International Journal of Computational Science
1992-6669 (Print) 1992-6677 (Online) © Global Information Publisher
2007, Vol. 1, No. 4, 407-421

ParaViz: A Spatially Decomposed Parallel Visualization Algorithm Using Hierarchical Visibility Ordering

Cheng Zhang[1], Scott Callaghan[2], Thomas Jordan[2], Rajiv K. Kalia[1],

Aiichiro Nakano[1*], Priya Vashishta[1]

[1] Collaboratory for Advanced Computing and Simulations, Department of Computer Science, Department of Physics & Astronomy, Department of Chemical Engineering & Materials Science,
University of Southern California, Los Angeles, CA 90089-0242, USA
{chengz,rkalia,anakano,priyav}@usc.edu
[2] Southern California Earthquake Center, University of Southern California,
Los Angeles, CA 90089-0742, USA
{scottcal,tjordan}@usc.edu

Abstract. A scalable parallel visualization algorithm has been designed to visualize large datasets that are spatially decomposed onto processors of a massively parallel computer. The ParaViz algorithm, which is based on hybrid sort-first/sort-last parallel visualization, employs distributed visibility ordering to implement a scalable hierarchical depth buffer. A visibility rank is computed for each processor depending on its relative position from the viewpoint. After each processor rasterizes its own primitives, individually rendered sub-images are hierarchically reduced to a final image using their visibility ranks. This technique allows on-the-fly visualization of parallel simulation data without data migration. The algorithm has been tested on 8–1,024 processors for molecular-dynamics simulation data. In a weak-scaling test consisting of 64,000 spherical objects (atoms) per processor, the image integration time of the ParaViz algorithm is 40–50% less than that of the conventional global Z-buffer approach. A strong-scaling test involving 16,777,216 atoms achieves a parallel efficiency of 0.98 on 1,024 processors.

Keywords: parallel visualization, distributed visibility ordering, sort-first and sort-last schemes, Z-buffer, hierarchical communication scheme, molecular dynamics simulation.

* Corresponding Author. Email: anakano@usc.edu.

1 Introduction

Computer visualization is an important tool for data analysis and presentation in computational sciences, where it is used to effectively extract and convey information contained in large datasets. However, continued growth in computing power has led to ever-larger scientific datasets that conventional single-processor visualization cannot support. Consequently, various parallel-rendering schemes have been proposed to decompose the visualization load onto multiple processors [1]. Software solutions utilize task parallelism [2] or data parallelism, which include sort-first, sort-middle, and sort-last schemes for spatial distribution of data [3]. Sort-first schemes redistribute raw primitives during geometry processing; sort-middle schemes redistribute screen-space primitives between geometry processing and rasterization; and sort-last solutions redistribute pixels during rasterization. Corresponding hardware solutions utilize distributed- or shared-memory systems to replace single graphics workstations. Open-source parallel visualization toolkits (such as VTK [4, 5]) and parallel scientific visualization applications (such as ParaView [6] and DataView [7]) are also available. VTK [4, 5] is a multi-platform parallel class library that uses a higher level of abstraction than OpenGL to render complex systems on cluster computers. ParaView [6] is a versatile parallel renderer and viewer implemented with the message passing interface (MPI) for various types of distributed systems. DataView [7] performs interactive rendering of large-scale scientific data using a client-server model implemented on a PC cluster.

A number of parallel rendering algorithms have been designed to utilize clusters containing graphics processing units (GPUs). Samanta *et al.* [8] proposed a hybrid sort-first and sort-last approach to render polygons on GPUs, which assigns each processor a partition of objects and a partition of image space. Liang *et al.* [9] used dynamic data distribution on a Sepia cluster to support interactive investigation of fine structures in large particle datasets. Magallón *et al.* [10] proposed a *commodity off the shelf* (COTS) sort-last solution for inexpensive distributed volume rendering. This approach was extended by Strenger *et al.* [11] to include wavelet compression and depth-sorted blending based on volume brick footprints to minimize blending operations. However, as the majority of existing multi-teraflops and future petaflops computers lack GPUs in the computing nodes, it is essential to use their central processing units (CPUs) for both computing and visualization. Mitra and Chiueh [12] have explored this idea by implementing a parallel version of the Mesa graphics library to handle communication and image composition on large computational clusters.

All parallel visualization algorithms require a data decomposition/sub-image compositing scheme [1]. For example, the image-space decomposition scheme distributes primitives using their screen coordinates and directly patches sub-images into a final image. Alternatively, object-space decomposition distributes primitives using their spatial world coordinates, and thus it requires depth comparison of sub-images for compositing. For large-scale simulation data, which are often spatially decomposed with a unique region assigned to each processor, object-space decomposition is more efficient, since sub-image sorting is much less time consuming than redistributing large datasets. Parallel visualization exploiting spatial decomposition has been implemented in

GeoFEM [13], a large-scale finite element analysis platform that uses the Earth Simulator to visualize volumetric data, and in SLIC [14], a sort-last algorithm that classifies rendered pixels and creates a compositing schedule for each processor.

Multiple techniques have been used for performing depth comparisons during integration. Mitra and Chiueh [12], Samanta *et al*. [8], and Li *et al*. [15] used a global Z-buffer, while others such as Chen *et al.* [13], Bethel *et al.* [16], and Strengert *et al.* [11] used a similar approach with larger granularity, assigning depth buffer values to groups of primitives. These global Z-buffer implementations can become a critical bottleneck when visualizing large systems on high-end parallel computers, *e.g*., a quantum-mechanical simulation involving 1.04 trillion grid points or a molecular dynamics (MD) simulation of 134 billion atoms on 131,072 BlueGene/L processors [17]. In this paper, we propose an algorithm named ParaViz, which is scalable on high-end parallel computers as well as on future multi-petaflops computing platforms. The algorithm requires no migration of data, which would be extremely expensive on various emerging architectures such as cell-processor systems. Furthermore, the algorithm enables concurrent visualization and computation with a hierarchical depth buffer based on visibility ordering and a hybrid sort-first/sort-last scheme.

The main contributions of this paper are an efficient algorithm for large-scale parallel visualization of spatially decomposed datasets, and its simple implementation that is faster than the conventional Z-buffer implementation and exhibits excellent scalability. The rest of the paper is organized as follows. Section 2 describes the ParaViz algorithm, and numerical results are presented in Section 3. Conclusions and future directions are contained in Section 4.

2 ParaViz Algorithm

2.1 Basic Methodology

The fundamental challenge in visualizing a spatially decomposed dataset on a parallel computer is the hidden-surface problem, that is, the task to determine which primitives are visible and which are obscured. The traditional hidden-surface algorithms, summarized by Sutherland *et al.* [18] decades ago, have proved efficient for moderate-sized data stored in a single computer. However, the standard solution of maintaining a single depth buffer for the entire system, though simple, does not scale well to large numbers of processors. Additionally, it requires the depth buffer size to increase with the system size in order to maintain adequate precision.

Our approach exploits the features of spatial decomposition and uses a visibility ordering of processors instead of a global Z-buffer, allowing for substantial savings in integration time. Visibility ordering of objects was pioneered by Schumacker *et al.* and reviewed in [18] and Newell *et al.* [19], and has been employed and extended in a number of visualization algorithms [20-23]. By definition, visibility ordering is a topological ordering of objects with planar interfaces for correct hidden-surface elimination. The ordering sequence depends on the viewpoint position. We extend

this concept to implement visibility ordering of parallel processors in a spatially decomposed system. Assuming two adjacent processors with a planar interface (common in spatial decomposition for parallel computing), one processor is "in front", *i.e.*, on the same side of the interface with the viewpoint. Primitives contained in the front processor may occlude those in the other, but not vice versa. If the viewpoint happens to be on the interface, either processor can be defined as frontal. By definition, any total ordering assigning a smaller visibility rank to the front processor in each adjacent pair is a visibility ordering [21], where a processor may occlude one with a larger rank, but not one with a smaller or equal rank (see the theorem below). With a processor visibility ordering, one can easily sort sub-images generated by individual processors in depth. However, in practice, it is costly and impractical to compute visibility ordering through pair-wise comparisons. Instead, we exploit features of common spatial decomposition and have each processor compute its own visibility order independently.

Theorem:

In a visibility ordering, a processor cannot occlude one with a smaller or equal visibility rank.

Proof:

Let P1 and P2 be two adjacent processors with a planar interface in a visibility ordering, where P1 is on the same side with the viewpoint and has a smaller visibility rank than P2. Suppose that P2 occludes P1, which implies a line of sight starting from the viewpoint and passing through P2 into P1. Therefore, the line must pass through P2 before it crosses the interface. This contradicts that P2 is on the other side of the interface from the viewpoint; therefore, P2 cannot occlude P1. Due to transitive nature of both occlusion relation and inequality relation in a total ordering, the occlusion relation applies to any pair of processors with visibility rank inequality and thus it completes the proof. □

After computing its own visibility rank, each processor proceeds to render primitives in its assigned region using universal visualization parameters (such as view angle and frustum planes). The spatial localization of primitives makes ParaViz partially sort-first. Subsequently, the individually rendered sub-images are assigned the visibility rank of their native processors and integrated in a hierarchical fashion. At each step, the opaque pixels of the front sub-image (lower visibility rank) are superimposed on the back sub-image to generate the new sub-image, whose visibility rank is set to the lower of the two. This process is repeated recursively until all sub-images are reduced to a final image that contains the entire rendered system. This makes ParaViz partially sort-last, since pixels must be sorted in image composition. Overall, ParaViz is a hybrid sort-first and sort-last approach, allowing us to capitalize on spatial decomposition without maintaining a costly global Z-buffer.

The present algorithm has a lower time complexity than the traditional Z-buffer approach. In order to render N primitives on P processors, each processor on average renders N/P primitives. Since the traditional Z-buffer approach needs $\log N$ bits to store the depth information correctly for the global system, the computational cost for each processor to sort the primitives is $O((N/P)\log N)$. The ensuing hierarchical communication consists of $\log P$ levels, and $O(\log N)$ bits must be communicated at each level. On the other hand, ParaViz only requires $\log(N/P)$ bits to maintain a local

Z-buffer. Its computational cost for sorting primitives is $O((N/P)\log(N/P))$. In image integration, only a single value of depth information, the visibility rank, is communicated, reducing communication cost to $O(\log P)$. The resulting time complexity of the ParaViz algorithm is $O((N/P)\log(N/P)+\log P)$, which is significantly lower than the $O((N/P)\log N+\log P\log N)$ complexity of the Z-buffer approach.

2.2 Implementation

Our implementation of ParaViz is written in the C language, with the message passing interface (MPI) library [24] for interprocessor communication. It also uses the Mesa [25] implementation of OpenGL for software rendering, which enables it to run on clusters that are not equipped with GPUs. Although ParaViz works for any system decomposed into a 3D parallelepiped mesh, the implementation presented below is for a regular orthogonal 3D mesh — a common decomposition scheme in parallel computing.

Each processor first computes its visibility rank as shown in Table 1.

Table 1. Visibility-rank algorithm

Algorithm visibility_rank
Input:
local processor position indices z_i ($i \in \{1, 2, 3\}$ for the x, y, and z directions)
viewpoint coordinates X_i
cell dimensions L_i
Output
visibility rank R
Steps:
for $i = 1$ to 3
$D_i \leftarrow | z_i - \lfloor X_i / L_i \rfloor |$
end for
$R \leftarrow \sum_{i=1}^{3} D_i$

In the general case with a parallelepiped mesh, each processor calculates the global processor coordinates (integers) of the viewpoint, using the viewpoint location and metadata regarding the spatial decomposition (see Fig. 1). If the viewpoint lies outside the system, one or more indices will be negative. The rank is calculated by summing the absolute differences between the viewpoint's integer coordinates and the processor's integer coordinates in each dimension. Because such an ordering is a topological ordering (thus a total ordering) and consistent with the occlusion relation for neighboring pairs, it is a visibility ordering. In our implementation, processors of lower visibility ranks are in front of those of higher ranks. While multiple processors can have the same visibility rank, they are guaranteed to have no overlap from the viewer's perspective and therefore no conflict when integrating.

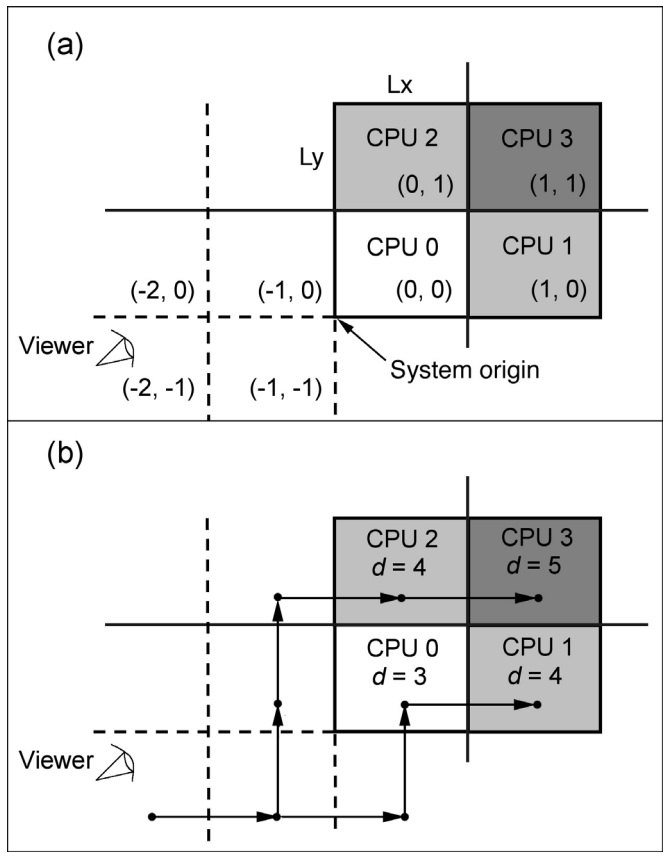

Fig. 1. (a) A 2-dimensional illustration, in which each processor determines the integer coordinates (bracketed) of the viewer's position using the viewer's location and the spatial decomposition parameters. Its own integer coordinates (*i.e.*, global indices; bracketed) are already known from the system setup (b) In each dimension, the integer distance between the viewer and the processor is calculated and summed to obtain the processor's visibility rank (variable *d* in the figure, the total integer distance from the viewpoint). In this example lower visibility ranks correspond to front processors (filled with lighter shade)

Subsequently, each processor renders its own primitives into a memory buffer of the size with the final image. We use back-face culling to speed up the rendering process, and future versions of ParaViz will include probabilistic occlusion culling and other advanced optimizations [26].

After the individual primitives are rendered, the buffers from each processor are integrated using a hierarchical scheme. Reduction is done first in the *x* dimension, then *y* and *z*, compressing the 3D system first onto a plane of processors, then a rcolumn, and finally a single processor that produces the final image. In each dimension, the reduction process follows a hierarchical scheme illustrated in Fig. 2, which works for an arbitrary number of processors in each dimension (see Table 2). For every communicating pair, the lower indexed processor merges two sub-images,

using visibility ranks for pixel sorting (see Table 3). To reduce integration time, we identify the first and last non-empty pixels in each sub-image buffer (a 1D array in memory). Only those pixels within the range are sent and integrated, avoiding communication and comparison of empty margin pixels.

Table 2. Image reduction algorithm

Algorithm image_reduction
Input:
local processor position indices z_i ($i \in \{1, 2, 3\}$ for the x, y, and z directions)
number of processors in ith dimension P_i ($i \in \{1, 2, 3\}$)
sub-images and visibility ranks from individual processors
Output:
globally-reduced total image
Steps:
$done \leftarrow 0$
for $i = 1$ to 3
$n \leftarrow 1$
while $2^n <= P_i$ **do**
if $done = 0$
if the nth bit of z_i is 0
$q_i \leftarrow z_i + 2^{n-1}$
if $q_i < P_i$
receive sub-image and visibility rank from q_i
integrate the two image buffers
end if
else
$q_i \leftarrow z_i - 2^{n-1}$
send its sub-image buffer and visibility rank to q_i
$done \leftarrow 1$
end if
end if
$n \leftarrow n + 1$
end while
end for

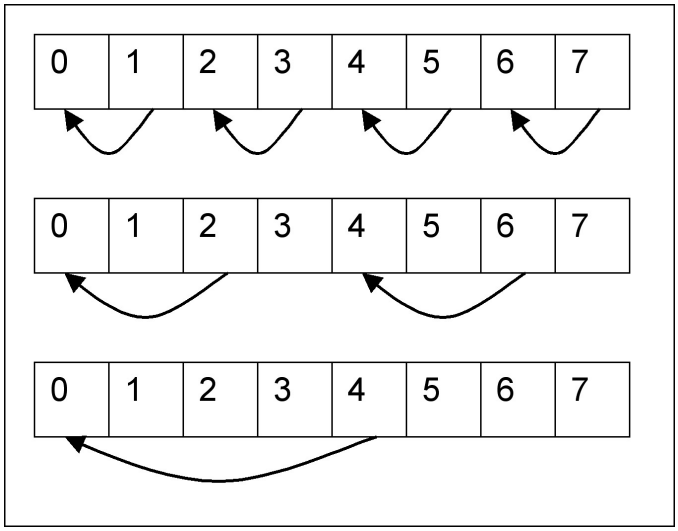

Fig. 2. Illustration of the hierarchical communication scheme for eight processors in 1D. For each of $n = \log P$ steps, processors with ID mod $2^n = 0$ receive the sub-image from the processor whose ID differs only in the nth bit. The communication scheme also works for an arbitrary number of processors in each dimension

Table 3. Buffer integration algorithm

Algorithm buffer_integration
Input:
visibility rank of local buffer R_{local}
visibility rank of received buffer $R_{received}$
Output:
integrated image
Steps:
determine the union of pixel bounding ranges of two sub-images
if $R_{local} < R_{received}$
for all pixels in the union range
if the pixel in the local sub-image is empty
replace it by the corresponding pixel in the received sub-image
end if
end for
else
for all pixels in the union range
if the pixel in the received sub-image is non-empty
replace the corresponding pixel in the local sub-image
end if
end for
end if
$R_{\text{local}} \leftarrow \min(R_{\text{local}}, R_{\text{received}})$

Upon completion of the integration stage, the combined image buffer resides on the head processor (processor with index 0 in all three dimensions). This image could then be written to disk or transferred through a network to display on a client machine.

3 Results

We test the ParaViz algorithm on a Linux cluster consisting of 256 nodes, each with two dual-core Opteron 2GHz processors and 4GB RAM (totaling 1,024 cores and 1TB RAM). We compare the performance of ParaViz to that of a conventional global Z-buffer implementation for a molecular dynamics (MD) simulation dataset consisting of millions of atoms. Both ParaViz and the global Z-buffer implementation use the same software rendering code, which renders each atom as a sphere.

We first perform weak-scaling tests, in which the number of spherical objects (atoms) per processor is fixed as 64,000 while varying the number of processors P from 8 to 1,024. Figure 3 shows that ParaViz is 40–50 percent faster than the global Z-buffer implementation in sub-image integration. The dips in the integration time curve at 128 and 1,024 processors are due to viewpoint reposition for a fixed horizontal viewing angle. When the system expands in the horizontal direction, the viewpoint is moved away from the system. This causes fewer pixels to be rendered by each processor and less data sent and processed during integration. To keep the system as close to cubic as possible, the number of processors is doubled each step in a cyclic order of width, depth and then height. From 64 to 128 (and from 512 to 1,024) the number of processor is doubled in the horizontal direction (width), therefore the viewpoint is repositioned and integration time decreases correspondingly. Expansion in other dimensions does not result in a viewpoint shift; thus the integration time increases due to more communication.

Since ParaViz uses the same rendering code as the Z-buffer implementation, they have similar rendering time, which can be reduced using advanced techniques such as occlusion culling. However, this paper focuses on reducing the integration time—the bottleneck to achieve scalable visualization on a large number of processors. The results presented in this section show that the ParaViz algorithm is significantly faster than the global Z-buffer implementation in integration time for intermediate to large numbers of processors. For small numbers of processors, on the other hand, the advantage of ParaViz is less significant. The speed difference is expected to increase for larger numbers of processors, *e.g.*, on the 131,072-processor IBM BlueGene/L mentioned previously. This can be deduced from the two algorithms' different parallel computational complexities shown in the previous section. In addition, on each processor, ParaViz uses fewer bits for pixel depth than the global Z-buffer implementation, because it only maintains depth metadata within individual subspaces. This allows for improved depth resolution with the same memory resources, a negligible advantage for small systems but significant for systems with large numbers of pixels. For both implementations, the integration time increases with the number of processors due to increased communication. However, the rendering time decreases, because each processor renders fewer pixels as the number of processors increases.

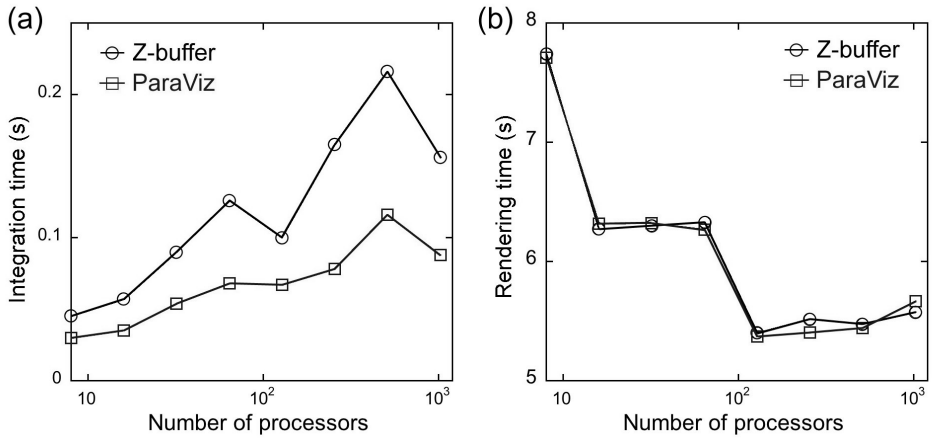

Fig. 3. The integration (a) and rendering (b) times of the global Z-buffer approach and ParaViz on 8–1,024 processors for 64,000 atoms per processor. The integration time of ParaViz is consistently shorter by 40–50%

Next, we perform strong-scaling tests, in which ParaViz is used to visualize a MD simulation dataset of 16,777,216 atoms using 8 to 1,024 processors (see Fig. 4). As the number of processors

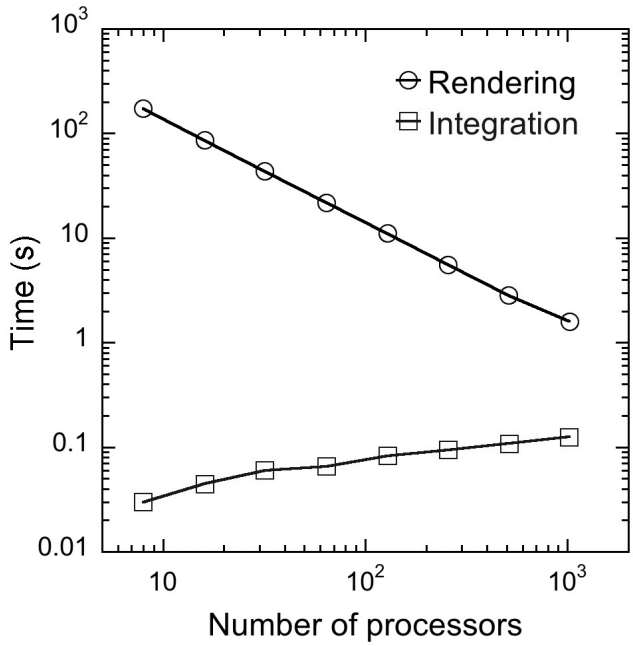

Fig. 4. Strong-scaling results for the rendering (circles) and integration (squares) times of ParaViz for an MD simulation data consisting of N = 16,777,216 atoms on P = 8–1,024 processors. Rendering time is roughly proportional to N/P, and the integration time increases moderately with P

increases, the rendering time decreases in proportion to the number of atoms being rendered per processor ($\propto N/P$). Integration time, in contrast, slowly increases proportionally to log$P$ due to increased communication. Figure 4 thus demonstrates the tradeoff between decreasing rendering time and increasing integration time as a function of the number of processors. Since the rendering time is dominant, the overall effect of parallelism is a monotonically decreasing computing time up to at least 1,024 processors.

To quantify the algorithm's efficiency, the speed is defined as the inverse of visualization wall clock time. The speedup is the speed on P processors divided by the speed on one processor, and normalized such that the speedup on eight processors is 8. We also define the parallel efficiency as the speedup divided by the number of processors. Figure 5 shows the strong-scaling speedup of ParaViz for the same dataset as in Fig. 4 (i.e., 16,777,216 atoms on 8 to 1,024 processors). The strong-scaling parallel efficiency obtained from Fig. 5 is 0.98 on 1,024 processors, which is close to the ideal value of 1.0, signifying the excellent scalability of our algorithm.

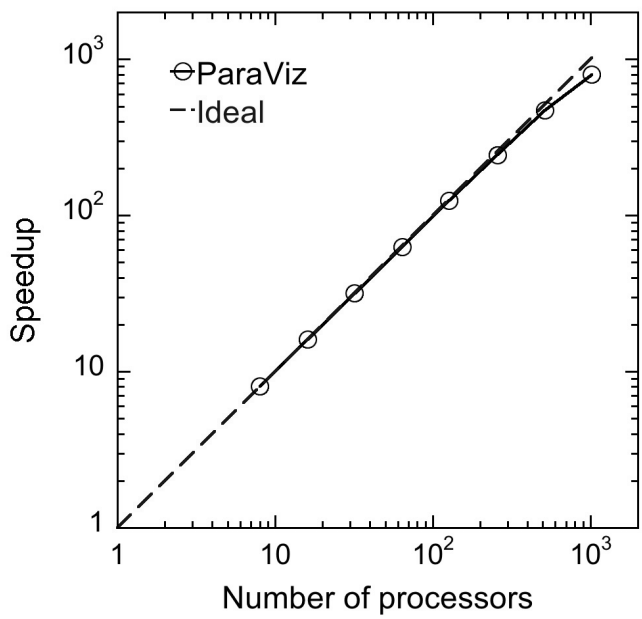

Fig. 5. The strong-scaling speedup of the ParaViz algorithm (circles) compared to the ideal speedup (dashed line) for an MD dataset consisting of 16,777,216 atoms on 8–1,024 processors

ParaViz has been used extensively to visualize large MD simulations of material processes such as hypervelocity impact [27], fracture [28], and indentation [29]. A nanocrystalline alumina (n-Al_2O_3) system containing 40 million atoms is visualized using ParaViz on 1,024 processors (Figs. 6a and 6b). An α-alumina single crystal of 540 million atoms under hypervelocity impact is visualized on as many processors (Fig. 6c).

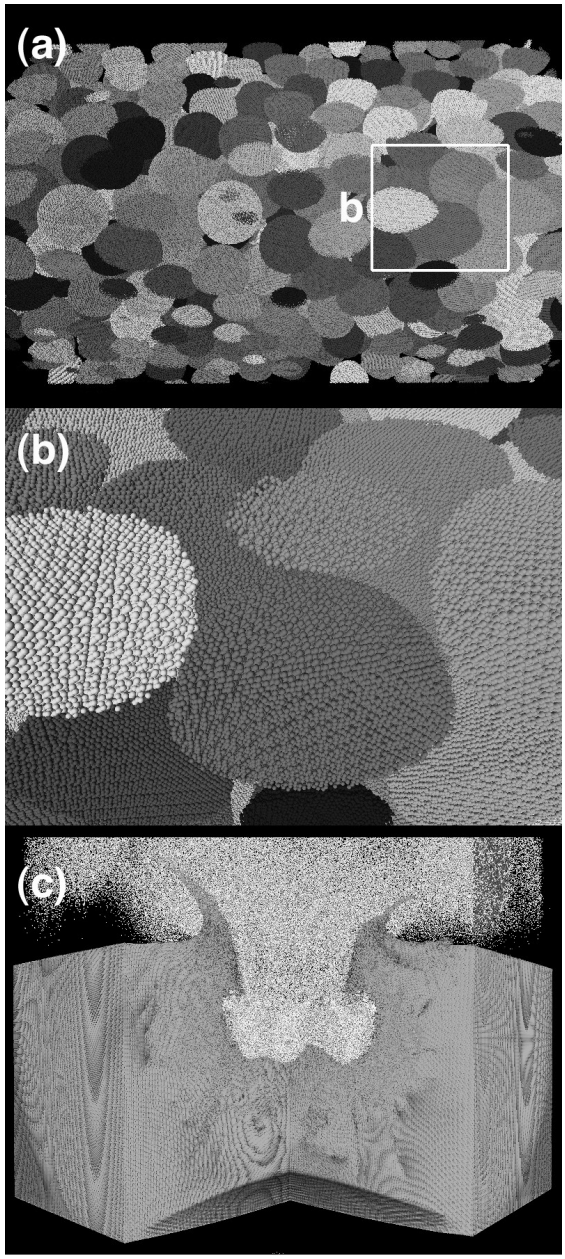

Fig. 6. (a) Nanophase ceramic material consisting of 512 crystalline alumina (Al_2O_3) nano-particles (totaling 40 million atoms) visualized with ParaViz, where each nano-particle is distinguished by color. The system is viewed along a diagonal of the *y*-*z* plane (the *x* axis lies in the horizontal direction)
(b) A close-up image of the boxed region in (a)
(c) An alumina substrate consisting of 540 million atoms under hypervelocity impact at 18km/s. The atoms are color-coded based on pressure values

4 Conclusion

For massive datasets generated by scientific computing, visualization has become highly challenging. To utilize high performance computing clusters for real-time rendering of spatially decomposed computational data, we have designed and implemented ParaViz, a hybrid sort-first/sort-last visualization algorithm with a scalable hierarchical depth buffer. Using viewpoint-dependent visibility ordering of processors to sort sub-images during integrated rasterization, we gain a speedup of 40–50% over the traditional global Z-buffer implementation. ParaViz also scales well with large numbers of processors and supports real-time visualization. Although our work has been focused on visualizing molecular-dynamics simulation data, the proposed scheme can be applied in other fields of research with intensive visualization requirements.

In our current implementation using Mesa, the majority of execution time is spent in the individual rendering pipeline. By applying advanced rendering techniques such as occlusion culling, we expect to substantially reduce the execution time. As the number of processors grows, the load imbalance increases since the processors closer to the viewer span a larger view angle than those farther away. Thus a load-balancing scheme will reduce the idle time of processors with less intensive rendering demands. We are also extending ParaViz to handle parallel volume rendering.

Acknowledgements

This work was partially supported by ARO—MURI, DOE—SciDAC, DTRA, and NSF. Numerical tests were performed using the 5,384-processor Linux cluster at the Research Computing Facility and the 2,048-processor Linux cluster at the Collaboratory for Advanced Computing and Simulations of the University of Southern California.

References

1. Crockett, T.W.: An introduction to parallel rendering. *Parallel Computing* 23 (1997) 819-843
2. Ahrens, J., Brislawn, K., Martin, K., Geveci, B., Law, C.C., Papka, M.: Large-scale data visualization using parallel data streaming. *Ieee Computer Graphics and Applications* 21 (2001) 34-41
3. Molnar, S., Cox, M., Ellsworth, D., Fuchs, H.: A Sorting Classification of Parallel Rendering. *Ieee Computer Graphics and Applications* 14 (1994) 23-32
4. Ahrens, J., Law, C., Schroeder, W., Martin, K., Papka, M.: A parallel approach for efficiently visualizing extremely large, time-varying datasets. *Technical Report*. Los Alamos National Laboratory (2000)
5. Schroeder, W.J., Martin, K., Lorensen, W.: *The Visualization Toolkit*. Kitware, Inc., (2003)
6. Law, C.C., Henderson, A., Ahrens, J.: An application architecture for large data visualization: a case study. *IEEE 2001 symposium on parallel and large-data visualization and graphics*. San Diego, California, United States (2001) 125-159

7. Slottow, J., Shahriari, A., Stein, M., Chen, X., Thomas, C., Ender, P.B.: Instrumenting and tuning data-View - a networked application for navigating through large scientific datasets. *Software-Practice & Experience* 32 (2002) 165-190
8. Samanta, R., Funkhouser, T., Li, K., Singh, J.P.: Hybrid sort-first and sort-last parallel rendering with a cluster of PCs. *ACM SIGGRAPH/EUROGRAPHICS workshop on graphics hardware*. Interlaken, Switzerland (2000) 97-108
9. Liang, K., Monger, P., Couchman, H.: Interactive parallel visualization of large particle datasets. *Parallel Computing* 31 (2005) 243-260
10. Magallón, M., Hopf, M., Ertl, T.: Parallel Volume Rendering Using PC Graphics Hardware. *Ninth Pacific Conference on Computer Graphics and Applications* (2001) 384-389
11. Strengert, M., Magallón, M., Weiskopf, D., Guthe, S., Ertl, T.: Large volume visualization of compressed time-dependent datasets on GPU clusters. *Parallel Computing* 31 (2005) 205-219
12. Mitra, T., Chiueh, T.: Implementation and Evaluation of the Parallel Mesa Library. *International Conference on Parallel and Distributed Systems*. Taiwan (1998) 84-91
13. Chen, L., Fujishiro, I., Nakajima, K.: Optimizing parallel performance of unstructured volume rendering for the Earth Simulator. *Parallel Computing* 29 (2003) 355-371
14. Stompel, A., Ma, K., Lum, E., Ahrens, J., Patchett, J.: SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering. *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003) 6-13
15. Li, P.P., Duquette, W.H., Curkendall, D.W.: RIVA: A versatile parallel rendering system for interactive scientific visualization. *Ieee Transactions on Visualization and Computer Graphics* 2 (1996) 186-201
16. Bethel, E.W., Greg, H., Brian, P., Brederson, J.D.: Sort-First, Distributed Memory Parallel Visualization and Rendering. *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003) Pages
17. Nakano, A., Kalia, R., Nomura, K., Sharma, A., Vashishta, P., Shimojo, F., van Duin, A.C.T., Goddard, W.A., III, Biswas, R., Srivastava, D.: De Novo Ultrascale Atomistic Simulations on High-end Parallel Supercomputers. *International Journal of High Performance Computing Applications* (2006)
18. Sutherland, I.E., Sproull, R.F., Schumacker, R.A.: A Characterization of Ten Hidden-Surface Algorithms. *ACM Comput. Surv.* 6 (1974) 1-55
19. Newell, M.E., Newell, R.G., Sancha, T.L.: A solution to the hidden surface problem. *Proceedings of the ACM annual conference* 1. Boston, Massachusetts, United States (1972) 443-450
20. Frieder, G., Gordon, D., Reynolds, R.A.: Back-to-Front Display of Voxel-Based Objects. *Ieee Computer Graphics and Applications* 5 (1985) 52-60
21. Williams, P.L.: Visibility Ordering Meshed Polyhedra. *Acm Transactions on Graphics* 11 (1992) 103-126
22. Callahan, S.P., Ikits, M., Comba, J.L.D., Silva, C.T.: Hardware-assisted visibility sorting for unstructured volume rendering. *Ieee Transactions on Visualization and Computer Graphics* 11 (2005) 285-295
23. Cook, R., Max, N., Silva, C.T., Williams, P.L.: Image-space visibility ordering for cell projection volume rendering of unstructured data. *Ieee Transactions on Visualization and Computer Graphics* 10 (2004) 695-707

24. MPI homepage: http://www.mpi.org
25. Mesa homepage: http://mesa3d.sourceforge.net/
26. Sharma, A., Nakano, A., Kalia, R.K., Vashishta, P., Kodiyalam, S., Miller, P., Zhao, W., Liu, X.L., Campbell, T.J., Haas, A.: Immersive and interactive exploration of billion-atom systems. *Presence-Teleoperators and Virtual Environments* 12 (2003) 85-95
27. Branicio, P.S., Kalia, R.K., Nakano, A., Vashishta, P.: Shock-induced structural phase transition, plasticity, and brittle cracks in aluminum nitride ceramic. *Physical Review Letters* 96 (2006) 065502
28. Lu, Z., Nomura, K., Sharma, A., Wang, W.Q., Zhang, C., Nakano, A., Kalia, R., Vashishta, P., Bouchaud, E., Rountree, C.: Dynamics of wing cracks and nanoscale damage in glass. *Physical Review Letters* 95 (2005) 135501
29. Szlufarska, I., Nakano, A., Vashishta, P.: A crossover in the mechanical response of nanocrystalline ceramics. *Science* 309 (2005) 911-914