Portal Access to Parallel Visualization of Scientific Data on the Grid

Charles Moad Beth Plale

Computer Science Department
Indiana University
Bloomington, IN

Technical Report TR593

Abstract

Visualizing extremely large and time-variant data presents a challenge in overcoming slow disk and network access. The Weather Research and Forecasting (WRF) model, used to model mesoscale weather phenomena, generates large amounts of storm system information that is visualized to better understand storm formation. WRF faces these caveats in disk and network bandwidth limitations. Using the open-source Visualization ToolKit (VTK) and the additional functionality of ParallelVTK, we devised an effective solution to view these data sets at a visually pleasing frame rate. In addition to viewing the data sets in real-time, it is also feasible to render off-screen and create a movie for reusable viewing. These remote jobs can be formed and launched from a remote portal environment using grid tools.

1 Introduction

The primary goal of this project is to effectively visualize the Weather Research and Forecasting (WRF) data model. Effective visualization entails a frame rate that is pleasing to the eye. In order to process the large data sets an open source and parallel visualization framework is needed. The solution does not have to be restrained to commodity hardware, since just the transfer time of the data set on a single IDE drive would yield less than desired results. A pre-existing cluster and storage array is available to distribute and parallelize the task.

The goal of the system is not to visualize the entire contents of the WRF model, but rather to visualize portions of the data. This model could then be extended to visualize the entire dataset if desired.

This paper describes the WRF data model and the components that are to be visualized. Next, the architecture of the hardware platform used for the rendering will be described. Understanding of the hardware will help the discussion of how to best distribute the work load. The software platform that is to be used will be discussed in some detail, as well as the different parallelism models that are available on that platform. Next, the proposed solution is explained in detail. This will include justifications and descriptions of the data and task distribution. Finally, a methodology for converting the visualization to be usable in a remote web/portal atmosphere will be described.

2 WRF Data Model

The Weather Research and Forecasting Data Model is a well known model for meteorological research in mesoscale storm formation. Due to the sheer volume of data generated by the model, the output files present a challenge for rendering and visualization in a real-time manner.

The WRF generated files are encoded in the network Common Data Form (NetCDF)[5] format. This self-describing binary encoding allows for easier transfer and portability of the data. NetCDF libraries are available for many languages, hence a user is not restricted to a specific programming environment.

The WRF files used for this project were generated at NCSA. A WRF execution consists of 150 times steps modeling a collision of two storm systems converging into one. Each time step is a self-contained NetCDF file of approximately 15MB. Thus a time series is roughly 4.2GB of data. The NCSA repository contains 290 time series, but each time series is a self-contained storm system.

A time step contains roughly 70 data variable. Some of these variables contains an array of data for

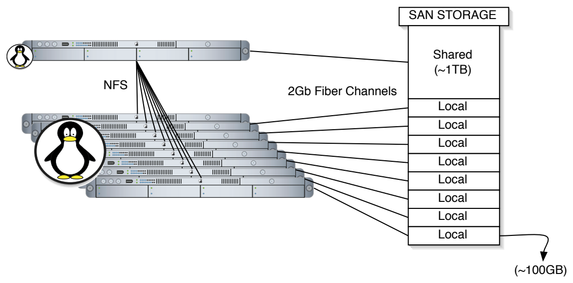


Figure 1: Visual diagram of Thor hardware architecture.

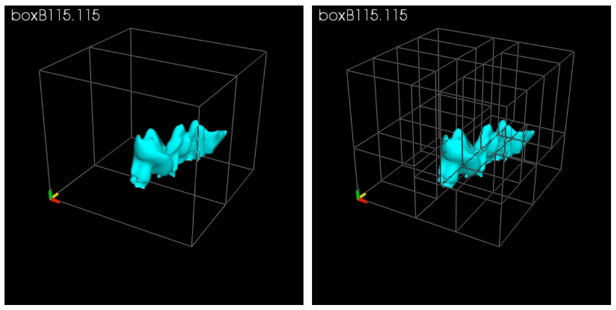


Figure 2: Sample divisions of data parallelism. The left represents 2 processes and the right shows the division of 18 processes.

the sample area, and the dimension of these are 90 by 90 by 59. The location of the data points are in a grid-mapped box, which does not necessarily have even divisions. Additional discussion of the data layout is discussed in section 5. These variables represent information such as temperature, pressure, or precipitation. For the visualization of concern, the values for clouds and rain are extracted and summed for each point. This yields an overall precipitation value.

3 Hardware Architecture

In order to achieve the best possible frame rate and throughput of data it is important to know the hardware architecture of the target system as it controlled several design decisions. In this section we describe the platform that was used for the project.

As part of a NSF Research Infrastructure grant, the Computer Science Department of Indiana University recently purchased a leading edge Linux cluster and storage system[1]. The Thor cluster consists of nine 2.8GHz dual Xeon machines. Each machine has 2GB of RAM and is currently running RedHat 8.0. The machines are organized as one head node, thor, which is primarily used to launch jobs and the remaining 8 machines, thor1-8, as compute and IO nodes. Network communication is currently gigabit ethernet, myrinet, and infiniband.

Each machine has a fibre connect to the Storage Area Network (SAN). Each connect runs to a 2Gbps switch which connects the 2.2TB raid array. For file system protection each machine has its own partition on the array. The head node has a 1TB partition, while each subnode has 100GB partitions. A visual depiction appears in Figure 1. The head node's partition is NFS mounted on all subnodes as a shared partition, and the subnode's partitions are not accessible by other nodes. To utilize the fibre connect of the SAN on all nodes the data sets are distributed to each of the subnode partitions.

[1]More detailed information can be found at `http://www.cs.indiana.edu/Facilities/notices/HagarThor.html`

4 Parallel VTK

The task of visualizing the WRF datasets presents the challenge of overcoming the high disk IO required to yield pleasing frame rates. In order to facilitate this and make efficient use of the Thor cluster and the SAN storage, a scalable and parallel framework was used. The Visualization ToolKit (VTK) and extension, Parallel VTK[1], offer an open source and multi-platform solution. Parallel VTK builds on top of the MPI of choice for parallel computation.

Parallel VTK offers three distinct parallel models. For extremely large, single time-step data, *data parallelism* evenly divides the data among the processes to be rendered. As can be seen in Figure 2, one has no control over the data division. Each time step in the WRF dataset is actually small enough that one process is quite capable of rendering the entire scene easily.

The second parallel option, *pipeline parallelism*, makes data parallelism less useful. It is intended for large, time-varying datasets and is relevant for use cases such as WRF. It allows one process to perform the disk IO, while another process can do the rendering. These processes can be performed concurrently, so process 1 can start reading in time step 2 while process 2 is still rendering time step 1. Communication is performed by input/output port classes that abstract the MPI sends/receives.

The final parallel model is task parallelism. This allows for a step in a VTK data flow to be performed in parallel. As described in the next section, I use this functionality to read in time steps of the data series in parallel. Rendering the data I pull from the WRF files does not require more than one process, since it is overpowered by the disk IO bottleneck incurred from reading in the data.

5 Implementation

A VTK visualization consists of a visualization pipeline (not to be confused with pipeline parallelism). The pipeline used for this visualization can be seen in Figure 3.

Traditionally a pipeline starts with a data source. In this case the data source is a vtkRectilinearGrid. A rectilinear grid is a 3-dimensional grid of data points with a fixed number of values in the x, y, and z directions. It also allows for the divisions in each coordinate direction to be arbitrary.

The input and output ports are VTK classes which allow processes to send and receive data sources. These processes could lie anywhere in a MPI environment, so they are most likely on separate machines. This communication method will allow the visualization to take advantage of pipeline and task parallelism. vtkInputPorts are designed to be "single source", so only parameters to a data set can be altered after the source is pushed across the network the first time. Since each time step is an entirely new data source a new instance of the input port is made for each time step. This forces the output port to push over the entire data source upon each update request.

Next the data source is passed to a contour filter. Contour filters perform isosurfacing of the data. A rectilinear grid is just a 3-dimensional array of values and not a traditional 3-dimensional object that is part of a visualization scene. Given a certain threshold the contour filter will convert the data source to the polygonal representation wanted for viewing.

Finally, VTK pipelines are often closed with a data mapper and an actor. The data mapper converts the object representation to a representation that the graphics card will better understand. Actors are traditional in scene graphs to represent objects and their appearance. All actors are added to the window to create the entire scene. Text, bounding boxes, and objects are all represented by different actors, so Figure 3 does not contain any of these other items that may appear in the scene. A view of the mapping of the visualization pipeline onto the cluster is shown in Figure 4.

Since reading in each time step from disk presents

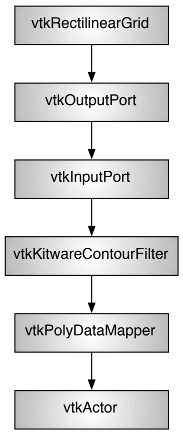


Figure 3: Abstract view of the VTK pipeline for this system.

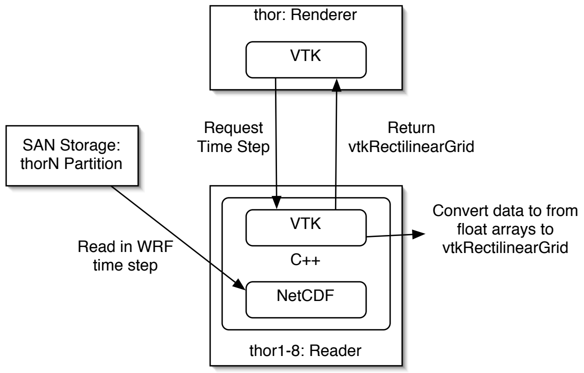

thor: Renderer
VTK
Request Time Step
Return vtkRectilinearGrid
SAN Storage: thorN Partition
Read in WRF time step
VTK
C++
NetCDF
thor1-8: Reader
Convert data to from float arrays to vtkRectilinearGrid

Figure 4: Flow of WRF data to VTK format. Readers/Renderer communication.

the largest time consumption, we identified this task for parallelization. Isosurfacing and rendering each time step is relatively fast and these tasks are performed by one node, which will be referenced as the rendering node. The remaining processes will be referred to as reader nodes.

The renderer lies on the head node of thor. All communication between the renderer and the readers is performed through the vtk input/output ports as discussed previously. The renderer will actively request data sets from the readers in a round-robin manner, modulo the number of reader nodes, hence requesting time step 1 from thor1, time step 2 from thor2, and so on. Time step 9 will be requested from thor1.

In order to make use of the fast fibre disk connect on each of the subnodes of thor we distribute the data set of the time series being rendered amongst the 100GB partitions such that each node has sole access. Fortunately, since the renderer is requesting time steps from the sub nodes in modulo 8 the entire time series need not be loaded onto each node's SAN partition, thus eliminating data redundency.

The MPI environment must be prepared to facilitate the needs as described thus far. For pipeline parallelism each process type is defined by a method and set via vtkMPIController.SetMultipleMethod call. Therefore, there is one method that defines the renderer process and one method that defines the reading process. As coded, the last MPI process will be the renderer and processes 0 through N-2 are readers. LAM MPI was used for the project, and the mpirun parameters best describe the environment: *mpirun c2,4,6,8,10,12,14,16,3,5,7,9,11,13,15,17,0 ...*. The LAM environment is launched stating that thor and thor1-8 each have two processors. When assigning the destination of processes to a machine, one can specify the machine, with an 'n', or cpu, as seen with a 'c'. It is important to note that the process will not truly be bound to a specific processor, but ideally the operating system will delegate the processes to different processors on each machine since there will be two reading processes on each subnode. As seen in the mpirun example the final process, c0, will lie on thor and be the renderer. Processes bound to c2 and c3 will be bound to thor1, c4 and c5 will be bound to thor2, and so on. By assigning the order these processes are bound to machines it assures that we can distribute the data set without having any redundency. It also assures that the round-robin scheduling will hold. This evenly distributes the workload and presents the most concurrency possible in the SAN storage IO.

All processes have access to a shared vtkMPIController object that allow the process to identify its index. These indices are used to set tags in the input and output port objects. The techniques allow the renderer to actively prepare an input port to request the next sequential time step from the corresponding process that lies on the machine with the time step on its SAN partition.

6 Web/Portal Access

Many users do not have access to the caliber of hardware used in this study. Further, running the same visualization job repeatedly may not be practical on machines under user contention. A bottleneck we found during remote displaying the visualization is a 100Mb Ethernet connection, typically used between a cluster and a personal laptop or workstation. Remote visualization saturates the network and slows down the vis as more appears in the vis window. Most users do not have access to a gigabit network to overcome this bottleneck. A more practical solution is to submit a job to the rendering cluster and save the output for reuse.

The first problem associated with rendering remotely is that a user will typically not have access to a X server to remotely display the visualization. VTK addresses this by offering the ability to render off-screen. Off-screen rendering allows the visualization to be rendered on the graphics card as normal, but the result does not have to be displayed on the rendering machine's screen. Instead, the output can be sent to a stream or in our case, a file. Each time step is saved to an individual image file. VTK offers several file formats such as TIFF, JPEG, and GIF.

The end user does not wish to see a collection of images, so the images must be adjoined into some sort of movie. Many libraries exist that can convert a series of images into a movie format such as MPEG, Quicktime, or animated GIF. We chose animated GIF since it can be viewed easily on all platforms and does not require a browser plugin. ImageMagick is a freely available collection of utilities for image manipulation. One such tool, *convert*, allows one to adjoin a collection of images from an recognizable image format to an animated GIF. It also allows for compression over the result leading to a result file of only a few megabytes, as opposed to tens of megabytes which is typical for an uncompressed animated GIF or MPEG movie.

The final problem exists in transferring the result to the user. GridFTP is a grid enabled file transfer protocol and serves to transfer a file directly to the web browser for immediate viewing or to another machine for reuse.

The steps described above we implemented in a workflow. Globus Toolkit 3 (GT3) offers several tools to manage work flow, one such tool is GridAnt[3]. In GridAnt a work flow is described in a xml/ant format. In the portal context a generic factory service can be used to create a service that performs the tasks described by the workflow. Status tracking is another feature that one gains from GridAnt, and the user can move around or even leave the portal as the workflow is being executed. Once the workflow has completed execution GridFTP is used to store the result in the user's context on the portal. The result can then be viewed, manipulated, or deleted from the user's context as desired. A layout of this process can be seen in Figure 5.

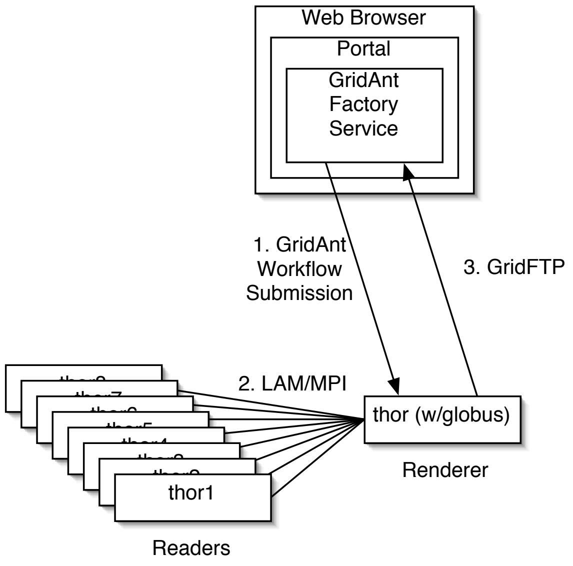
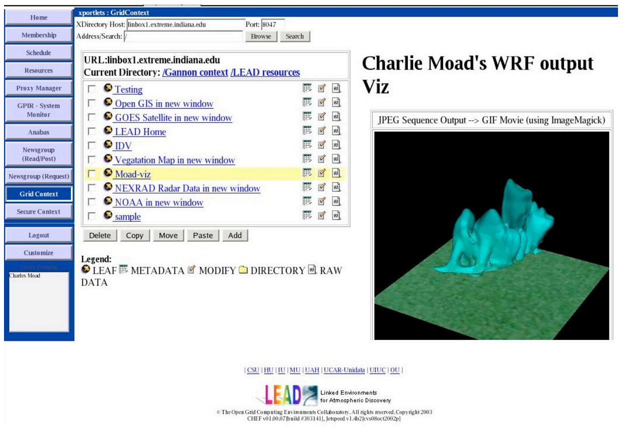


Figure 5: 1. GridAnt submits the work flow via GRAM job launch submissions. 2. The workflow consists of launching and running the LAM-MPI/ParallelVTK rendering job. The animated GIF is also generated using ImageMagick. 3. Finally a launch of GridFTP is used to bring the final movie back to the browser or the user's context in the portal.



xportlets : GridContext

XDirectory Host: linbox1.extreme.indiana.edu Port: 8047

Address/Search: /

URL:linbox1.extreme.indiana.edu

Current Directory: /Gannon context /LEAD resources

- Testing
- Open GIS in new window
- GOES Satellite in new window
- LEAD Home
- IDV
- Vegatation Map in new window
- Moad-viz
- NEXRAD Radar Data in new window
- NOAA in new window
- sample

Legend:
LEAF METADATA MODIFY DIRECTORY RAW DATA

Charles Moad

Charlie Moad's WRF output Viz

JPEG Sequence Output --> GIF Movie (using ImageMagick)

| CSU | HU | IU | MU | UAH | UCAR-Unidata | UIUC | OU |

LEAD Linked Environments for Atmospheric Discovery

© The Open Grid Computing Environments Collaboratory. All rights reserved. Copyright 2003
CHEF v01.00.07[build #303141], Jetspeed v1.4b2[cvs08oct2002p]

Figure 6: A sample Grid context for a user in a portal environment.

In the Grid portal each user has a Grid context. This is a small online locker that can be used to save anything from files to services. Each of the items in a user's context is represented by a moc file. This file could contain a GridAnt script to launch a job when clicked, or it could contain the html wrapping needed to display an applet. In Figure 6 we show an instance of the visualization running in a user's context. One link can be created to launch the GridAnt workflow to remotely render the parallel visualization on a cluster, copy the resulting movie from the cluster to a user's Grid context, and display the result in the browser. This is all achievable without leaving the portal.

7 Related Work

The problem of visualizing extremely large data sets remains a computational challenge. Groundwork has been laid out with ParallelVTK by Kitware and various national labs[4][1]. They provide the class extensions to VTK to allow for data, task, and pipeline parallelism to easily be accomplished. Using these classes we were able perform parallel computation using a combination of pipeline and task parallelism. Pipeline parallelism provides an ideal solution for time-varying datasets, and in our case task parallelism provides an advantage when using several reader processes pulling from the same disk array. The combination would prove even more useful when pulling time steps from separate disks or if the reader process was more cpu bound.

Similar work has been done in the field of scientific portal visualization via the grid[6]. Most noticeably is the collection of services provided with VisPortal[2]. The services provided by VisPortal include file management to distribute and maintain large data-sets, MPEG movie generation, AMR Volume Rendering, AMR WebSheet, and Visapult. All these services aim to reduce the number of steps that a scientists would have to enter to perform a distributed grid-enabled visualization. This lends to an overall goal of abstracting the entire process into a single launch.

8 Conclusion

We have presented a solution for visualizing extremely large data sets in a real-time high performance environment, that can also be easily modified for use on commodity hardware. Likewise, the result can be viewed in real-time while being rendered or can be saved for reuse at a later time.

With the growing need of offering scientists access to remote, high performance applications in a easy to use context, the Grid and the portal are uniting to offer a unified and simple solution. Viewing real-time results is not always necessary or practical, so by offering the ability to launch a job and come back later for the results scientists can be more efficient in their work. The portal also offers web administrators the ability to package a collection of tools simply into one context.

References

[1] J. Ahrens, C. Law, W. Schroeder, K. Martin, and M. Papka. A parallel approach for efficiently visualizing extremely large, time-varying datasets, technical report laur-001630, los alamos national laboratory, 2000., 2000.

[2] E. Wes Bethel C. Siegerist, J. Shalf. Visportal: Increasing scientific productivity by simplifying access to and use of remote computational resources, technical report lbnl-pub-893, lawrence berkeley national laboratory, 2003. 2003.

[3] K. Amin S. Hampton G. von Laszewski, B. Alunkal and S. Nijsure. Gridant-client-side workflow management with ant, whitepaper, july, 2002. 2002.

[4] K. Martin B. Geveci C. Law J. Ahrens, K. Brislawn and M. Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications*, 21(4):34 – 41, jul 2000.

[5] R. Rew, G. Davis, and S. Emmerson. Netcdf user's guide: An interface for data access, version 2.3, 1993.

[6] Kwan-Liu Ma Bernd Hamann Kenneth I. Joy T.J. Jankun-Kelly, Oliver Kreylos. Deploying web-based visual exploration tools on the grid. *IEEE Computer Graphics and Applications*, 23(2):40 – 50, mar 2003.