IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 12, NO. 5, SEPTEMBER/OCTOBER 2006
NTR; NAS-07-002; January 2007

Concurrent Visualization in a Production Supercomputing Environment

David Ellsworth, Bryan Green, Chris Henze, Patrick Moran, and Timothy Sandstrom

Abstract— We describe a concurrent visualization pipeline designed for operation in a production supercomputing environment. The facility was initially developed on the NASA Ames "Columbia" supercomputer for a massively parallel forecast model (GEOS4). During the 2005 Atlantic hurricane season, GEOS4 was run 4 times a day under tight time constraints so that its output could be included in an ensemble prediction that was made available to forecasters at the National Hurricane Center. Given this time-critical context, we designed a configurable concurrent pipeline to visualize multiple global fields without significantly affecting the runtime model performance or reliability. We use MPEG compression of the accruing images to facilitate live low-bandwidth distribution of multiple visualization streams to remote sites. We also describe the use of our concurrent visualization framework with a global ocean circulation model, which provides a 864-fold increase in the temporal resolution of practically achievable animations. In both the atmospheric and oceanic circulation models, the application scientists gained new insights into their model dynamics, due to the high temporal resolution animations attainable.

Index Terms—Supercomputing, concurrent visualization, interactive visual computing, time-varying data, high temporal resolution visualization, GEOS4 global climate model, hurricane visualization, ECCO, ocean modeling.

1 Introduction

In one of the original reports delineating the field of scientific visualization, *Visualization in Scientific Computing*, McCormick et al. [19] described a vision for the future where scientists could analyze and interpret data from supercomputing calculations as they were running. They called this capability *interactive visual computing*, also known as *concurrent visualization*. Related ideas have been developed further in the visualization community (see next section), but in general have seen limited use by the computational science community.

There are two primary benefits of concurrent visualization. First, it shows a view of the current state of a calculation, which allows runtime monitoring, steering, or perhaps termination. Second, concurrent visualization allows higher temporal resolution visualization compared to traditional post-processing because I/O and storage space requirements are largely obviated. This higher temporal resolution may show features in a simulation that would otherwise not be visible.

Given these benefits, we implemented a concurrent visualization pipeline within the "Columbia" supercomputer environment at NASA Ames Research Center. Our driving application was the MAP'05 project [18] led by a team at NASA Goddard Space Flight Center. The project used Columbia to run 5-day weather forecasts every six hours during the 2005 hurricane season (June to November). The hurricane tracks—not our visualizations—from each simulation run were sent to Florida State University and combined with other model forecasts as part of a "superensemble" [9], which uses machine learning techniques to create a single forecast. The single forecast was made available to the National Hurricane Center.

The MAP'05 project's high profile and tight schedule imposed several unusually strict requirements on the development of our visualization pipeline. Because of the hard deadlines for submitting forecasts to FSU, our visualization system could not significantly impede the simulation, and most importantly could not cause it to fail. In addition, any modifications to the simulation code had to be minimized in order to facilitate validation. Partial failures of the visualization pipeline had to be handled gracefully – we wanted to avoid killing the entire process, if possible, and, since runs were often unattended, partial or catastrophic errors in one run should not affect the next run. A final requirement stemmed from the fact that the MAP'05 researchers are across the country from the supercomputer, and are reachable only over a shared, medium-speed network connection.

These requirements led us to a design which is different than earlier developed systems (many of which are described in the next section). Our system limits modifications to the simulation code by having it only copy data to a shared memory segment. The next stage in the visualization pipeline receives data via the shared memory segment, and it runs on separate processors in parallel with the simulation. This decoupling effectively prevents any visualization failures from adversely affecting the simulation. Data are sent from the shared memory segment (via an intermediate system) to multiple rendering nodes, each of which produces a time-varying visualization. Frames from the visualizations are compressed using MPEG encoding, and the resulting MPEG streams can be sent to the remote sites, where the currently completed time steps of the forecast are continuously shown in an animation loop. Using MPEG compression greatly decreased the network requirements; overall we saw a 66 to 1 compression ratio. The visualizations are both rendered and displayed on small visualization clusters. For display we typically used a 3×3 tiled array, and showed different simulation variable/view combinations on each of the nine displays. Figure 4 shows a typical configuration.

The rest of the paper is structured as follows. First, we describe some related work, and then present an overview of the MAP'05 simulation project. Next, we describe our concurrent visualization architecture, first giving an overview, and then describing the data extraction, visualization, and MPEG production portions of the pipeline. The following sections describe the system's effectiveness for the MAP'05 project, and briefly discuss the use of our system with a second application. We finish with conclusions and some areas of future work.

2 Related Work

Concurrent visualization has been explored by many different groups. However, we did not find an existing system that met our application's

- *David Ellsworth is with AMTI at NASA Ames Research Center, E-mail: ellswort@nas.nasa.gov.*
- *Bryan Green is with AMTI at NASA Ames Research Center, E-mail: bgreen@nas.nasa.gov.*
- *Chris Henze is with NASA Ames Research Center, E-mail: chenze@nas.nasa.gov.*
- *Patrick Moran is with NASA Ames Research Center, E-mail: patrick.j.moran@nasa.gov.*
- *Timothy Sandstrom is with AMTI at NASA Ames Research Center, E-mail: sandstro@nas.nasa.gov.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

NTR; NAS-07-002; January 2007

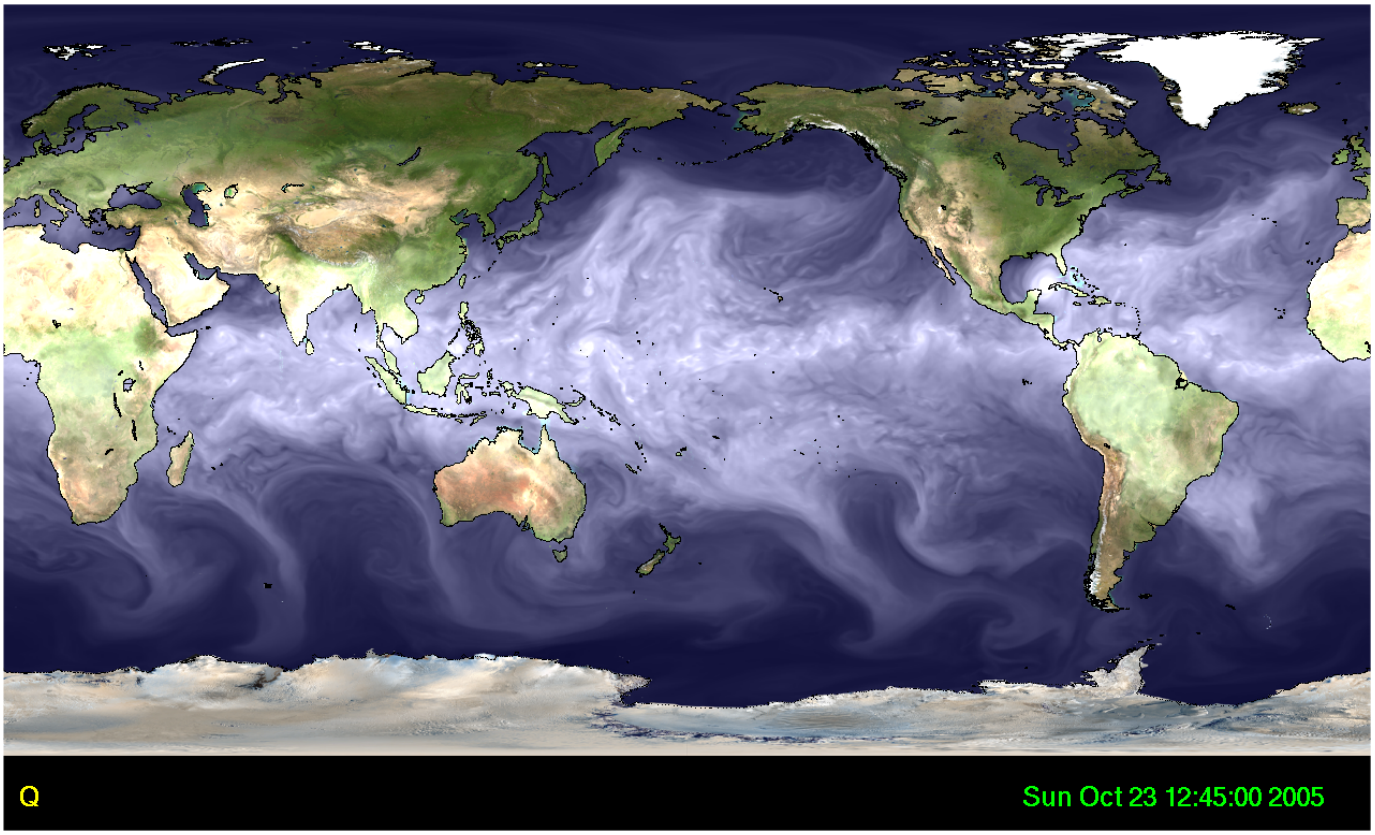


Fig. 1. A frame from a concurrent visualization animation of GEOS4, showing Hurricane Wilma approaching Florida. The frame shows the specific humidity (Q, mass(H_20)/mass(air)), summed over all elevations and mapped onto luminosity.

requirements. For example, the pV3 system [12] computes requested visualizations using the simulation processors, which could impact the simulation's run time and reliability. Other earlier efforts in this category are SCIRun [14], the commercial package RVSLIB [7], the Earth Simulator's geoFEM [20], and VisIt [5].

The requirement of sending multiple visualizations across a moderate-speed network as they were generated dictated our choice of compressed MPEG streams. Earlier systems sent the original data [16], data extracts [12], geometry [20], or images [7, 20] to the remote system. Distributed visualization systems built using AVS [4] or CM/AVS [22] send inter-module communication over the network, which we expect would require too much bandwidth for our purposes.

The requirement of minimal modifications to the simulation code eliminated approaches that tightly couple the simulation to the visualization. Problem solving environments, such as SCIRun [14] and Cactus [1], are examples of this approach. Other visualization systems built as libraries or frameworks also require substantial modifications to the application. Two such frameworks are the CUMULVS parallel processing framework [10], and the VisAD [13] component visualization framework. The DICE [6] system's use of Network Distributed Global Memory for transport similarly does not meet the minimal modification requirement.

The use of compression for the remote transmission of animations is widespread. Two packages that use compression for remote visualization are RVSLIB [7] and ParaView [3].

3 Hardware Resources

The MAP'05 simulations were run on the Columbia supercomputer at NASA Ames. Columbia contains 20 SGI Altix nodes, each of which has 512 Itanium 2 processors and 1 TB of memory. Each processor in a node has cache-coherent access to all the node's memory, and the nodes are connected by Ethernet and InfiniBand. The GEOS4 forecast runs were run on one 512-processor node.

Our rendering cluster has 50 dual-processor 1.67 GHz Athlon systems, each with a graphics card and a 100Mbit connection to a private net. A separate 16-processor Altix system, called `chunnel`, serves as an intermediary between the Columbia compute nodes (via one 4x InfiniBand link) and the graphics cluster switch (via 8 Gigabit Ethernet connections).

The current bottleneck in our system is the 100Mbit connections to the rendering cluster nodes. This precludes transferring 3D fields for the GEOS4 application.

4 GEOS4 Application Overview

The MAP'05 project employed the fourth generation of the Goddard Earth Observing System (GEOS) suite of models, developed at NASA Goddard. GEOS4 is an implementation of a finite-volume general circulation model (fvGCM), which uses a Lin-Rood semi-Lagrangian dynamical core [17] in conjunction with the Community Climate Model (CCM3) [15] for physical parameterizations and land surface model [2].

The production form of GEOS4 is a nominal 1/4 degree global model, using a standard latitude-longitude staggered grid of dimensions $1000\times721\times32$ (23 million cells). Three-dimensional scalar and vector-component quantities defined on the grid are represented in double precision and thus require $1000\times721\times32\times8 = 184.6$ Mbytes of storage each. Horizontal 2D scalar fields occupy 5.8 Mbytes each, and represent either surface quantities or values representing an average over the entire vertical column. Parallelism is implemented using a one-dimensional latitudinal decomposition, with 3 ghost cell layers to the north and south of each computational subdomain. On the Altix architecture, a hybrid MPI-OpenMP parallelism strategy was employed. This uses both the distributed-memory MPI and shared-memory OpenMP programming models. Production runs used 60 MPI processes with 4 OpenMP threads each (240 processors total). The runs simulated 5 full days of atmospheric dynamics, using integration

time steps that represented 450 seconds of physical time (960 time steps total). Some test runs instead used 480 processors, 480 time steps, or both.

Our standard visualization configuration copied about 900 GB of simulation data per run. Of this, 177 GB, or one 3D field, was vertically integrated into a 2D field, and 709 GB, four 3D fields, had a horizontal 2D slice removed from each 3D field. In addition, two 2D fields were used, consisting of 11 GB. After conversion to 32-bit floating point, the resulting seven 2D fields (totaling 19.4 GB) were sent from the Columbia system to the rest of the pipeline.

Wall-clock times for the production runs took about 55 minutes. We saw a minimum wall clock time per integration time step of 2.5 seconds. This is less than the nominal time per time step of $55 \times 60/960 = 3.4$ seconds because the simulation spends a significant part of the total run time doing I/O for initialization and diagnostic output. Thus our system must run at the 2.5 second rate if we are to visualize every integration time step without impeding the simulation. Adding buffering to the system would not substantially relax the frame rate requirement unless many frames of buffering were added, requiring substantial memory. This is true because the time steps where the pipeline could catch up, the time steps that take much longer than 2.5 seconds, are widely spaced. However, additional buffering would reduce the number of dropped frames due to temporary pipeline hesitations.

Forecast runs had to be completed in roughly a two hour window before the superensemble submission deadline. In addition to the 55 minute forecast calculations, roughly 30 minutes of post-processing was necessary before results submission. This tight schedule drove our reliability goals.

5 Concurrent Visualization System Overview

Figure 2 shows an overview of the complete concurrent visualization pipeline. This pipeline is divided into three sections: data extraction, visualization, and MPEG production.

The pipeline starts with data extraction. The GEOS4 simulation runs on one of the Columbia nodes. On that same node but running on a separate processor is the `ibcolumbia` process. The simulation processes copy data into a shared memory segment. Then, `ibcolumbia` does any remaining data formatting and copies the data over InfiniBand to another Columbia node called `chunnel`. We use this system because it is the gateway to the rendering cluster's private network.

The visualization portion of the pipeline starts on the `chunnel` system. A process called `ibchunnel` receives data from the InfiniBand network, and writes it to a shared memory segment. Another process called `gserv` copies the data out of shared memory and distributes it to nodes in the rendering cluster. Those nodes render the data, and write the resulting images to local disk.

The pipeline continues with MPEG production. Each rendering cluster node has an MPEG encoder process that encodes frames as soon as they are written, and writes the resulting MPEG stream to a file server. The MPEG files are sent, as they are being created, to the display cluster master node by processes running on the file server. The final step runs on the display cluster. That cluster's master node sends a looped version of each growing MPEG stream to a node on the cluster for display, doing it in a way that synchronizes the streams.

Our approach takes advantage of the available extra processors and memory of the Columbia system used, plus its shared memory architecture. Our system could be adapted for use in distributed-memory systems by extracting data from each node and sending it to one or more separate processors for visualization. However, this would still use the CPU, memory, and interconnect of the running simulation, possibly affecting its performance noticeably.

The following sections describe each section of the pipeline in more detail.

6 Data Extraction

Our visualization pipeline starts with the simulation, or, more specifically, our modifications to it. Our simulation modifications use a shared memory segment to transfer data to a separate process, `ibcolumbia`, which decouples the simulation and visualization code for improved reliability. Also, using shared memory on the Altix allows for very fast data transfer from the simulation, minimizing impact to the simulation run time.

We modified the simulation start-up script and instrumented the simulation code itself. The script modifications activate `ibcolumbia` on separate processors and pass configuration data to it. The simulation code instrumentation causes each MPI thread to register its data structures with the visualization pipeline and to send its data into the pipeline at the completion of each integration time step.

6.1 Start-up Script Modifications

The GEOS4 start-up script requests resources from the batch scheduler, arranges input and output files and directories, and specifies a number of runtime model parameters. Our script modifications extract many of these parameters into a file so they can be passed to `ibcolumbia`. The parameters include the start and stop times of the simulation, the integration time step size, and the fields selected for visualization, so we can properly annotate the visualization frames. They also include the computational domain dimensions, the number of MPI processes and OpenMP threads, and the total number of model time steps, since these data are necessary to `ibcolumbia` to properly deal with the simulation's domain decomposition. Other script modifications increase the number of CPUs requested from the scheduler to accommodate `ibcolumbia`, then invoke and assign `ibcolumbia` to the additional processors so that it does not interfere with any of the simulation processes. Since `ibcolumbia` is invoked prior to launching the simulation code, it creates the shared memory segment for receiving model output ahead of the simulation.

6.2 `ibcolumbia`

`ibcolumbia` serves as the interface between the instrumented simulation code and the rest of the visualization pipeline. Once started, it reads the run-specific metadata generated by the start-up script, and sends some of these data further down the pipeline. It then allocates the shared memory that is mapped into the simulation MPI processes. When all MPI processes have copied their field data from a given time step into the shared memory buffer, `ibcolumbia` invokes a function to process the data. This processing may include conversion from double to single precision, taking 2D slices out of 3D fields, vertical integration of 3D fields onto 2D, interpolation from a staggered to unstaggered grid, and so forth. The output from this processing step is written into a pre-allocated RDMA buffer for fast transfer across the InfiniBand network to `ibchunnel`. If `ibchunnel` has signaled that it is ready to receive another time step, the transfer is made; otherwise `ibcolumbia` drops the time step and sends notice of this event down the pipeline. This mechanism ensures that only time-consistent frames are generated.

6.3 Simulation Code Instrumentation

We modified the simulation code by adding three function calls, which are implemented in a module linked with the main executable. Two of the functions are called once each during initialization, and the third is called at the end of each integration time step. All three functions are called from each MPI process. The first initialization function saves pointers to its MPI process's region of the model fields that are available for concurrent visualization, along with offsets of the MPI subdomain into the global computational domain and ghost cell dimensions. These metadata allow the field data to be reassembled from the per-MPI-thread domains into a single global domain. The second initialization function is called at the end of the simulation initialization; it maps the shared memory arena, created by `ibcolumbia`, into the address space of each MPI process. After every integration time step, each MPI process calls the third function, which copies its field data into the appropriate locations in the shared memory buffer, undoing the domain decomposition and stripping ghost cells by using metadata collected in the first initialization function.

We designed this instrumentation code to insulate it from the remainder of the visualization pipeline. If either of the two initialization

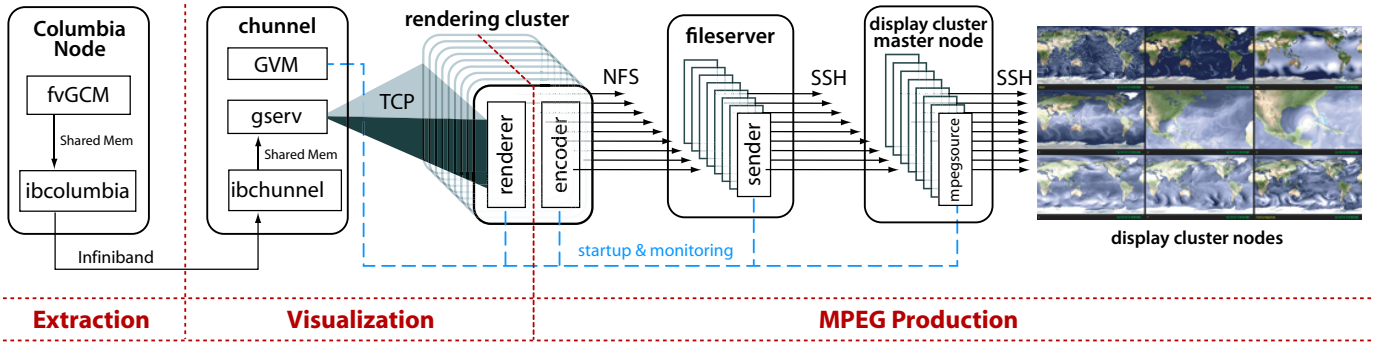


Fig. 2. Our concurrent visualization pipeline. Rounded rectangles indicate systems, and rectangles indicate processes.

functions fails in any MPI process, the copy function becomes a no-op, leaving the simulation unimpeded. Assuming proper initialization, the data copy function ensures that all MPI processes have copied their data into the shared memory arena for processing by `ibcolumbia`. If `ibcolumbia` has not finished accessing the shared memory contents by the time the first MPI process is ready to output its next time step, copying of that next time step is skipped by *all* MPI processes, and notice of this event is sent down the pipeline so that the image streams are properly marked and counted. This mechanism ensures that the visualization pipeline processes only fully intact frames at the maximum rate possible. Alternatively, the user can specify a stride so that time steps are skipped deterministically. This may be useful if the highest possible temporal resolution is not desired, or if a particular run configuration results in the visualization pipeline not being able to keep up with the simulation. With the production GEOS4 runs on the Altix, we were consistently able to capture every model time step, and dropped frames only occasionally due to transient glitches somewhere in the visualization pipeline.

7 Visualization

The visualization step of the pipeline breaks the single data stream from the data extraction step into multiple streams for visualization. Each branch eventually results in an independent MPEG stream, produced from a selection of data fields and the visualization techniques applied to them. The multiple processes in each branch are centrally managed from the `chunnel` system.

To be adequately robust and fault tolerant, the remainder of the visualization pipeline has the following characteristics. Failures that occur in one branch of the pipeline cause truncation of only that branch, and do not affect the other branches. Other failures may cause the entire visualization system to terminate, and then automatically restart itself for the next run. In the worst case, where a failure occurs and attempts at restarting also fail, the entire pipeline is shut down indefinitely and the event logged. Extensive logging occurs throughout the process, so errors can be quickly located and categorized. Missed time steps are recorded in the logs, as well as being visually documented in the final product.

There are two distinct functional roles in the visualization part of the pipeline: process management, which includes configuration and failure handling; and data management, which includes time step accounting and dispatching. The set of managed processes and the flow of data are largely determined by configuration files.

7.1 Pipeline Process Management

At the top of the visualization process hierarchy is the `gserv` daemon. `Gserv` acts as a central data server, routing incoming model data to a set of visualization clients. `Gserv` also acts as a process manager for two other processes: `ibchunnel`, which handles InfiniBand communication; and the Gserv Vis Manager (GVM), which handles visualization configuration and management.

On startup, `gserv` forks a copy of itself. The parent `gserv` process only monitors the child, restarting it if it fails, and exiting if failures occur repeatedly. The child `gserv` process monitors `ibchunnel` and GVM; if either fails it stops the pipeline and reinitializes itself.

GVM is a Perl script that reads the configuration file, and then creates and manages one branch of the pipeline per visualization, consisting of rendering, encoding, sending, and viewing components, all of which are created on remote hosts via `rsh`/`ssh` connections. In the event of a localized error, GVM terminates only the portion of failing branch downstream of the error, allowing any intermediate results to be archived. In the case of complete failure of the visualization phase, GVM shuts down the pipeline and exits.

GVM interprets configuration files in order to construct the multiple branches of the visualization pipeline. When started, GVM receives the list of fields that are being exported by the current model run. It uses a table, constructed from a configuration file, to associate available sets of fields with a particular rendering setup. For each match in the table a new branch in the visualization pipeline is created: a rendering program is launched on the next available node in the graphics cluster, the parameters specified in the table entry are passed in, and the supporting processes for movie generation, distribution, and display are created.

7.2 Visualization Clients

When GVM starts a visualization client on one of the rendering nodes, the client is given one or more field names on the command line. These names are sufficient for the visualization program to independently register with `gserv` to receive time step data for those fields as asynchronous events. Then, as frames of data arrive in shared memory from InfiniBand, `gserv` sends the appropriate field data to each visualization client using TCP, based on the client registrations.

Our current visualization client is very simple: it can display a 2D array of either scalar values or the magnitude of vector values using a luminance map. Parameters specify min-max values, viewpoint, and so on. While this simple technique has been sufficient for visualizing global climate models, we can easily incorporate other visualization techniques.

7.3 Pipeline Data Management

`Gserv` transfers data to the visualization clients using a distributed object framework called `growler` [11], which is capable of managing dynamic connections from multiple clients. The low level details of client connections, requests, and data events are handled by the `growler` framework. An Interface Definition Language is used to describe the communication semantics between `gserv` and the visualization clients.

Time step data are never purposely dropped downstream from `gserv`. If data arrive faster than can be consumed, `gserv` blocks until the renderers complete, and as a result, `ibcolumbia` will not receive the acknowledgment it needs to propagate the next time step.

The visualization portion of the pipeline uses multiple threads for improved performance in several places. In `gserv`, field data are

copied from shared memory into buffers for transmission to visualization clients using one thread. Multiple other `gserv` threads handle moving the buffers into the TCP stack. In the visualization client, one thread receives incoming field data, while a second thread renders the data and writes them to disk.

Once the image has been written to disk, subsequent processing of the image data is completely decoupled from the simulation time step loop, and cannot contribute to lost frames.

8 MPEG Production

The last stage of the pipeline, MPEG production, begins with the frames written to local disk by the renderers. GVM manages the processes of the MPEG production pipeline, which has three main components: encoding, transmission, and viewing.

8.1 MPEG Encoding

An encoding process runs on each of the rendering nodes. On startup, an introductory MPEG is created, which is used to allow the MPEG viewers to start up immediately, before the simulation has generated any time steps. The process then waits for the first real visualization frame to appear in the local file system. Once frames begin appearing, a modified version of the Stanford PVRG MPEG-1 encoder is launched to process the frames. We modified the encoder so it will wait for frames to appear on disk. The resulting MPEGs are written over NFS and collected on a single shared file server.

For backup purposes, another process uses `gzip` to losslessly compress the frames and write them to the file server for long term storage. Individual frames on the local disk are deleted once they have been MPEG encoded and archived.

An example MPEG animation is included on the conference DVD to illustrate the MPEG output quality.

8.2 MPEG Transmission

Each MPEG can be sent to remote viewing systems while it is being created. A simple program waits for the MPEG file to be created on the file server. As the MPEG file grows, this program outputs the new MPEG data through an `ssh` pipe to the remote system, where it is written to disk. The transmission pipeline exits when it finds a MPEG "Sequence End" marker, which marks the end of the animation.

The bandwidths required to send the MPEG streams are quite low. For the nine standard views, the average bandwidth required per MPEG stream ranged from 8 to 39 KB/sec; the total bandwidth for the nine streams was 157 KB/sec. These bandwidths correspond to compression ratios of 140:1 to 29:1 compared to the original 24-bit RGB frames. The overall compression ratio was 66 to 1.

8.3 MPEG Viewing

One possible destination for the streaming MPEGs is a *hyperwall*: a two-dimensional grid of displays backed by a cluster [21]. For this project, a 3×3 hyperwall was used, enabling nine simultaneous visualizations of a running simulation on a single wall. A tenth "master" system controls the nine display systems, and has the single Internet connection.

Initially, the introductory MPEG animation for each MPEG stream is displayed on the corresponding display node, and the animation is paused until enough frames with simulation data have arrived to enable looping without high-frequency flicker. Then, the viewers on each display node loop at full speed (30 Hz) over the frames of the MPEG that have arrived. The animations on the nine nodes are synchronized.

All this is accomplished using `mpegsource` (a Perl script), and the `mplayer` multimedia application. Nine copies of `mpegsource` run on the master node, and the `mplayer` processes on each display node are configured to read from standard input. Each invocation of `mpegsource` first sends the introductory movie to the `mplayer` process, which allows it to create an output window. Then, each invocation waits for the MPEG file to grow to 20 data frames before sending the frames to `mplayer`. The `mpegsource` scripts then seek back to the start of the MPEG file, and repeatedly send the MPEG frames to the `mplayer` applications. The MPEG files are parsed to ensure that only complete frames are sent to `mplayer`; incomplete frames at the end of the file are skipped. The `mpegsource` processes also wait on a common barrier (created using sockets to a single barrier process) when they reach end of file, so the nine looping MPEGs run synchronously.

Table 1. Shared Memory Copy Times for 480 Time Steps

MPI x OpenMP	Fields	Run Time	Copy Time
60 x 4	1 3D & 1 2D	3117.52	3.83
120 x 4	1 3D & 2 2D	2239.65	3.02
120 x 4	6 3D & 3 2D	2169.13	12.25
120 x 4	7 3D & 4 2D	2154.75	13.625

9 Evaluation

9.1 Reliability

We were able to meet our primary reliability goal: out of 247 production runs with concurrent visualization, there was not a single GEOS4 failure due to the visualization pipeline. The visualization pipeline itself was not perfectly reliable, however, partly due to system configuration changes made during production. Analysis of 132 completed runs (126,720 time steps) showed that we skipped a total of 87 time steps, or 0.07% of the total. The skipped time steps were unfortunately distributed over a fairly large number (44) of runs, which we are currently investigating. Note that 126,720 frames is 70 minutes of animation.

9.2 Model Runtime Performance

A second important design goal was to impact runtime model performance as little as possible. Our design accomplished this by copying the current simulation time step into shared memory, and then running all downstream portions of the pipeline while the simulation was processing the next time step. Thus the majority of the visualization pipeline ran "off the simulation clock," predominantly on separate processors and systems. Our only direct effect on the model runtime was due to the data copy, which was done synchronously, i.e., we blocked the simulation during the copy. The data copy was done in parallel across all MPI processes, and used shared memory that exploited the tremendous aggregate memory bandwidth of the Altix. We determined that an asynchronous strategy involved unnecessary overhead and complexity, although this might be the preferred choice on another architecture.

Table 1 shows overall timings from several model test runs with aggregate times of all data copies during the run, measured with high resolution hardware timers. (The two visualization initialization function calls are negligible, and are not included.) With 120 MPI processes running on 480 CPUs, the overhead for copying 6 full 3D fields and 3 2D fields, at each of 480 time steps, is approximately 12 seconds over a 2100 second calculation, or a little over 0.5 percent. The total amount of data copied is about 0.5 TB. (These timings are from initial test runs. During the season, the model integration time step was cut in half, so the total number of time steps was 960, and we copied about 900 GB per run.)

Although the destination of the data copies is logically a single shared memory buffer allocated by `ibcolumbia`, the physical location of the memory pages is determined by the default "first touch" placement policy. We can touch each page of the shared memory buffer in an initialization routine in `ibcolumbia` so that physical memory resides on the same processors where we have assigned `ibcolumbia`. Alternatively, we can let each MPI process have "first touch" and thus distribute the shared memory's physical pages across the entire simulation's CPU set. We chose the latter, faster, strategy.

A concern with this choice is whether the additional memory pages on the simulation CPUs, and accessing these pages from `ibcolumbia` during its processing phase, will result in indirect effects on the model runtime; these effects were not captured with the timer calipers described above. We investigated such potential interference effects by looking at the overall run times of the model, before

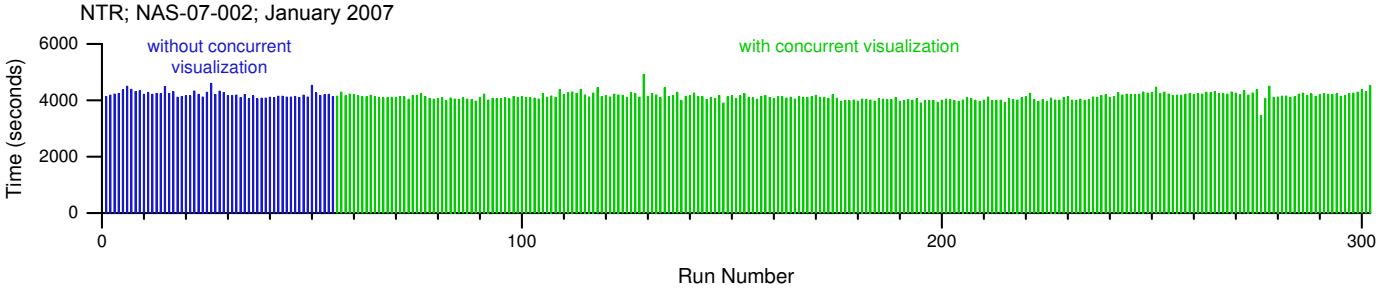


Fig. 3. Overall run times, with and without concurrent visualization.

and after the concurrent visualization was incorporated. Results are shown in Figure 3. Unfortunately, the only long term overall timing data available include some pre- and post-model run activities which involved a lot of disk activity, so the timings are somewhat variable. Nevertheless, it is apparent that there is no systematic increase in runtime when the concurrent visualization was activated part way through the season. In fact, there is a slight *decrease* in the mean run times with visualization. We believe this just shows that any effects we may have produced are slight, and that the data are insufficient to resolve them.

9.3 Visualization Pipeline Performance

We instrumented `ibchunnel` and measured its performance during a test run that used 480 integration time steps that ran at a 2.7 second (wall clock) rate. Of the 2.7 seconds, 0.3 seconds were lost due to the staggered finishing of the simulation processors, 1.8 seconds were spent copying and vertically integrating the field data, and 0.1 seconds were spent sending the data and waiting for acknowledgment, leaving 0.5 seconds of idle time. If the time steps took the 2.5 seconds as measured for production runs, the idle time would be 0.3 seconds. Parallelizing the data copying and integration step would likely be necessary if more data were needed for visualization.

9.4 Application Impact

The real-time remote visualization aspect of our system had limited impact. Due to problems at the remote site, the MPEG streams were only sent to the MAP'05 researchers during a few test runs. Instead, the production runs sent the animations to our local hyperwall so we could monitor the system's progress. In addition, we were able to show real-time production runs at the SC'05 conference in Seattle.

However, post-run user analyses of our high temporal resolution animations had two significant positive impacts on the MAP'05 project. One was the discovery of fast-moving pressure waves that appear to be caused by start-up transients. Earlier post-production visualizations did not capture these transients in sufficient detail for them to be noticed. Our system also helped find the solution to a numeric instability problem that occurred before our system was added to the production runs. Test runs of our system showed how the instability grew from a single cell to the entire domain over just a few time steps. Our high temporal resolution output captured this behavior and immediately suggested the instability's cause.

10 The MITgcm High-Resolution Global Ocean Model

We have also applied our concurrent visualization pipeline to a high-resolution global ocean model, the MITgcm, developed by the ECCO Consortium [8]. Like GEOS4, the MITgcm is a hybrid MPI/OpenMP code which runs on a staggered latitude/longitude grid at various resolutions. Initially, we deployed our concurrent visualization facility on several MITgcm 1/8 degree model runs distributed over 480 processors. In these runs, the global domain is $2880\times2176\times50$, or about 313 million grid points. Thus each 3D scalar field requires about 2.5 GB of storage, and a horizontal slice occupies about 50 MB. Figure 5 shows a frame from the simulation output.

The MITgcm uses a two-dimensional domain decomposition, breaking the domain into full-depth tiles in both zonal and meridional directions, and supplying ghost cells along all four shared tile faces. Our data copying routine that is linked with the simulation code needed to be modified accordingly.

The MITgcm is primarily used for relatively long term climate studies, but needs to resolve the important meso-scale turbulent eddy processes which underly energy and fresh water transport. For these reasons, the model spatial and temporal discretization is very refined, but the runs are long—yielding potentially enormous output data. Integration time steps for the 1/8 degree model are 5 minutes (300 seconds), and for performance and disk capacity reasons only 3-day average field data are output to disk during runs that may simulate months to years of oceanic circulation.

10.1 Month-Long Run

Using our concurrent visualization pipeline, we generated a set of eight animations that captured *every* time step of a single simulation of the month of February 2002. These animations contain 8000 frames each, and represent an 864-fold increase in temporal resolution compared to animations created by post-processing normal run output. Our system was able to handle the higher resolution data because we only copied 2D horizontal slices from the simulation, and because the simulation used more wall clock time per time step (4.4 seconds) than the GEOS4 simulation.

These high temporal resolution visualizations have revealed hitherto unseen model dynamics that have given new insights to the ocean modelers. For example, dramatic diurnal variations in the mixing layer depth shown by our animations are now receiving increased attention as an important factor in the air-sea exchange of CO_2, heat, and momentum.

10.2 Year-Long Run

We have used our system with a single year-long MITgcm simulation run on four 512-processor Columbia nodes, using 1920 processors in total for the simulation. While the simulation has just completed, we do have some initial results. For this run, we generated many more visualizations than previous runs. We extracted 2D slices from 24 fields and showed both a global and North Atlantic view of each field, for a total of 48 visualization streams. We captured every simulation time step, as before. We modified our visualization system so it collects data from each of the four Columbia nodes and forwards the data to `chunnel`. A new program running on `chunnel` collects the per-processor domains from the nodes and reassembles them into the overall domain.

Since this was not a time-critical production run, we modified our system to pause the simulation if the visualization fell behind instead of dropping frames. The extraction and visualization parts of the system nearly always kept up with the simulation even though it was processing much more data per time step at a faster time step rate than the first MITgcm run. Using 1920 instead of 480 processors decreased the minimum wall clock time step time (when I/O is not being done) to an average of 2.8 seconds; the time ranged from 2.7 to 3.2 seconds. Initial measurements show that we only slow the simulation down for about 14% of the frames, and increase the overall run time by 3%. We feel that this is an acceptable slowdown, as do our collaborators. In addition, since this was the first large run using this configuration, we believe that the simulation delay could be reduced or eliminated by further optimizations.

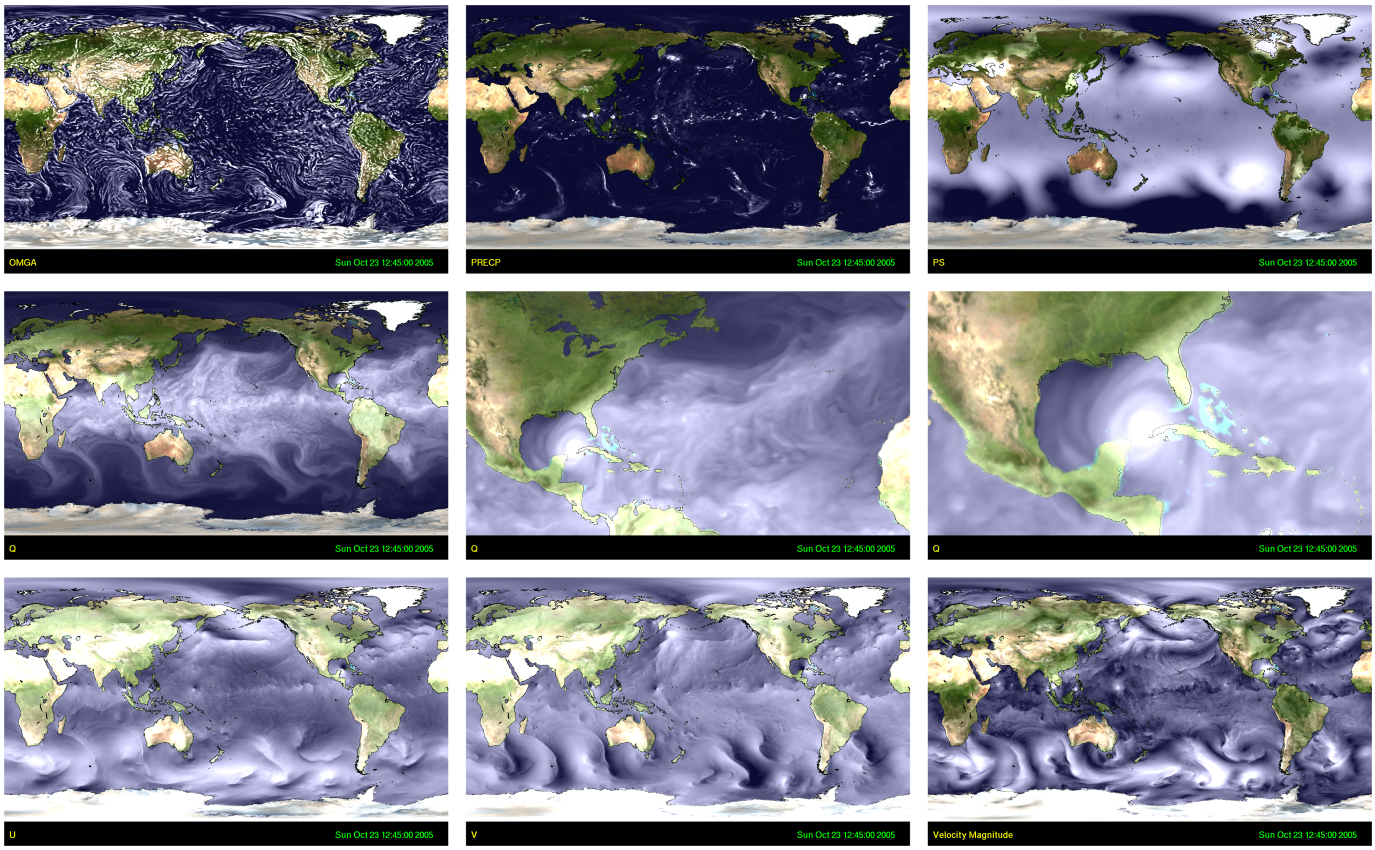


Fig. 4. Typical output display configuration, showing the same simulation and time step as Figure 1, where Hurricane Wilma is approaching Florida. The top row shows (left to right) shows OMGA, the vertical velocity of the pressure grid over time; PRECIP, the total precipitation; and PS, the surface pressure. The middle row shows three views of Q, the specific humidity, integrated vertically. The bottom row shows U, V, and VelocityMagnitude, respectively the east-west component, north-south component, and magnitude of the near-surface wind velocity vector. The OMGA, U, V, and VelocityMagnitude fields are 2D slices from the 3D field that show the values one layer above the bottom layer; PRECIP and PS are 2D fields.

The MPEG part of the system often fell behind the simulation. Several of the visualizations showing the global view had the MPEG encoding and frame compression many frames behind. No data were lost because the MPEG part of the system is decoupled from the earlier part of the pipeline: it reads frames from each node's local disk, which has room for thousands of frames. The slow encoding did affect the MPEG streaming part of the pipeline because the MPEG streams were not synchronized. We believe the slow encoding and compression were due to congestion on the file server, `chunnel`.

In addition to the frames and MPEG animations, we also saved a 2D slice of floating-point data from a 500×500 portion of the grid which shows the North Atlantic for each of the 24 fields visualized. This will allow more complex visualizations, such as LIC, to be computed after the simulation has completed. Our current visualization cluster does not have sufficient CPU or graphics capability to compute these visualizations at the rate required.

Overall, the 110-hour run extracted and visualized a total of 63 TB of data from the simulation, which corresponds to a sustained data rate of 215 MB/second when no I/O is being performed. We saved 2.5 TB of North Atlantic floating-point data and a total of 5 million frames that would require 12 TB of storage if uncompressed.

11 Conclusion and Future Work

We have presented a concurrent visualization system that was specialized for a demanding production application, the GEOS4 simulation, that produced hurricane track forecasts during the 2005 hurricane season. Our system was able to show the simulation's progress to distant researchers using a moderate-speed network link. We have shown that our implementation did not adversely affect the simulation's reliability or run time. The high temporal resolution animations produced by our system led to important new insights with both the GEOS4 and MITgcm applications.

We are currently enhancing our system by adding a variety of visualization and feature detection techniques. We are also working to connect one of our rendering platforms directly to the Columbia compute nodes using InfiniBand. This will avoid our current 100-baseT bottleneck in the pipeline and allow full 3D fields to be transferred to the graphics nodes, which will enable using other techniques such as volume rendering. This will be important when working with other expected new applications where 3D field visualizations are essential.

Acknowledgements

We would like to thank Mike Seablom, Gail McConaughy, and the rest of the MAP'05 team at Goddard Space Flight Center; Chris Hill and Dimitris Menemenlis of the ECCO Consortium; Sue Kim for graphics help; and the anonymous reviewers for their helpful comments. This work was done in part under NASA Contract NNA05AC20T.

References

[1] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, and E. Seidel. Cactus grid computing: Review of current development. In R. Sakellariou, J. Keane, J. Gurd, and L. Freeman, editors, *Lecture Notes In computer Science; Vol. 2150, Euro-Par 2001: Parallel Processing, Proceedings of 7th International Euro-Par conference Manchester*, Aug. 2001.

[2] G. B. Bonan. The NCAR land surface model (LSM version 1.0) coupled to the NCAR Community Climate Model. Technical Report NCAR/TN–

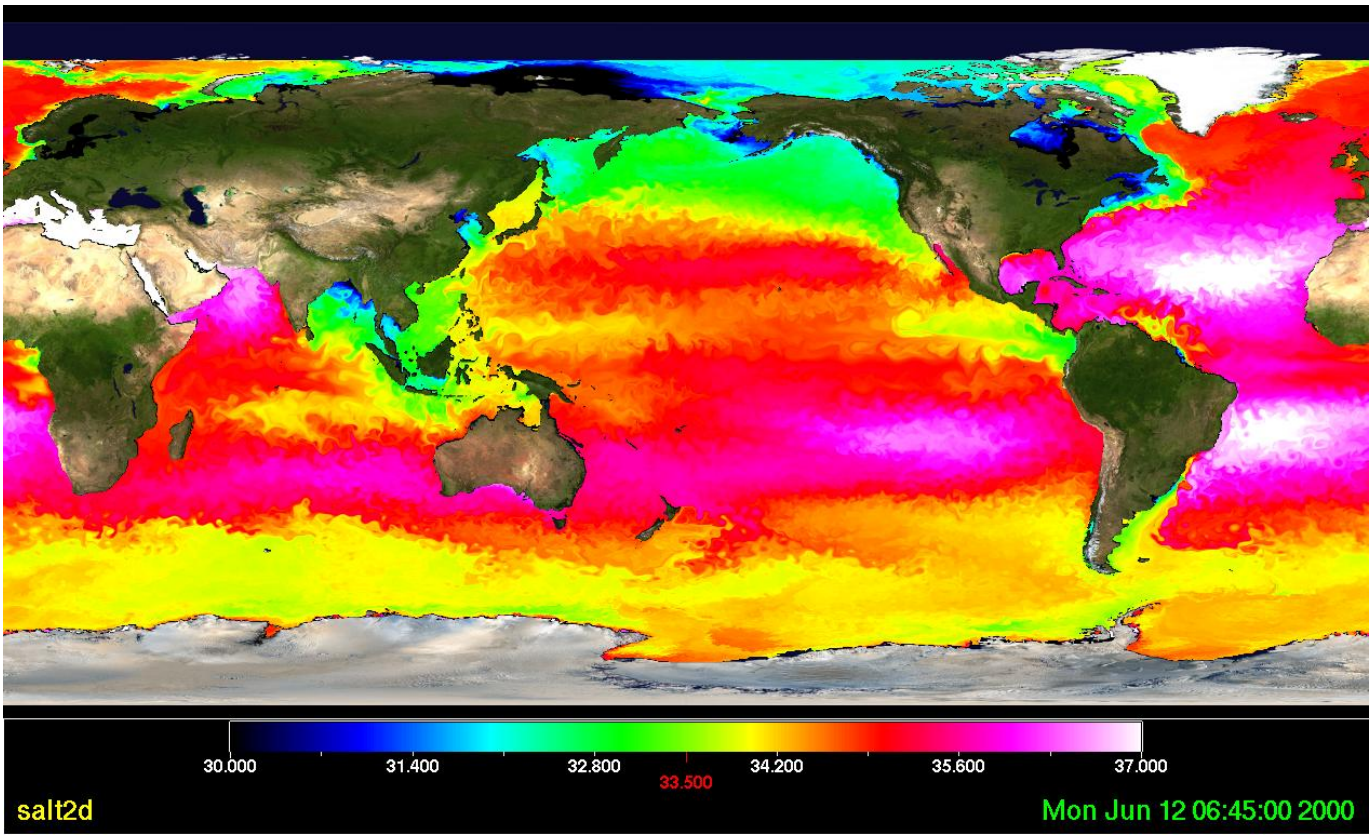


Fig. 5. Frame from the year-long MITgcm simulation showing salinity (parts per thousand of salt by weight) at a depth of 15 meters.

429+STR, National Center for Atmospheric Research, Boulder, Colorado, 1996.

[3] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote large data visualization in the paraview framework. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 163–170, 2006.

[4] G. Cheng, Y. Lu, G. Fox, K. Mills, and T. Haupt. An interactive remote visualization environment for an electromagnetic scattering simulation on a high performance computing system. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 317–326, 1993.

[5] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Witlock, and N. Max. A contract based system for large data visualization. In *Visualization 2005*, pages 191–198, 2005.

[6] J. A. Clarke, C. E. Schmitt, and J. J. Hare. The distributed interactive computing environment. In *Proc. High-Performance Computing Workshop*, July 1998.

[7] S. Doi, T. Takei, and K. Matsumoto. Experiences in large-scale volume data visualization with RVSLIB. *Computer Graphics*, 35(2):10–13, May 2001.

[8] ECCO. `http://www.ecco-group.org/`.

[9] FSU "superensemble" approach to predicting hurricanes' path. `http://www.research.fsu.edu/scicol/3/teest.html`.

[10] G. A. Geist, II, J. A. Kohl, and P. M. Papadopoulos. CUMULVS: Providing fault-tolerance, visualization and steering of parallel applications. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):224–235, 1997.

[11] Growler. `http://www.nas.nasa.gov/~bgreen/growler.html`.

[12] R. Haimes and D. E. Edwards. Visualization in a parallel processing environment. In *AIAA Paper 97-0348, Proceedings of the 35th AIAA Aerospace Sciences Meeting*, Jan. 1997.

[13] W. Hibbard, C. Rueden, S. Emmerson, T. Rink, D. Glowacki, T. Whittaker, D. Murray, D. Fulker, and J. Anderson. Java distributed components for numerical visualization in VisAD. *Commun. ACM*, 48(3):98–104, 2005.

[14] C. Johnson, S. G. Parker, C. Hansen, G. L. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *IEEE Computer*, 32(12):59–65, Dec. 1999.

[15] J. T. Kiehl, J. J. Hack, G. B. Bonan, B. A. Boville, B. P. Briegleb, D. L. Williamson, and P. J. Rasch. Description of the NCAR Community Climate Model (CCM3). Technical Report NCAR/TN–420+STR, National Center for Atmospheric Research, Boulder, Colorado, 1996.

[16] J. C. Lakey and R. J. Moorhead, II. Concurrent visualization of time varying CFD simulations. In *Proc. SPIE Vol. 2178 Visual Data Exploration and Analysis*, pages 123–126, Feb. 1994.

[17] S.-J. Lin. A vertically Lagrangian finite-volume dynamical core for global models. *Monthly Weather Review*, 132:2293–2307, 2004.

[18] MAP05. `http://map05.gsfc.nasa.gov/`.

[19] B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6):1–14, Nov. 1987.

[20] H. Okuda, K. Nakajima, M. Iizuka, L. Chen, and H. Nakamura. Parallel finite element analysis platform for the Earth Simulator: GeoFEM. In *Proc. 3rd ACES Working Group Meeting*, June 2003.

[21] T. A. Sandstrom, C. Henze, and C. Levit. The hyperwall. In *Proc. Conference on Coordinated and Multiple Views in Exploratory Visualization*, pages 124–133, July 2003.

[22] A. Vaziri, M. Kremenetsky, M. Fitzgibbon, and C. Levit. Experiences with CM/AVS to visualize and compute simulation data on the CM-5. In *1994 AVS Users Conference Proceedings*, May 1994. Also available as NASA Technical Report RNR-94-005.