes are summing their partition, they do not require any results from the other processing nodes. Only at the end is there a communication of results.

With the map projection equations, such a partitioning does not exist. The implementation of these projections is a chain of algorithms that are used to go from one coordinate system to another. There is a very tight coupling between the inputs of one algorithm and the results of the previous algorithm. Also, a lot of looping is involved in the code, where a loop pass depends on the values of the previous passes. This would seriously degrade performance, as the output from one part of an algorithm would need to be encoded and transmitted to another node that is performing the next part of the algorithm. It may even be the case that parallelizing the map projection algorithms on a multiprocessor machine may be inefficient due to communications overhead between steps. The overhead of transmission would be very great if done each time the individual algorithms are run.

Another problem was discovered during this study concerning network congestion. In the case of converting a DOQQ from the UTM system to a State Plane Coordinate System, the nodes were finishing their processing so quickly that they ended up flooding the network when trying to return their processed scanlines to the master node. Flooding is a term used to describe a condition in which a network is suffering from a massive number of transmission collisions. A collision occurs when multiple nodes try to send data over the network at the same time. In the case of an Ethernet switch, the other nodes must wait until the node they are trying to communicate with finishes its current transmission or reception. This resulted in such an extreme slowdown in processing that a single-threaded projection program finished much faster than the distributed version. Flooding was also observed when performing other coordinate system conversions and could have led to a processing slowdown in those cases as well.

There are several possible solutions to this problem, but none appear to be the proverbial "silver bullet." Multiple input/output nodes could be used for the data processing. Ethernet switches can support multiple node-to-node communications at the same time. Multiple I/O nodes would allow multiple processing nodes to return their data at the same time, but all of the other nodes would still be stalled as they waited to get a connection to an I/O node. Flooding would still be an issue, as some of the nodes would be vying with each other for access to their assigned I/O node. This also means that there would be fewer nodes available to do the actual processing work.

Another option would be for each node to hold its results until the master node requests them. This would work by storing the processed scanlines in a temporary file, as there would be too much data to cache them in physical memory. The problem with this method comes from the fact that the master node would suffer from data starvation while

trying to create the output file. This happens because the main node must wait for requested data to be sent from the slave nodes.

The fundamental problem is that efficiently processing large amounts of data over a network requires further study. Methods exist that work well in providing the capability to handle large numbers of I/O requests, but these methods often are not suitable for handling very large amounts of data that must be passed over the network. The problem arises because these methods typically split a file across multiple disks on multiple machines. This can cause multiple I/O machines to hold the parts that several processing nodes need, generating increased network traffic as there are more nodes that must communicate.

There were also some problems noticed with how the Linux 2.2 series kernel handled virtual memory. Most modern operating systems implement some type of virtual memory, commonly known as swap files. Virtual memory systems use permanent storage to free up physical memory by allowing the operating system to temporarily move processes from physical memory to the permanent storage device. These are generally processes that have not run in a long time or that are waiting for some other event to happen. Operating systems also use as much physical memory as possible to cache data that have been read from permanent storage. This allows faster access to the information if a process tries to read it again. The size of this cache usually shrinks as the memory demands of running processes increase.

In the case of the Linux 2.2 kernel installations on the cluster, the operating system seemed to be spending a large amount of time swapping processes in and out of virtual memory. It was observed that the disk cache constantly stayed at a very large size instead of shrinking as process memory demands grew. The resulting swapping sometimes took up a significant amount of processor time.

There is another relevant theory concerning processing large amounts of data. Dedkov and Eadline have proposed that given two multiple processor machines that are identical except one has slower processors, the slower machine will process I/O-dominant applications more efficiently than the machine with the faster processors. Although this may seem counterintuitive, it can be explained by understanding the inner workings of a computer. Everything in a computer is connected by a system bus, including the processors on a multiprocessor machine. Multiple processors must spend a lot of time synchronizing with each other over this bus. With high-speed processors, there is a great deal more traffic per time period than with slower processors. The problem arises because the data to be processed must also travel over this system bus, whether such data come from a network device or from a permanent storage medium. The greater amount of cross-processor traffic with high-speed processors reduces the amount of bandwidth that is available for passing data along the bus. This could result in data starvation as the high-speed processors end up waiting for data to reach them. Slower speed processors would not flood the bus as much and would therefore more efficiently process the large amounts of data passed to them.

Conclusion

Technology has finally reached the point where it can offer large amounts of computing power for the processing of digital geospatial data. By using modern technology such as Beowulf clusters, these data can be processed much faster than is possible with typical workstation computers in use today. Processing time can be critical for applications such as hazard response or planning.

The increased processing facilities of distributed processing also allow larger digital datasets to be more easily processed than was possible before. Not only can larger areas be examined, but finer resolution data can also be processed more easily. This type of processing can benefit activities such as urban dynamics modeling and prediction activities. These activities can benefit because the increase in resolution and detail can produce better predictions of urbanization.

Several issues must be examined when developing these types of systems. The first issue is how to distribute processing across a network. Although many techniques are available, the master/slave approach is well suited for data-bound distributed processing applications. As map projection algorithms are very tightly coupled, there is no easy way to distribute the projection algorithms themselves. As a result, a better approach may be to distribute the actual data processing.

The primary issue is distributed processing of geospatial data involves large amounts of data that must be dealt with. Processing these data over a network can lead to severe congestion problems that can actually slow down the runtime of an application. As a result, the systems may not be processing information as efficiently as possible. This is an important issue to research, as most of the computer science work in distributed processing has dealt primarily with processor-bound applications. Solutions to the network congestion problem can lead to systems that can process digital data more quickly than is currently possible with today's clusters.

Care must be taken when designing a cluster to project digital geospatial data. Purchasing a large number of high-end machines can result in a cluster that performs no better than one that was built for a fraction of the cost with half the number of machines. Many techniques will begin to see diminishing speed improvements past a certain number of processing nodes on the cluster. Because of this, organizations must first evaluate what they intend to do with their system before they begin throwing machines at a problem.

Distributed processing is a technique that can offer many advantages to processing digital geospatial data. The runtime improvements that it can bring to projecting data can assist projects from hazards planning to decision support systems. Remote sensing systems have evolved to the point that more information is being produced than a typical desktop system can easily process. This is especially true as the size of typical datasets increases from megabytes to gigabytes of information.

Future Work

More research is needed on efficient methods for the distributed processing of large amounts of data. Finding ways to overcome the problem of massive network collisions will enable large datasets to be more efficiently processed. In the case of map projections, the only way to effectively speed up processing is to increase the efficiency of data passing between nodes on the network.

Research into using parameterized matrix operations for certain coordinate systems conversions should also be investigated more thoroughly. This method may well work for certain coordinate conversions at certain scales. Coordinate system transformations such as UTM to State Plane may benefit from this type of algorithm. A cluster-based system might be better able to handle large numbers of small-scale matrix operations that are necessary to treat certain conversions as an image processing application.

References

Cannan, Edward, 2001, *The Origin of the Law of Diminishing Returns*: McMaster University. <http://socserv2.mcmaster.ca/~econ/ugcm/3ll3/cannan/ cannan003.html>.

Dedkov, Anatholy F. and Eadline, Douglas J., Performance *Considerations for I/O Dominant Applications on Parallel Computers*: Paralogic Corporation. <ftp://www.plogic.com/pub/papers/exs-pap6.ps>.

Matthews, John H., 1992, *Numerical Methods for Mathematics, Science, and Engineering*, 2nd ed.: New Jersey, Prentice-Hall, Inc.

Message Passing Forum, *Message Passing Interface (MPI) Forum Homepage* <http://www.mpi-forum.org/>.

Nichols, Bradford, Buttlar, Dick, and Farrell, Jacqueline Proulx, 1998, *Pthreads Programming*: O'Reilly.

Oak Rdge National Laboratories, 2000, *PVM: Parallel Virtual Machine*. <http://www.epm.ornl.gov/pvm/pvm_home.html>.

Sitaker, Kragen, 2000, *Beowulf Mailing List FAQ, version 2*: Kent State University. <http://dune.mcs.kent.edu/~farrell/equip/beowolf/beowulf-faq.txt>.

USGS Eros Data Center, 1998, *gctpc Announcement*. <http://edc.usgs.gov/programs/sddm/lasdist/contrib/gctpdisclaimer.html>.