Chapter 7

Remote Visualization of Cosmological Data Using Salsa

Orion Sky Lawlor

Department of Computer Science, University of Alaska Fairbanks

Thomas R. Quinn

Department of Astronomy, University of Washington

7.1 Introduction

As discussed in the previous chapter, the experimental cosmology application ChaNGa simulates particle datasets consisting of tens of millions to billions of particles. Salsa, the visualization tool we describe in this chapter, is the simulated observatory intended to help extract useful information from the simulation results.

There are two key challenges in extracting scientific results from simulations of cosmology. First, the natural dynamic range of cosmology runs from hundreds of megaparsecs, which are required to get a "fair sample" of galactic environments, to star formation environments in individual galaxies with scales of parsecs. This later scale is required because it is primarily the light from the stars or the supernovae by which we get the observational constraints on our cosmological models. Simulating such a large dynamic range leads to the generation of large datasets. The second challenge is that galaxies and galaxy clusters are non-trivial 3-dimensional objects. They sit in triaxial halos; they are characterized by their morphology which is a complicated interaction of components with different shapes; they are formed via a complicated

merger process. Interpreting the flatness of the disk, the strength of the spiral arms, the size of the bulge, the presence of a bar and the warping and bending of any of these components requires the ability to grasp the full 3-dimensional structure of the galaxy. This information is most effectively conveyed to the investigator by rendering the model at a smooth interactive rate.

Unfortunately, these two challenges are incompatible at face value. The large dynamic range results in datasets that run into the terabytes, while interactive rendering is limited to a few gigabyte datasets even with high-end graphics workstations. The memory requirements alone require parallel architectures which do not lend themselves to interactive use. Furthermore these intrinsic problems are compounded by the fact that, in a typical situation, the simulation datasets are computed at a supercomputing center, far away from the application scientist and connected to him via a relatively slow internet link, running at only megabytes per second.

In this chapter, we show how we designed Salsa to visualize these large multi-gigabyte multidimensional datasets over a relatively slow network link, and provide interactive analysis capabilities to the scientist. This design makes use of a number of capabilities of the Charm++ framework, as well as the construction of a client application that exploits the graphics capabilities of desktop workstations. To demonstrate the utility of our work, we end with a description of how Salsa was used in a recent galaxy cluster cosmology project.

A separate data analysis challenge is to be able to interactively query this large simulation result for quantitative results. Furthermore, coupling such queries to the interactive visualization can be very productive; e.g, "What is the mass of the cluster that I just selected with the graphical interface?" To address this challenge, Salsa has the capability of both querying and manipulating the simulation data via a Python based interface. For a more complete description of this capability, we refer the reader to a previous publication [79].

7.2 Salsa Client/Server Rendering Architecture

Generally, ChaNGa is run on thousands of processors, which are needed to provide the computational capacity to perform the simulation in a reasonable time. Large processor allocations like this are typically batch scheduled, which is not convenient for interactive scientific data analysis. Luckily, data visualization has lower computational complexity than simulation, so we can normally load the simulation data onto fewer processors for visualization and analysis, typically a few dozen processors, small enough to be allocated for interactive use. For the analysis of a very large dataset, for example with hundreds of billions of particles, in principle we could load data across tens of thousands of processors, since our visualization infrastructure is designed

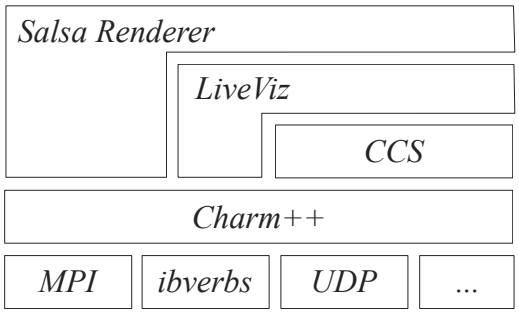

FIGURE 7.1: Software layer diagram for Salsa server-side renderer.

to scale well. But in any case, to display anything useful to the application scientist, we need to send data to a local client machine.

As shown in Figure 7.1, Charm++ provides several useful tools to solve this remote visualization problem. The Charm++ network protocol Converse Client Server (CCS) [113, 47] allows a Charm++ program to act as a network server and allow authenticated connections from clients via the network. On top of CCS, the application-layer image assembly protocol LiveViz [46] allows individual array elements to provide fragments of a 2D image, which LiveViz assembles across processors, compresses and transmits to the client via CCS.

Our overall data movement during runtime is thus shown in Figure 7.2. We typically need several dozen gigabytes of RAM to store the entire particle dataset, so we leave this parallel particle list distributed across the visualization cluster. The exact distribution of particles across processors can be transparently adjusted at runtime using the Charm++ object migration and load balancing libraries. To display this dataset on the client, we begin by rendering each array elements' particles into either a 2D image or a 3D volume impostor—a low-resolution stand-in for the actual data, described in the next section. These rendered images are then composited hierarchically at the core, node and inter-node levels across the cluster using the LiveViz library. The assembled image is then compressed and sent to the client, as described in Section 7.2.2. The Java client application maintains the CCS connection to the server, decompresses the received image data, and renders it using the local graphics card via JOGL, a Java binding for OpenGL graphics calls.

7.2.1 Client Server Communication Styles

In theory, the data analysis server could send any arbitrary data to the scientist's desktop machine. For example, the server could simply send fully-computed numbers, such as the measured galactic density fall-off exponent. For this style, Salsa supports a Python-based programming interface for writing and executing new data queries at runtime. The primary drawback of

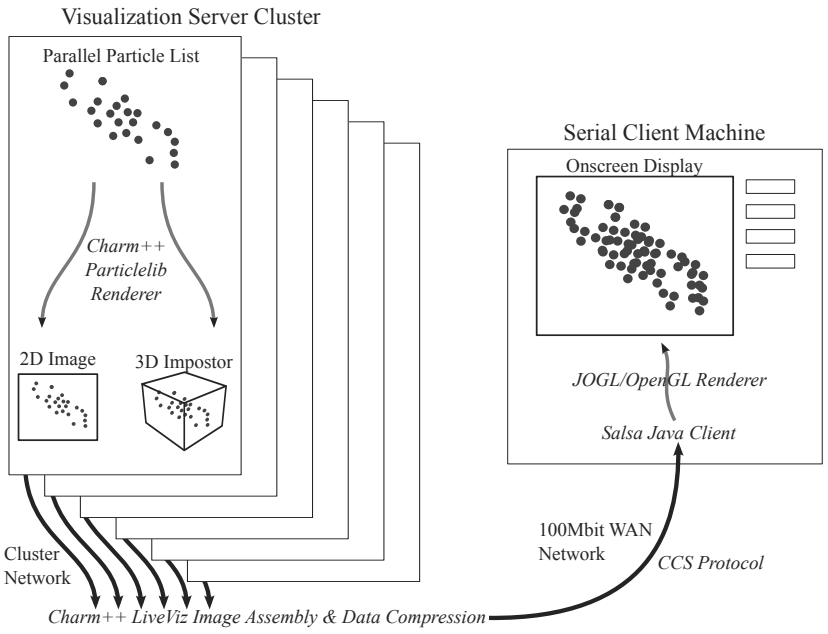

FIGURE 7.2: Data movement in Salsa. Particles are distributed across the cluster, rendered to 2D image or 3D impostor fragments, composited across the cluster, compressed and sent across the network to the client.

sending only numbers is summed up by Hamming's quote [97], "The purpose of computation is insight, not numbers." Visualization is useful because the story of scientific progress often begins with "Oh no!" rather than "Ah ha!" moments, and it isn't always obvious where to begin analysis. The questions are as important as the answers, when trying to discover aspects of a dataset that you don't yet know to look for.

The server could send simple 2D images of the particle dataset, as viewed from some particular 3D viewpoint and scale factor. This is currently our primary method of interacting with data in Salsa. Sending images is general purpose, and leverages the spectacular data analysis capacity available in the visual cortex of the human brain. But one difficulty with 2D images is that changing the 3D viewpoint requires a round-trip to the server—the client requests a new viewpoint, the server renders the dataset from the new viewpoint and the image must be compressed and transmitted back to the client. This process can take several seconds, which makes it difficult to understand the 3D structure of the data. Astrophysicists are used to this problem in observational data, where it is difficult to estimate the 3D structure of objects billions of light years away, but in a simulation we should be able to do better.

To allow the scientist to interactively move through a 3D dataset, the server could send a 3D representation of the data. For tiny datasets, we could simply download the entire dataset, but most realistic datasets require more storage and rendering capacity than are available on most desktop machines. Random subsampling is promising, but tends to over-represent dense regions near galaxy and supercluster cores, and under-represent more distant regions. Hybrid approaches, such as mixed volume/particle methods [158], create difficult to tune tradeoffs between the particle and volume areas.

We have extensively explored volume rendering, where the particle dataset is sifted into a 3D voxel grid, and the resulting volume dataset sent to the client, where it can be explored in 3D with no further communication. We call this approach "volume impostors" as a generalization of the well known 2D texture-based "impostors" technique [159]. There are several advantages to volume rendering: for example, volume datasets compress well, with a 128 million voxel 512^3 image compressing to only a few megabytes. Surprisingly, even a 2007 desktop graphics card can interactively render a dense 512^3 volume image at over 20 frames per second. Salsa currently supports 3D volume impostors, but there are difficulties with this approach. The graphics hardware can most efficiently render from a regular 3D grid, which spans only a limited region of space; this means to get more detail in one part of the simulation, another area must be clipped away. Salsa maintains a "point of interest" as the user pans and zooms, to reduce this effect. Further, when the 3D viewpoint is stationary, the client requests a perfect 2D rendering including all the particles, and displays this as soon as it arrives, combining the interactivity of the 3D volume with the quality of the 2D image (see Figure 7.3).

7.2.2 Image Compression in Salsa

For the long trip over the network to the client, LiveViz supports several image compression mechanisms. The CPU-limited[1] throughput and bandwidth savings for several possible image compression algorithms are summarized in Table 7.1. Overall, only run length encoding is efficient enough to be useful on a gigabit network (100MB/s), though on slower networks GZIP can be a net benefit. While lossy compression methods like JPEG can be competitive in time and space efficiency, the resulting distortions are usually not acceptable, especially when scientific data is plotted in false color. We have shown that a careful GPU implementation of run length encoding can double the compression throughput compared even to multicore [211], but normally rendering is the bottleneck.

[1]The CPU in this table is one processor of a 3.1GHz Intel Core i5-2400.

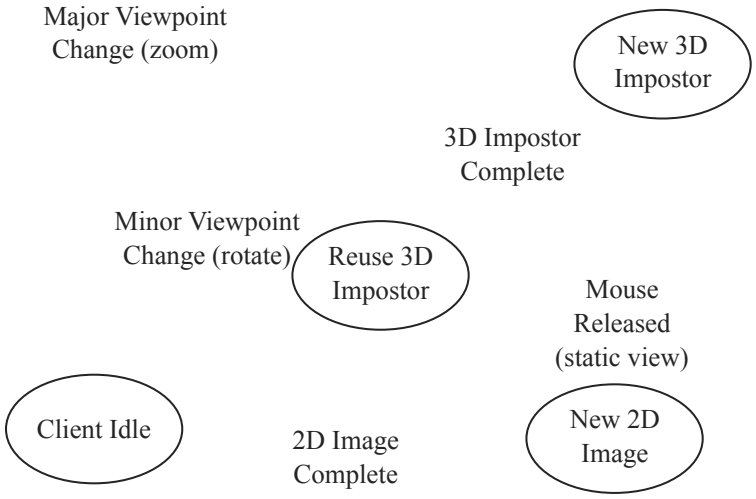

FIGURE 7.3: Finite state machine for network communication on the client.

7.2.3 GPU Particle Rendering on the Server

Since the server's job is to render a large array of particles into an image, it is natural to consider using the graphics processing unit (GPU) to accelerate rendering. There are a number of complications with this, however.

First, the particle data is loaded on the CPU, and must be copied to the GPU. This copy time exceeds the time to render the data, so this is affordable only if the time to copy the data onto the GPU can be amortized across several frames. Hence Salsa tracks changes to the particle data such as when the user selects a new attribute, coloring or dataset otherwise Salsa leaves the particle

Algorithm	Compress (MB/s)	Savings (%)	Decompress (MB/s)	SNR
Run Length (RLE)	91	40	125	*lossless*
DEFLATE (GZIP)	16	80	67	*lossless*
DEFLATE (PNG)	9	80	59	*lossless*
Discrete Cosine (JPEG)	25	75	33	20dB

TABLE 7.1: Compression algorithm comparison for Salsa server-to-client images. Performance is shown for a representative 1000×1000 pixel grayscale image.

GPU Data Writes	Performance (Gop/s)
Naive byte writes	3.1
Software atomic byte max	0.6
Naive int writes	4.9
Atomic int max	4.4

TABLE 7.2: Memory write performance on NVIDIA GeForce GTX 580 graphics card. Speeds are in billions of memory operations/second. Note that naive writes will produce incorrect answers when particles overlap.

data on the GPU. In particular, multiple 2D viewpoints or 3D zoom factors can be rendered by the GPU without needing to copy the data each time.

Second, because 3D particles can be drawn to arbitrary locations in the 2D framebuffer, it is not obvious how to render multiple points in parallel—there is a race condition whenever two points try to modify the same pixel. Although modern GPU hardware supports atomic memory operations quite efficiently, atomic operations are only available for integer or float data types. Typically images use byte pixels, for which the hardware does not provide atomic memory operations. We resolve this by promoting pixels to full integer width, which allows us to use the hardware native atomic operations. As shown in Table 7.2, this is a higher performance solution than simulating atomic byte memory operations in software.

Finally, write contention can be a significant GPU performance problem when several 3D points project to the same 2D pixel, requiring the parallel hardware to serialize the pixel writes at runtime. Astrophysics data, where points gravitationally cluster and collapse into very small regions, causes especially poor write contention when rendering the dense core of galaxies. On a single core CPU, this is not a performance issue, because the pixels covered by the galaxy core stay in cache; but these areas are a serious performance issue on either multicore CPU or the GPU.

We have found that declustering the original particle data, by randomizing with a Fisher-Yates shuffle during upload, reduces memory contention during the atomic pixel write operations. This, in turn, improves runtime rendering performance over threefold for modern GPU hardware. Surprisingly, this effect persists even in modern cached GPU memory architectures such as NVIDIA's Fermi. It is surprising that on modern parallel hardware, high data access locality can actually create contention, decreasing performance. Instead, randomizing memory accesses can reduce memory contention and improve performance—this is the exact opposite of the principle of locality to improve sequential cache performance!

7.3 Remote Visualization User Interface

Figure 7.4 shows the Salsa user interface. On the right is a Java Swing interface for selecting particle families and fields of the particle to visualize. On the left is a 3D rendering shipped across the network from the server.

An interesting network design issue arises when the user manipulates the 3D view. As the user drags the mouse to rotate the image, the operating system reports the mouse position about a hundred times per second. Although we can easily send out a hundred new image requests per second across the network, with billions of particles even a large parallel machine cannot render them this quickly, and often the network cannot deliver frames this rapidly either. This asymmetry makes it easy for the client to unintentionally overload the server and saturate its own network link; so, to maintain responsiveness, we must limit the number of image requests sent. A hardcoded or user-selectable limit on the number of outgoing requests per second does not scale well to the variety of parallel rendering hardware, particle counts and network links encountered in reality; but we find that a limit on the number of requests outstanding self-adapts to these limitations. With a limit of one outstanding request, we effectively have a "last in, only out" queue of size one. This ensures that as soon as the server is free, we immediately send it the most up-to-date request [146].

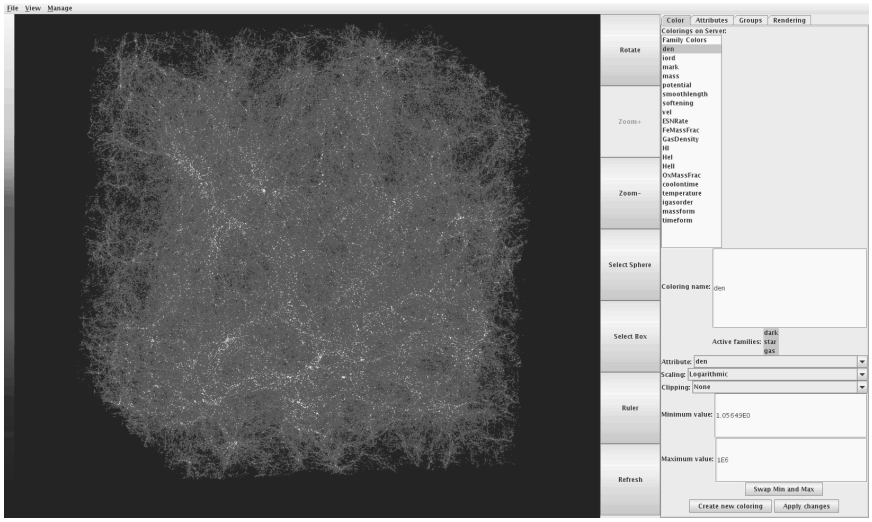

FIGURE 7.4: Screen user interface screenshot, showing the density field for a 2-billion particle simulation. The data is in Oak Ridge, Tennessee; the client is in Seattle, Washington.

7.4 Example Use: Galaxy Clusters

Here we give an example of the use of Salsa for a specific cosmology project: understanding observations of galaxy clusters. Clusters of galaxies are excellent probes of cosmology for a number of reasons. First, they are sensitive probes of the growth of cosmic structure because their number has evolved rapidly in the recent past. Second, since they are the largest gravitationally bound objects in the Universe, they can be observed over a large range of cosmic time. One effective method for detecting clusters at very large distances is through the distortions they produce in the Cosmic Microwave Background (referred to as the Sunyaev-Zeldovich Effect, SZE, [249]); however, many clusters found this way seemed to have disturbed morphology [198], perhaps indicating a selection bias. In the work described below, we use ChaNGa simulations of a galaxy cluster to follow the evolution of the SZE through the growth of a large galaxy cluster. This work is described more fully in [210]. Here we point out the use of Salsa at various stages in this project.

The first stage of simulating the growth of a galaxy cluster is picking a representative cluster. This usually requires simulating a "fair sample" of the Universe, identifying the clusters, and selecting one of the desired mass and formation history. Figure 7.5 shows a slice through such a simulation rendered with Salsa. Cluster identification was done by a separate program that implements the "friends-of-friends" algorithm [43] and produces a cluster ID for each particle. This program was run for each of 30 snapshots in time. Salsa ran a python script that successively read in each of these data files, calculated the cluster masses and, for each particle, calculated the ratio of the mass of the cluster it was currently in to the mass of the cluster it was in at the end of the simulation. A formation time was defined to be when that ratio crossed a chosen threshold (e.g., 75%). While this calculation did not need the visualization capabilities of Salsa, having a parallel framework controlled by a high level scripting language made it easy to program and quick to run.

Once a cluster of suitable mass and formation history is selected, the gas dynamics is followed in high resolution by determining the Lagrangian region from which the cluster formed and resimulating that region with much greater resolution. Salsa is helpful for several aspects of this process. First, determining the quality of the initial conditions involves verifying that the entire region from which the cluster formed is represented by high resolution particles, as well as a "buffer" region surrounding it. This is easiest done by visual inspection of the particle distribution. Then after the simulation is performed, its quality can again be assessed by investigating whether the cluster is "contaminated" by lower resolution particles. As well as visual inspection, quick subselections of particles via python scripts were used to determine whether this was the case. The final output of the gas dynamical simulation of the cluster is shown in Figure 7.6.

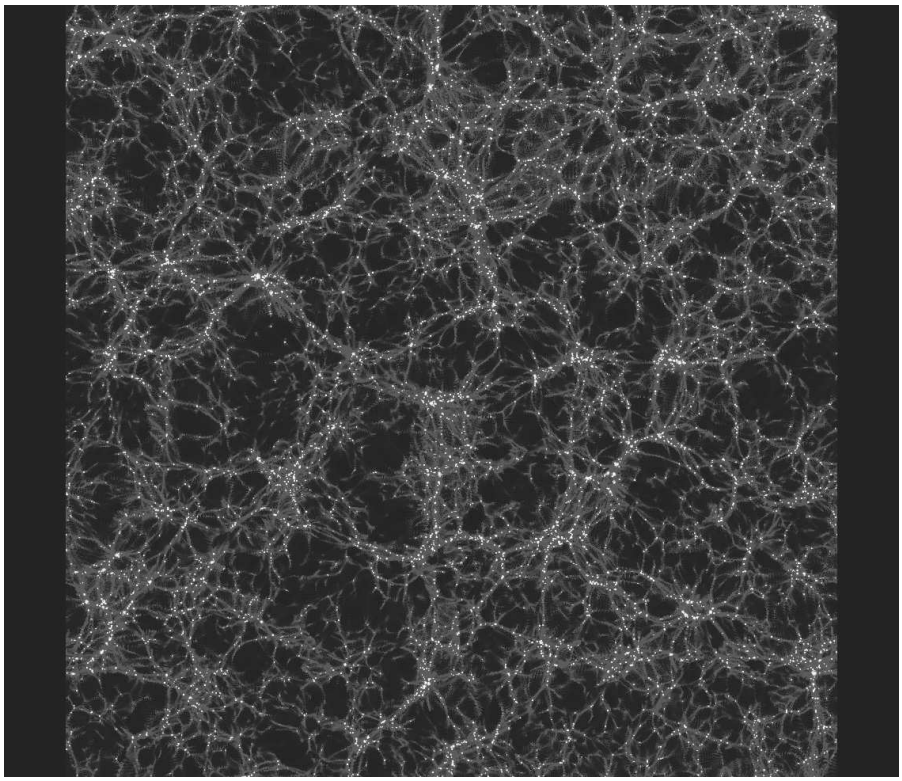

FIGURE 7.5: Salsa rendering of the dark matter density distribution in a simulation of galaxy cluster formation **(see Color Plate 6)**.

Finally, the simulation is analyzed to produce the scientific results. The nature of the final merger that produced the cluster was determined by visually inspecting the cluster in a number of snapshots to identify the two pre-merger clusters, and calculating their mass. Some of the quantities to be measured are strongly dependent on the direction from which the cluster is viewed: substructure most easily identifiable when the apparent separation from the cluster center is maximized, a kinetic component of the SZE is a function of the line-of-sight velocity of the substructure. Salsa was used to interactively discover these optimum viewing angles. Simulated observations of the cluster were made with Python scripts that binned the particles into instrumental pixels and calculated the appropriate summation of the particle quantities.

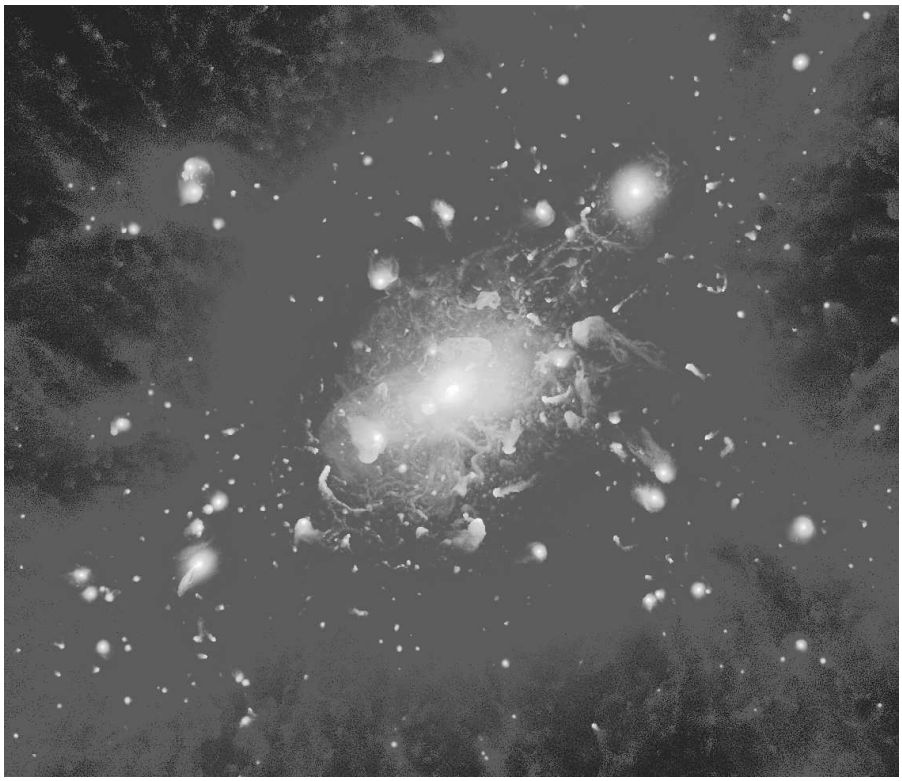

FIGURE 7.6: Salsa rendering of the gas density within a cluster of galaxies **(see Color Plate 6)**.

Acknowledgments

The construction of Salsa was supported by a grant from the NASA Applied Information Systems Research effort and a grant of HPC resources from the Arctic Region Supercomputing Center and the University of Alaska, Fairbanks.