Interactive Terascale Particle Visualization

David Ellsworth*
Advanced Management Technology, Inc.
NASA Ames Research Center

Bryan Green
Advanced Management Technology, Inc.
NASA Ames Research Center

Patrick Moran
NASA Ames Research Center

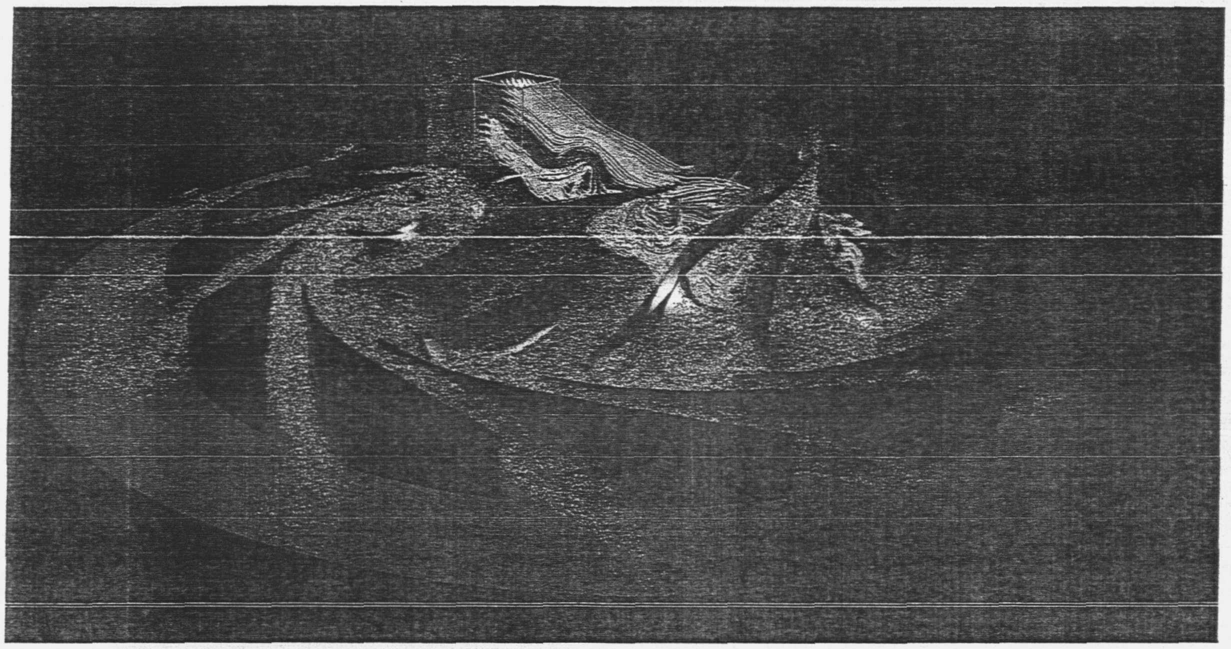

Figure 1: Streakline visualization of a 2 TB liquid hydrogen turbopump data set.

Abstract

This paper describes the methods used to produce an interactive visualization of a 2 TB computational fluid dynamics (CFD) data set using particle tracing (streaklines). We use the method introduced by Bruckschen et al. [2001] that precomputes a large number of particles, stores them on disk using a space-filling curve ordering that minimizes seeks, and then retrieves and displays the particles according to the user's command. We describe how the particle computation can be performed using a PC cluster, how the algorithm can be adapted to work with a multi-block curvilinear mesh, and how the out-of-core visualization can be scaled to 296 billion particles while still achieving interactive performance on PC hardware. Compared to the earlier work, our data set size and total number of particles are an order of magnitude larger. We also describe a new compression technique that allows the lossless compression of the particles by 41% and speeds the particle retrieval by about 30%.

*e-mail: {ellswort,bgreen}@nas.nasa.gov, patrick.j.moran@nasa.gov

CR Categories: I.3.8 [Computer Graphics]: Applications; E.4 [Coding and Information Theory]: Data compaction and compression; D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming

Keywords: visualization, particle tracing, large data, out-of-core, PC hardware, clusters, computational fluid dynamics

1 Introduction

Interactive visualization of data sets containing a terabyte or more is difficult or impossible to do even on the largest systems. Very few systems have enough memory to store the data in memory. Out-of-core visualization using traditional visualization algorithms is impossible since the data rates of tens of gigabytes per second necessary are currently not possible. However, it is currently quite possible to generate multi-terabyte data sets of CFD or physics calculations on today's supercomputers. In addition, using PC-class hardware for the visualization is desirable since this allows scientists to examine their results on their desktop.

One approach that scales to large data sets is to precompute the visualization. In most cases, the resulting geometry can be displayed interactively. For time varying data, a visualization can be computed for each time step, and, if not too large, can be animated. Otherwise, each frame can be rendered beforehand and shown as a static movie. However, all of these methods suffer from a lack of interactivity since the visualization computation must be repeated whenever the visualization parameters (particle seedpoint, isosurface value, etc.) are changed.

The approach introduced by Bruckschen et al. [2001] for interactive particle visualization does not have this limitation. By computing a large number of streaklines from a regular grid of seedpoints and storing them on disk, a subset of the traces can be retrieved and viewed interactively. This approach stores the traces on disk in a format that allows the streaklines to be read from disk quickly. Each trace is stored contiguously. In addition, the traces are written to disk in the order of a Morton space-filling curve [Sagan 1994], also known as a Peano or z-curve. This ordering reduces the number of disk seeks required to retrieve a 3D box of seedpoints.

In this paper, we describe several extensions to this work and the results from applying the resulting system to a 2 TB CFD simulation of a liquid hydrogen turbopump (see Figure 1). We extend the approach to allow for particle advection through a data set defined on a multi-block curvilinear grid. We also describe how the particle advection can be computed on a Beowulf cluster with a limited amount of memory per node and how the particle data can be compressed by about 40%. Finally, we describe a viewer implementation that interactively retrieves particles from a file server. The viewer uses a server process that runs on one or more file servers, retrieves particle data, and sends it to a display process running on a workstation. The viewer prefetches data from one or more file servers for increased performance.

2 Related Work

Visualization of large data sets has been an area of active research. A commonly used technique is to precompute the visualization by saving a series of images or sets of geometry. Two of the many systems that use precomputation are IBM Visualization Data Explorer (now OpenDX) [Abram and Treinish 1995] and UFAT [Lane 1994]. Out-of-core visualization is another approach to handling large data sets. Chiang et al. [Chiang et al. 1998] propose a fast out-of-core technique for extracting isosurfaces using a precomputed disk-resident index; Chiang [2003] has recently extended the technique to handle time-varying data. A different out-of-core technique is to load only the portion of the data needed to produce the visualization via demand paging [Cox and Ellsworth 1997]. While this technique supports particle tracing and other visualizations, it does not allow interactive visualization of terascale data sets. Ueng et al. [1997] have implemented a different out-of-core particle tracing system that works with unstructured meshes.

A different large data visualization technique is to stream the data through a series of filters that produce the visualization, as proposed by Ahrens et al. [2001]. This technique scales to handle very large data sets, and can be run in parallel. It should allow interactive visualization if the data are not too large and the visualization is computed on a sufficiently large system. However, streaming systems are not suitable for particle tracing because streaming requires a priori knowledge of the data access pattern, which is not available with particle tracing. Finally, Heerman [1999] documents many of the issues encountered when dealing with terascale data on a day-to-day basis.

3 Algorithm Overview

Our visualization approach has two phases that run at different times. The particle computation application runs as a preprocessing step, and the viewer application is used for the interactive visualization. The computation application uses the input data set and writes metadata and particle traces to disk. The viewer application shows the particle traces to the user.

0	1	8	9
2	3	10	11
16	17	24	25
18	19	26	27

top plane

4	5	12	13
6	7	14	15
20	21	28	29
22	23	30	31

second plane

32	33	40	41
34	35	42	43
48	49	56	57
50	51	58	59

third plane

36	37	44	45
38	39	46	47
52	53	60	61
54	55	62	63

bottom plane

Figure 2: 4×4×4 Morton curve.

0	1	8
2	3	9
12	13	16

top plane

4	5	10
6	7	11
14	15	17

bottom plane

Figure 3: 3×3×2 Morton curve.

The particle data are organized as a series of files, one per time step. Each file has the streaklines computed for each seedpoint stored contiguously, which allows the streakline to be read with one disk read. Furthermore, the particle traces are placed in the file according to their Morton order [Sagan 1994], which means that traces for seedpoints near to each other in physical space are usually near to each other in the particle file, further reducing the number of disk seeks.

The Morton order is based on a space-filling curve, and is the same as the ordering seen when performing a depth-first traversal of an octree's leaf nodes. Figure 2 shows the Morton order of a 4×4×4 cube. Because the Morton order is only defined for cubes with powers-of-two sizes, we use a modified Morton order that handles arbitrary dimensions. This order is the same order as the one you would get if you traversed a cube that was the smallest power of two possible enclosing the desired array, but did not count elements outside the array. Figure 3 shows an example. The earlier implementation [Bruckschen et al. 2001] has more details on how the Morton order reduces disk seeks.

Each file has a header giving the length of each trace, followed by the particle traces. Unlike Bruckschen et al.'s implementation, which uses a single trace length for each file, we choose to store variably-sized traces, which only contain particles remaining in the domain. While using a single trace length simplifies the data access and makes saving the particle trace lengths unnecessary, it would have increased the amount of uncompressed particle data by about 33% or 700 GB. We could have limited the excess storage by limiting the maximum trace length, but doing so would limit our ability to determine the amount of recirculation and particle mixing, an important CFD visualization task. Compressing single-length particle traces would reduce the amount of extra storage, but we have not investigated this.

We follow Bruckschen et al. by compressing the particles' 32-bit floating point coordinates to 16 bits. The 16-bit values are computed by subtracting one corner of the mesh's bounding box, dividing by the size of the bounding box, and quantizing the resulting fractions to 16 bits. Given the resolution of current screens, the resulting particle coordinates should place the particles on the screen with a position error smaller than a pixel unless the view only shows

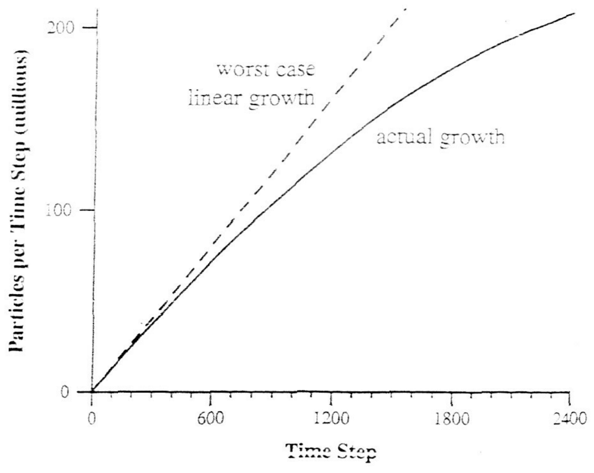

Figure 4: Number of particles calculated per time step.

a very small fraction of the overall data set.

Since we do not limit the length of particle traces, the worst-case total number of particles over all the time steps is proportional to the square of the number of time steps. As shown in Figure 4, the number of particles per time step is not very far from the worst case, which results in large particle files and very long computation times (see Section 4).

3.1 Curvilinear Data

Computing particles in a data set using a regular grid [Bruckschen et al. 2001] is somewhat simpler compared to a curvilinear grid. Particle integration requires retrieving velocity values at arbitrary points in physical space. This is straightforward with regular grids, but is more complicated with multi-block curvilinear grids. These grids require point location code to find the cell enclosing the requested physical location, and additional code to resolve cases where multiple grids overlap. We use the Field Model library [Moran 2001] for accessing velocity values, which simplifies the retrieval from an application's point of view to a single function call once the grid has been read.

An additional complication is that the domain of curvilinear grids are much more irregular than regular grids, which means that finding the seedpoints for the particle integration requires a bit more work. Like Bruckschen et al.'s [2001] implementation, we use a regular grid of seedpoint locations. However, the regular grid of seedpoints is usually evenly spaced throughout the bounding box of the mesh (the user can specify a different box of seedpoints if an area is of particular interest). With many curvilinear grids, most of these *initial* seedpoints are outside the grid. In our data set, only 14% of the initial seedpoints are inside the grid. We find the seedpoints inside the grid, the *active* seedpoints, at the start of the computation by testing whether each initial seedpoint is inside the grid domain. Our grid varies over time, which means that holes in the grid that correspond to the interior of a turbine blade can move over time. Thus, we test each initial seedpoint against a number of different grid time steps; seedpoints inside any time step are considered active seedpoints. We test twenty time steps each spaced three time steps apart (i.e. time steps 0, 3, 6, ..., 57). The number and spacing of time steps to check is configurable since it is data set dependent.

Once the validity of each initial seedpoint has been determined, the computation algorithm writes out a metadata file. This file has the dimensions of the initial seedpoint grid, the box containing the initial seedpoints, the grid bounding box, the number of active seedpoints, and a value for each initial seedpoint. This value is −1 if the seedpoint is not active, and is the number of the active seedpoint otherwise. This array of values is needed because the particle files only contain traces for the active seedpoints; otherwise the viewer application would not be able to determine which selected initial seedpoints have particle data, nor the position of the valid seedpoints within the particle files.

Another option with curvilinear grids is to place seedpoints evenly in computational space. Computational space seeding is clearly desirable when the scientist would like to see particles placed around a moving object, such as a rotating turbine blade. Computational seeding might be considered superior because the seedpoint density follows the cell density, although the density of particles in cells will not be constant once the particles have been advected a significant distance. Our current implementation does not support computational seeding, but may in the future.

Size of grid files	381 MB
Size of solution files	476 MB
Number of time steps	2400
Total data set size	2055 GB
Initial seedpoint grid size	$35\times167\times167$
Number of initial seedpoints	976,115
Number of active seedpoints	136,111
Total number of particles	296 billion
Particle storage (uncompressed)	1777 GB
Particle storage (compressed)	1038 GB

Table 1: Initial data set and particle data statistics (M=10^6, G=10^9).

4 Particle Computation

The particle computation was done on a 49-node Beowulf cluster (see Section 7). Dividing up the particle computation among the nodes was straightforward since the calculations for each seedpoint are independent. However, getting the data to the calculation was difficult since we obviously cannot load the entire 2 TB data set into each node's 1 GB of memory, or even on each node's 100 GB disk. The amount of data needed at once can be greatly reduced by only loading pairs of time steps (both grid and solution) at a time, advancing the particles, and then loading the next time step [Lane 1994]. However, this would still require 1.7 GB of memory, and would cause a lot of page swapping and low CPU utilization. We use several techniques to reduce the data and memory requirements, as described below.

We use the following system architecture. The input CFD data are stored on a file server with 4.5 TB of disk. Each node reads the input data it needs from the file server, and sends the computed particles to the master node. The master node buffers the computed particles and writes them to each output file in order. The output files are stored on the file server, and written via NFS.

The seedpoints are divided into 196 equal-sized chunks (4 chunks per node). Each chunk of particles is a contiguous section of the active seedpoints, sorted by Morton order. Using sections of contiguous seedpoints means that the particles do not fill the entire domain. This reduces the amount of solution data that must be loaded via demand paging, described below. Seedpoints in different areas of the domain will need different amounts of computation. For example, seedpoints near the domain exit will have shorter traces than ones near the domain entrance. We give each node 4 disparate chunks of seedpoints to reduce load imbalances caused by this effect.

We reduce the memory usage via several techniques:

- Loading only a pair of time steps at a time, as described above.
- Reducing the particle memory footprint. This is significant since each node will have about 4 million particles at the end of the computation. Our initial particle tracing code was very general (allowing particles constrained to a computational plane, streaklines or streamlines, etc.) and stored several intermediate results for increased speed, and used over 200 bytes per particle. We reduced the memory to 24 bytes per particle by removing unnecessary features and not storing the intermediate results.
- Exploiting the regularities in the grid, as described below.
- Using an out-of-core algorithm to load solution data via demand paging, also described below.

4.1 Exploiting Mesh Regularities

The mesh has 35 blocks, or *zones*, each representing a part of the domain. Many of the zones contain vertices near various features, such as turbine blades. Some of the zones rotate over time, such as those surrounding the rotating blades of the turbine (see Figure 1). Other zones do not change over time. Furthermore, many of the zones are rotated copies of other zones in the same time step, such as the zones that contain points around each of the blades.

We used the techniques described in [Ellsworth and Moran 2003] to find the regularities described above and replace the time-varying mesh (contained in 2400 mesh files) with a replacement mesh that requires that only 46% of one mesh file be loaded. The replacement mesh uses three zones that refer to static data, four zones that rotate over time, 15 zones that rotate over time and can reuse the vertices from another zone, and 12 zones that are static and can use rotated vertices from another zone. All of the zones have unique IBLANK data (per-vertex flags that indicate validity and correspondences between zones) that do not vary over time. Thus, all the IBLANK flags from a single time step must be loaded. Using the replacement mesh cuts down the amount of mesh data by over a factor of 5000. (However, not all of the original mesh data would need to be loaded if the demand paging techniques described below are used.)

4.2 Demand-Paging Solution Data

Because each cluster node computes particles for seedpoints clustered in a few parts of the domain, each node does not access all of the solution data (the velocity values). We avoid loading unnecessary solution data by using demand paging. This technique divides each solution file into a number of fixed-size blocks. When a solution file is opened, a data structure is created that indicates whether a given block is present, and points to the block if it is. Then, when solution data are requested, the data retrieval code checks whether the corresponding blocks are present, loads the blocks if not present, and then retrieves the requested data from the blocks. The data blocks are allocated from a fixed-size pool of blocks. If a block is needed when all the blocks are allocated, an in-use block that has not been recently accessed is chosen and reused. Each block contains an $8 \times 8 \times 8$ cube of data, which reduces the number of blocks needed compared to blocks of data organized in standard array order. More details about this technique can be found in [Cox and Ellsworth 1997].

Unfortunately, the computation must wait while a block is loaded from the file server. We reduce this waiting by using a number of different threads, each working on a trace from a different seedpoint. When a thread starts waiting for a block of data, a different thread is made runnable (if one is available) so the processor is kept busy. This multithreading technique is also used to provide work for the two processors in each node. We speed the retrieval of blocks from the file server by using a custom protocol that allows multiple outstanding read requests, and only sends the requested data. Using this protocol increases the speed compared to using NFS. See [Ellsworth 2001] for more details about the multithreading technique, the remote protocol, and their associated speedups.

The particle computation algorithm reduces the latency due to loading solution data by prefetching blocks when a new time step is started. If the previous time step calculation used files t and $t+1$, the new time step calculation will use files $t+1$ and $t+2$. When the new step is started, a separate thread finds which blocks are currently present for file $t+1$ and quickly loads them for file $t+2$. This prefetching is effective because there is a high correlation between the particle positions, and hence the blocks used, in adjacent time steps. However, this prefetching scheme will continue to load blocks that were used in one time step and are not used in future time steps. To reduce the amount of unused prefetched data, no prefetching is used for every tenth time step.

4.3 Particle Computation Performance

The memory reduction techniques result in significant time savings. Using the replacement meshes reduced the computation time by roughly 25%. Prefetching solution blocks saves about an additional 10%. Loading the solution data via demand paging results in loading only 25 to 45% of the data.

However, the current implementation is still slower than we would like: the particle computation for the 2 TB data set took 13 days. We believe that significant optimizations are possible because the CPU utilization is less than 50%. One possible cause of the low CPU usage is that the nodes are waiting for solution data from the file server. A second possible cause is that the operating system does not work well with the multithreading algorithm (used to keep the CPU working while data are being loaded) because the algorithm switches contexts at a very high rate. We are currently investigating the cause of the low CPU utilization.

Another way to speed the computation would be to use an SMP system instead of a cluster. The SMP system could easily prefetch entire time steps and hold them in memory, which would remove the performance impact of the demand paging. Finally, implementing dynamic load balancing is another way to increase performance.

5 Compression

Since the computed particles use a large amount of disk space, even after the 16 bit quantization, we have explored compressing the particle files. We used a prediction preprocessing step that reduced the entropy of the data as well as the zlib compression library [Deutsch and Gailly 1996], used by gzip. The prediction step uses particles earlier in each trace to compute the predicted value, and the difference between the actual and predicted value is compressed. We tried several different prediction methods as well as different zlib compression settings. All of the compression methods are lossless.

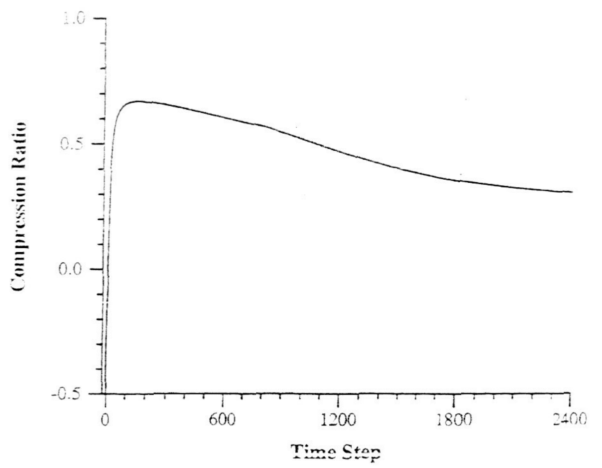

Figure 5: Compression ratio of saved particles by time step $((s_{orig} - s_{comp})/s_{orig})$, where s_{orig} is the original size of the particle traces, and s_{comp} is the compressed size.

The prediction methods tried were no prediction, and zeroth, first, and second order prediction. When no prediction was used, the particle traces were compressed between 2.5 and 3.4%, depending on the zlib compression settings. Zeroth order prediction, where each particle was predicted to be the same as the previous particle, works much better, compressing the particle data by 27 to 34%. First order prediction uses the formula $p_i = x_i - 2x_{i-1} + x_{i-2}$, where p_i and x_i are the i^{th} preprocessed and original particle in the trace, respectively. First order prediction predicts that the particles travel in a straight line, and gets the best results: 42 to 43% compression. Finally, using second order prediction, where $p_i = x_i - 3x_{i-1} + 3x_{i-2} + x_{i-3}$, results in only 1 to 2% compression.

The different zlib settings had little effect. Changing the effort parameter from 6 (the default) to 9 (the maximum) did not change the compression ratio. Specifying that the data were filtered instead of using the default or Huffman-only compression increased the compression ratio by about 1%. However, Huffman-only compression took about half the time to compress the data, and sped the retrieval process by about 10% since it uses a much simpler algorithm.

The compression ratio was not the same for each time step, as shown in Figure 5. The initial time steps did not compress at all because the compression algorithm does not work well on very short sequences, and because the increased bookkeeping data used in compressed files is noticeable with very short traces. The compression ratio climbs to about 65% once the traces have about 100 particles. The compression ratio then slowly drops to about 30% as the trace length increases. A possible explanation for the smaller compression ratio is that more fully evolved traces are much more complex than short traces, which results in less compression. The overall compression ratio is 41% when using first order prediction and Huffman-only zlib compression.

The particle file format is slightly different with compressed particle traces. Instead of storing the number of particles in each trace in each file's header, we store the compressed size of each particle trace, which requires a 32-bit integer. The 16-bit particle count is moved to the beginning of each trace. Including the count allows the compression algorithm to store uncompressed traces if the traces cannot be compressed, and also simplifies allocating memory for the uncompressed traces.

Compressing the particle traces is currently done as a post-process to the particle computation, but it could easily be done during the computation. We estimate that compressing all of the traces using the cluster could be done in less than an hour (if the data were already in memory), which is less than 1% of the particle computation time.

6 Viewer

Once the particles traces have been computed, a separate application allows the traces to be viewed interactively. The viewer application has two components, which are separate programs: a workstation component, which runs on a workstation and handles user interaction and graphical display, and a server component, which runs on the file server and reads the requested traces from disk. The two components communicate using a custom protocol and a TCP connection. For increased performance during animation, multiple instances of the server component can be run on one or more file servers, providing interleaved access to time steps.

The viewer allows interactive manipulation of the viewpoint and the selected seedpoints. It also allows interactive manipulation of the current time step or allows time to move forward automatically, animating the particles. Seedpoint selection is done using a selection box widget, which allows arbitrary axis-aligned selections. The widget allows changing the size or position of the selection via a direct manipulation interface. An additional viewer feature is that the viewer allows other precomputed geometry, such as surfaces or cutting planes, to be displayed with the particle traces. The geometry can be static, time-varying, or rotating at a constant rate. The additional geometry allows the particles to be visualized in the context of the overall data set. Finally, the viewer allows the particles to be colored according to seedpoint position.

The viewer was written with attention to performance. When the particles are being animated, it overlaps the display with the retrieval of the next time step's particles. Particle retrieval is optimized by reading multiple adjacent particle traces in a single I/O request. Unlike the implementation by Bruckschen et al., our implementation does not read particle traces that are not needed in an interest to reduce the number of disk seeks because our measurements did not show any speedup.

The viewer operates differently depending on the interaction mode:

- **Time held constant and selection box modified.** In this mode, the viewer requests any necessary particle traces from the server. Retrieved traces are saved in a cache to avoid retrieving the same trace multiple times. The cache is flushed when the current time step is changed.
- **Time modified and selection box held constant.** The viewer requests the particle traces for each requested time step and displays them.
- **Time animated and selection box held constant.** When the particles are animating, the workstation component fetches the next time step's particles while the current particles are being displayed. The server component sends the requested particles, and, if time is available, fetches particles for the next time step, which is two time steps ahead of the time step being displayed.
- **Time animated and selection box modified.** This mode is similar to the previous mode except that the modified selection box causes the prefetching to not retrieve all of the necessary particles. The workstation component requests the missing particles via a second request. Unfortunately, this sec-

Statistic	Small Selection	Large Selection
Number of seedpoints	275	512
Average number of particles	268,116	577,911
Maximum number of particles	421,851	1,092,775
Average trace length	975	1,128

Average Frame Rate	Small	Large
Uncompressed particles, 1 server	5.4	3.4
Compressed particles, 1 server	7.0	4.3
Compressed particles, 2 servers	12.2	6.1

Table 2: Statistics and viewer performance for two selection sizes. The frame rates are given in frames per second.

ond request slightly decreases the response rate. This is partially alleviated by applying selection box changes to the next frame's prefetch request.

7 Equipment

The particle computation was done on a 49 node Beowulf cluster. Each node had two 1.67 Hz Athlon MP processors and 1 GB of memory, and had a Fast Ethernet network connection. The master node was similar but had 2 GB of memory and Gigabit Ethernet. The input data as well as the computed particles were stored on a pair of file servers. Each had two 3 GHz Xeon processors, 4 GB of memory, dual channel-bonded Gigabit Ethernet, and 21 250 GB data disks organized into three RAID 5 arrays (4.5 TB total storage each). The viewer timings were done on a workstation that had two 3 GHz Xeon processors, 4 GB memory, Gigabit Ethernet, and a nVidia Quadro FX 3000 graphics card. All the systems ran Red Hat Linux.

8 Performance

We measured the speed of the viewer application for a few different configurations so we could gauge its overall speed, and quantify the effect of using compression and different numbers of servers. All of the measurements used the turbopump data set and the same set of particles and seedpoints. Each run measured the time the viewer application took to retrieve a static selection of seedpoints for each of the time steps in the simulation. The performance figures are for runs that do not include any additional reference geometry in the viewer.

We measured three different configurations: one using uncompressed particle traces and one file server, a second using compressed particle traces and one server, and a third using compressed particles and two servers. Each configuration was run using two different particle selections: a small selection near the middle of the pump and a larger one near the pump's inlet. Table 2 gives some statistics about the selections and the resulting performance, and Figure 6 shows a few representative frames of the visualizations resulting from the two selection boxes.

The small selection box contained a $5 \times 7 \times 8$ array of seedpoints and had one row of missing seedpoints because the box was not entirely in the domain. The frame rate when using this selection, uncompressed particle traces, and one server was 5.4 frames per second. Using compressed traces increased the speed to 7 frames per second, a 30% speedup. Adding a second file server increased the frame rate by 75% to 12.2 frames per second. When using compressed particles, the frame rate is high enough to allow easy interaction.

The large selection box contained an $8 \times 8 \times 8$ array of seedpoints. This is the same number of seedpoints used in the measurements by Bruckschen et al. [2001]. However, our visualization had a larger average trace length, 1,128 particles, than the maximum trace length of 130 particles used in the earlier paper. (Of course, comparisons between the two implementations and data sets are difficult due to equipment differences.) The frame rate with the larger selection box and uncompressed particle traces was 3.4 frames per second, which is a bit low. Using compressed traces on one server increased the frame rate by 27%, to 4.3 frames per second. Animating the particles using compressed traces and two file servers ran at 6.1 frames per second, a 41% increase. This latter frame rate is reasonably fast; it is fast enough for the overall particle movement to be understood.

Overall, using particle trace compression results in a noticeable performance improvement in the viewer. In addition, using two servers instead of one adds a large performance boost since disk reads can be done in parallel.

The above measurements only give performance information for one mode of operation: when the particles are being animated with a static selection box. Other modes have different performances. Changing the selection while animating has somewhat lower performance since the particle prefetching does not retrieve all of the needed particles. Interactively modifying the time step is also a bit slower since prefetching cannot be used. However, changing the selection box when time is held constant is quite fast. It is fast because many of the particle traces displayed with the previous selection box can be reused with the new selection box since the boxes almost always overlap. The relative performances of the different modes can be seen in the included video.

We are still struggling with one performance problem: the frame rate is not constant. The time to read the particles from disk for some time steps is about twice the time for the others, even though the number of particles retrieved is about the same. The slow time steps are not the same between different runs, and occur roughly every 5 to 20 frames. We have found that running the server process at a very high priority level and locking the memory used sometimes decreases the problem. We have also found that the variable frame rate only occurs on our file servers; using a different system results in a reasonably constant frame rate. However, even with the variable frame rate, the particle viewer system is quite effective in showing the flow inside the turbopump data set.

9 Conclusions and Future Work

In this paper, we have shown that the method of visualizing particle flow by precomputing particle traces for later retrieval and display can be scaled to handle multi-terabyte data sets and more than a terabyte of particles. We have discussed the necessary modifications to Bruckschen et al.'s original algorithm that allow a data set with a curvilinear mesh to be used, and that allow the particle computation to be done using a PC cluster. In addition, a new compression technique allows the particle traces to be compressed by 41%, saving a significant amount of storage space, and also improves the interactive viewer performance by roughly 30%. Overall, we have demonstrated a visualization system that allows interactive particle visualization of a multi-terabyte CFD data set using PC hardware.

In the immediate future, we plan to improve the performance of both the particle computation, as described in Section 4.3, and the viewer application. We also hope to soon include adding the capability to record a small set of scalar values for each precomputed particle, such as pressure, velocity magnitude, and particle age, which will allow the scientist to gain additional understanding of the data set. In particular, recording each particle's age would allow the viewer to select particles according to age, which would allow limiting the maximum age of particles during the interaction. It would also allow the display of timelines, where particles are emitted every *n* time steps.

10 Acknowledgments

We could like to thank Cetin Kiris for providing the data set utilized in our experiments, and Tim Sandstrom for writing the program we modified to create the viewer application. This work was funded by the NASA Computing, Information, and Communications Technology (CICT) Program, partially via NASA contract DTTS59-99-D-00437/A61812D.

References

ABRAM, G., AND TREINISH, L. 1995. An extended data-flow architecture for a data analysis and visualization. In *Proceedings of Visualization '95*, IEEE Computer Society Press, 263–270.

AHRENS, J., BRISLAWN, K., MARTIN, K., GEVECI, B., LAW, C. C., AND PAPKA, M. 2001. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics & Applications 21*, 4 (July/August), 34–41.

BRUCKSCHEN, R., KUESTER, F., HAMMAN, B., AND JOY, K. L. 2001. Real-time out-of-core visualization of particle traces. In *Proceedings IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, IEEE Computer Society Press, 45–50.

CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. 1998. Interactive out-of-core isosurface extraction. *IEEE Visualization '98* (October), 167–174. ISBN 0-8186-9176-X.

CHIANG, Y.-J. 2003. Out-of-core isosurface extraction of time-varying fields over irregular grids. In *Proceedings of Visualization 2003*, IEEE Computer Society Press, 217–224.

COX, M. B., AND ELLSWORTH, D. A. 1997. Application-controlled demand paging for out-of-core visualization. In *Proceedings of IEEE Visualization '97*, IEEE Computer Society Press, R. Yagel and H. Hagen, Eds., 235–244.

DEUTSCH, P., AND GAILLY, J.-L., 1996. RFC 1950 - ZLIB compressed data format specification version 3.3.

ELLSWORTH, D. A., AND MORAN, P. J. 2003. Accelerating large data analysis by exploiting regularities. In *Proceedings of Visualization 2003*, IEEE Computer Society Press, 561–568.

ELLSWORTH, D. A. 2001. Accelerating demand paging for local and remote out-of-core visualization. Tech. Rep. NAS-01-004, NAS Division, NASA Ames Research Center, June.

HEERMAN, P. D. 1999. Production visualization for the ASCI one teraFLOPS machine. In *Proceedings of Visualization 1998*, IEEE Computer Society Press, 459–462.

LANE, D. 1994. UFAT: A particle tracer for time-dependent flow fields. In *Proceedings of Visualization '94*, IEEE Computer Society Press, 257–264.

MORAN, P. 2001. Field model: An object-oriented data model for fields. Tech. rep., National Aeronautics and Space Administration. NAS-01-005.

SAGAN, H. 1994. *Space-Filling Curves*. Springer-Verlag, New York.

UENG, S., SIKORSKI, C., AND MA, K. 1997. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (December), 370–380.

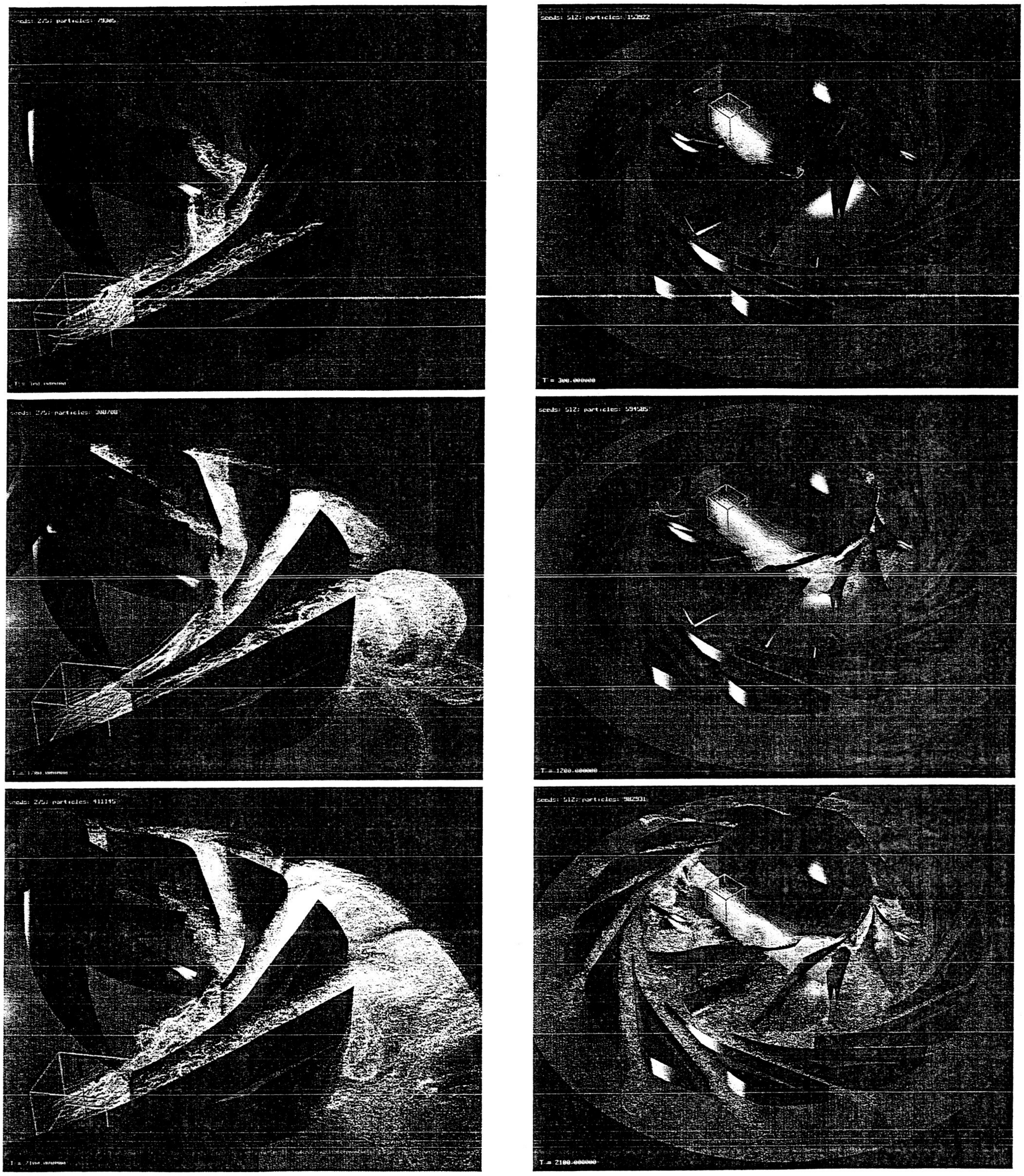

Figure 6: Two particle trace visualizations of the turbopump data set. The left column images show a visualization using a small selection box in the middle of the domain, and the right column images show a larger selection box near the inlet. The images in the three rows show the visualizations at time steps 300, 1200, and 2100 (respectively, top to bottom). The particles in the images on the right are colored according to their respective seedpoint's location within the selection box.