Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers

Ulrich Neumann

neumann@cs.unc.edu
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

This work examines the network performance of mesh-connected multicomputers applied to parallel volume rendering algorithms. This issue has not been addressed in papers describing particular parallel implementations, but is pertinent to anyone designing or implementing parallel rendering algorithms. Parallel volume rendering algorithms fall into two main classes - image and object partitions. Communication requirements for algorithms in these classes are analyzed. Network performance for these algorithms is estimated by using an existing model of mesh network behavior. The performance estimates are verified by tests on the Touchstone Delta. The results indicate that, for a fixed screen size, the performance of 2D mesh networks scales very well when used with object partition algorithms - the time required for communication actually decreases as the data and system sizes increase. A Touchstone Delta implementation of an object partition algorithm is briefly described to illustrate the algorithm's low communication requirements.

1. Introduction

The computational expense of volume rendering has motivated the development of parallel implementations on multicomputers. Through parallelism, higher rendering rates may be achieved which provide more natural viewing control and enhanced comprehension of three dimensional structure. Many parallel implementations have been reported, but no framework has been established to allow comparisons of their relative merits independent of their host hardware. This work builds on a taxonomy of parallel volume rendering algorithms suitable for MIMD multicomputers with distributed memory and a communication network [Neumann93]. Three classes of parallel algorithms are considered in a system-independent fashion that reveals their inherent communication requirements. To illustrate the usefulness of these results, the algorithm communication requirements are applied to mesh-connected multicomputers and the predicted performance levels are compared.

It is important to clarify the distinction between a *parallel volume rendering algorithm* and a volume rendering method like ray casting [Levoy88] or splatting [Westover89]. A parallel algorithm describes how data and computation is distributed among the resources of a system. In such a description, the rendering method is not an issue and may be unspecified. For example, a simple parallel algorithm for a system with n nodes divides the screen into n regions and assigns each node a separate region to render. This parallel algorithm does not specify what rendering method is used by each node to render its region. By considering parallel algorithms and rendering methods independently, the performance ramifications of each issue may be considered separately and more clearly.

The choice of parallel algorithm has a major impact on the communication requirement between nodes. Unless all nodes have a local copy of the data, or viewing positions are severely restricted, a parallel volume rendering algorithm intrinsically requires data to be communicated between compute nodes. Replication of data at each node is prohibitively expensive for large numbers of nodes and restricting the viewing positions limits the ability to explore the data; so, in the general case, communication must occur. Communication between nodes in a parallel system is time consuming and may degrade performance significantly, so understanding the inherent requirements of an algorithm is important when considering its use. The peak amount of data communicated per frame is independent of the rendering method. The choice of rendering method may reduce the actual requirement; for example, nodes that render by ray casting may adaptiveiy terminate rays and therefore not access portions of the data that would otherwise be needed. Such efficiencies are data dependent but often significant [Levoy90] [Danskin*92]. In this analysis, the peak communication requirement is derived as an upper bound with the understanding that rendering efficiencies may reduce this by some factor. The behavior of the communication requirement as data, image, and system sizes vary is also important and clearly revealed.

Communication between nodes in multicomputers is frequently through two and three-dimensional mesh connected networks [Dash] [Delta] [JMach] [Mosaic] [Paragon]. The performance of these communication networks with parallel volume rendering algorithms is the focus of this work. Of particular interest is the analysis of network performance as the volume data and system size increases. This becomes increasingly important as technology advances make more computing nodes practical in parallel systems. Using simulation results and the analysis of the communication requirements for three classes of algorithms, an existing model of mesh network behavior is used to produce expressions for the time consumed by communication. These results predict that, *for a fixed image size, the class of object partition algorithms requires decreasing communication time as the data and system sizes grow*. This scaling behavior of object partition algorithms on mesh networks is verified by experimental tests run on the Touchstone Delta.

2. Parallel Algorithms

The two main classes of parallel algorithms are *image partitions* and *object partitions*. In an image partition, nodes are assigned regions of screen-space to render. Data must be communicated to the nodes based on the view transformation. In an object partition, each node renders a color and opacity image of its local data subset. The local images are then communicated to facilitate their composite into the final image. The member algorithms in each class differ in the shapes of the data and image subsets, the subset's static or dynamic nature over time, and the spatial relationship of the subsets to each other [Neumann93]. This study is limited to regular lattices of data and the rendering algorithms associated with them. Three classes of useful algorithms arise because image partitions divide into two different subclasses - one with a static data distribution, and one with a dynamic data distribution.

0-8186-4920-8/93 $3.00

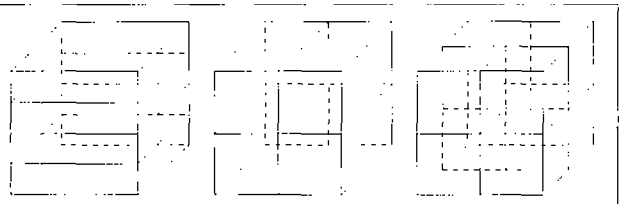

Fig. 1. - Data subset shapes

In any parallel algorithm, volume data subsets may be distributed among nodes in three shapes: *slabs*, *shafts*, and *blocks* (Fig. 1). When data is communicated, the subset size is the granularity of the transfer. To allow reasonably small transfers there may be more data subsets than nodes, so a node may store multiple subsets in its local memory. If these multiple subsets are spatially adjacent, (e.g., multiple slices forming a slab) they are classified as *contiguous*. Any other spatial arrangement is classified as *interleaved*. If the distribution of subsets varies between frames, the distribution is *dynamic*. An unchanging distribution is *static*.

The communication of data or images that arises from any algorithm is referred to as *redistribution*. The redistribution size for each algorithm class and the cost for transmission through the network is formulated in section 4. It bears repeating that no rendering method is implied in this analysis - the peak redistribution size and cost are not impacted by the choice of rendering methods.

3. Network Model

Current generation mesh and toroidal networks employ *virtual cut-through*, *oblivious*, *wormhole* routing techniques [Delta] [Paragon]. This terminology and the characteristics of these networks are reviewed below.

Virtual cut-through refers to the way messages pass through intermediate network nodes between the source and destination nodes. Routing logic on intermediate nodes detects the message destination encoded into the message header, and forwards the message to a neighboring node without interrupting the intermediate node's processor.

A network that has fixed, deterministic message routing paths for any source-destination node pair, is referred to as *oblivious*. In contrast, an *adaptive* network routes a message based on the utilization of local paths.

A *wormhole* routing network establishes a connection between the source and destination nodes through which the message flows. If a needed path is already occupied, progress toward establishing the connection is blocked until the needed path is relinquished. Once a connection is established, the full message flows through it without interruption. A partially-routed blocked message occupies paths that may in turn block other messages.

John Ngai [Ngai89] characterized these networks while proposing adaptive enhancements. Some of Ngai's test results for 2D and 3D mesh and torus topologies are reproduced in figure 2. The test conditions of uniformly-random message destinations and fixed-length single-packet messages are reasonable simplifications of the conditions encountered in some of the parallel algorithms considered here. The major performance aspects of these networks are the *throughput* and average *latency* of messages as a function of *applied load* and *bisection bandwidth*.

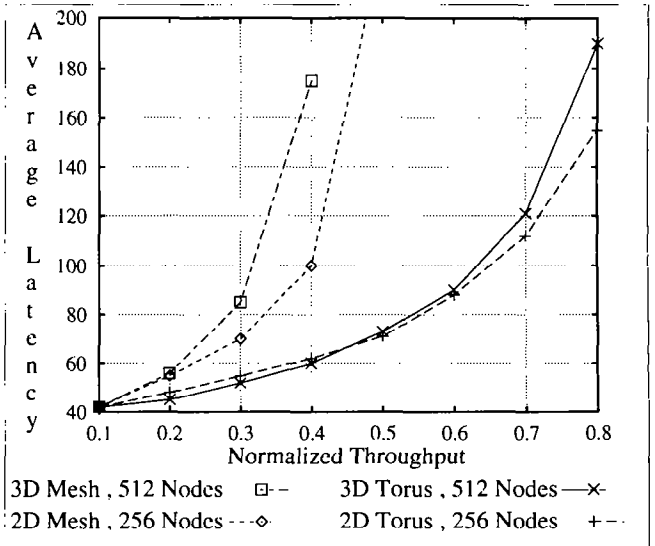

Fig. 2 - Average latency vs. normalized throughput (adapted from [Ngai89])

Throughput is a measure of aggregate network message delivery bandwidth.

Latency is the delay from a source node's injection of a message header into the network until the complete message exits the network at the receiving node.

Applied load is the aggregate message injection bandwidth into the network.

Bisection bandwidth is the aggregate peak bandwidth through the minimal set of routing channels that, when removed, splits the network into two equal and disjoint parts.

For a network with n nodes, let $n = k^a$, where k is even and a is the dimension of the mesh. The bisection width of a mesh is n / k channels. The bisection bandwidth of a mesh and torus is

$$b_{mesh} = c\, n / k \qquad (1)$$

$$b_{torus} = 2\, c\, n / k \qquad (2)$$

where c is the bandwidth of a single communications channel. Toroidal topologies have additional wrap-around connections that double the mesh bisection for a given k and n.

Under steady state conditions, network throughput equals the applied load. As the applied load increases beyond what the network can deliver, messages are queued at the source and delayed without bound; this *source queueing* time is separate from the network latency measure. Throughput in figure 2 is normalized to the maximum load that saturates the bisection bandwidth. All nodes inject fixed-length messages into the network at a uniform rate and to uniformly distributed destinations. The network is bidirectional with separate paths for message flow in opposite directions. Nodes on each side of the bisection send one-half of their messages across the bisection. An *injection bandwidth* of q at each node saturates the bisection paths when

$$q_{mesh} = 4\, c / k \qquad (3)$$

$$q_{torus} = 8\, c / k \qquad (4)$$

Since a torus has twice the bisection bandwidth of a mesh with identical dimensions, the injection bandwidth required to saturate the bisection is also doubled. At this *saturation load*, the aggregate bandwidth injected into the network is $n\,q$, which represents a normalized load of 1.0. The normalized load and normalized throughput are a fraction of the saturation load.

Communication times are estimated under the assumption that c is sufficiently great to keep the normalized load and throughput ≤ 0.3 for meshes and ≤ 0.6 for tori. Under these conditions, the average latency is roughly equal in either network of size n.

4. Parallel Algorithm Performance

Three classes of parallel algorithms are considered: image partitions with static and dynamic data distributions, and object partitions with block data distributions.

4.1. Image Partition with Static Data Distribution

In this class of algorithms, nodes are assigned one or more rectangular screen region to render [Challinger91] [Corrie+92] [Montani+92] [Nieh+92] [Vézina+92] [Yoo+91]. Data subsets are distributed among the nodes in a predetermined fashion. Such data distributions are considered static since each access to a given data point is always to the same node. To render their region(s), nodes access remote or local data as necessary, based on the current view transformation. Interleaved static data distributions can produce redistribution routing patterns that approximate the uniform random distribution used to characterize network performance. A fine-grain randomly-interleaved block data distribution achieves this and makes the redistribution size view-independent [Nieh+92].

Redistribution size may be lowered by replication of the data set. Define a data size d and a replication factor $r \mid (r \leq n)$. Each node needs about $1/n$'th of the data to render its assigned region. Nodes have $(r\ d/n)$ randomly located data points in their local memory, and of those, $(r\ d/n^2)$ points are needed for rendering their assigned region. Redistribution size is

$$m_{redist} = d - r\,d/n \qquad (5)$$

If $r = n$, every node has a complete copy of the data and the redistribution size is zero. As d increases for a fixed n and node memory size, r approaches zero and the redistribution size approaches d. The redistribution time for a 2D mesh under a normalized load of 0.3, is

$$\begin{aligned} t_{2Dredist} &= m_{redist} / (0.3\ n\ q_{mesh}) \\ &= (d - r\,d/n) / (1.2\ n\ c/k) \\ &= (d - r\,d/n) / (1.2\ n^{1/2}\ c) \end{aligned} \qquad (6)$$

Network throughput is $O(n^{1/2})$; if n is scaled in proportion to d, throughput increases too slowly to maintain constant redistribution time. Toroidal 2D topologies exhibit the same behavior except for a factor of four in their throughput. This is the expected behavior of mesh networks - the average injection bandwidth approaches zero as the mesh size increases.

Recall that the throughput of a 3D mesh of n nodes is $n^{1/6}$ greater than a 2D mesh for the same latency, so

$$t_{3Dredist} = (d - r\,d/n) / (1.2\ n^{2/3}\ c) \qquad (7)$$

Equation 7 shows that 3D topologies scale only slightly better than their 2D counterparts for this class of algorithms.

These results indicate that the redistribution time for image partition algorithms with static data distributions does not scale well on mesh networks as d and n are increased proportionally - the redistribution time grows as d and n increase. However, if the data size is kept constant, the redistribution time decreases as n increases.

Section 1 described how rendering efficiencies can reduce redistribution costs. These costs may also be lowered by using large caches to take advantage of temporal coherence [Corrie+92] [Hubbold93]. A node's cached values from the previous frame are likely to be a substantial fraction of the values it needs for this frame.

Note that computation time has not been addressed. The rendering method is the major factor impacting a node's computation time and is not addressed here. Equations 6 and 7 indicate that if d and n are increased together, regardless of the rendering method, the algorithm's redistribution time will grow and potentially limit the overall performance.

4.2. Image Partition with Dynamic Data Distribution

This class of parallel algorithms differs from all others in that data migrates among nodes in response to view changes - there is no particular node that will always have a particular data value. No implementations of this class of algorithms have been reported. The main advantage of a dynamic distribution over a static one is that network utilization is high, even for large systems. By matching the network and partition dimensions, and mapping neighboring screen regions to neighboring nodes, communication can be limited to *adjacent* nodes. Adjacent nodes are defined as having a routing distance of one or zero along each dimensions of the network. Network throughput for communication between adjacent nodes is similar to nearest-neighbor throughput. Due to the bounded distance between nodes, they are within a constant factor of each other. Throughput for adjacent-node communication is proportional to n.

View changes must be bounded to ensure that data subsets migrate no farther than adjacent nodes. Figures 3, 4, and 5 show experimentally measured redistribution sizes as a function of rotation

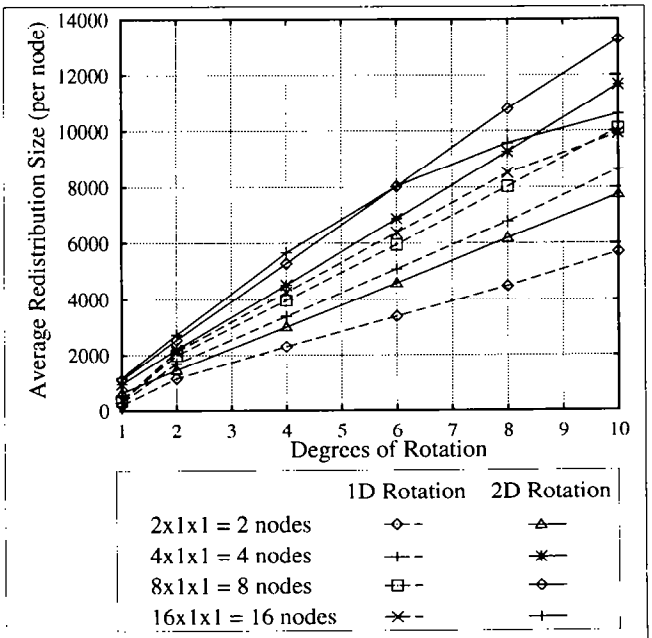

Fig. 3 - Redistribution with slabs on 1D network

about one or more axes. A 64^3 data set is transformed by the rotation angle given by the abscissa. Transformed data points that fall in different image regions from their starting regions are counted towards redistribution. Each figure has best-case and worst-case rotations for the angles given by the abscissa. Figures 3, 4, and 5 are for slab, shaft, and block image-space regions on 1D, 2D, and 3D mesh topologies, respectively. Although average redistribution size is plotted, the position of a node's data subset, relative to the axis of rotation, will affect the redistribution size at that node.

Slab distributions (Fig. 3) show an approximate doubling of average redistribution size between two and sixteen nodes. This is due to the fact that for n nodes, there are n-1 boundaries for data to migrate across. For large n, the average redistribution size remains constant. The downward curve in the sixteen node case is caused by a rotation angle large enough to cause data to migrate beyond adjacent regions. With slabs oriented as shown in figure 1, rotation about the vertical axis causes no redistribution. The 1D rotation data in figure 3 corresponds to rotation about the horizontal axis. The 2D rotation data in figure 3 is equivalent to 3D rotation and represents the worst-case redistribution size for a given angle, applied successively about each axis.

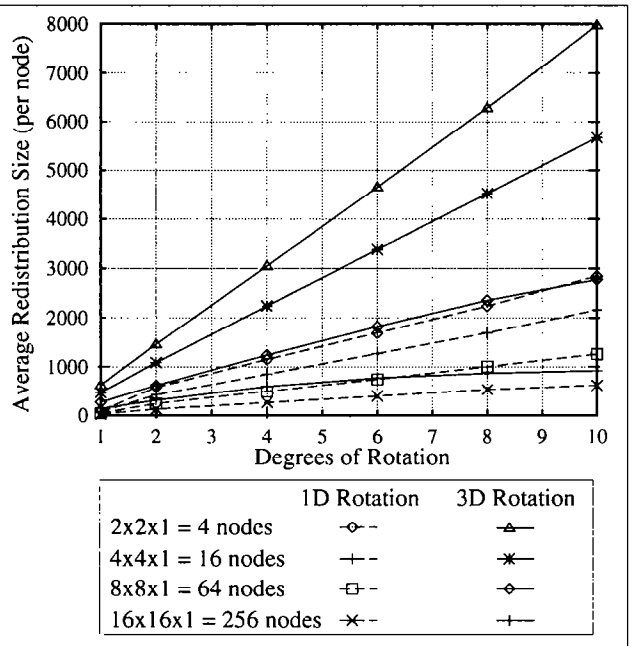

Fig. 4 - Redistribution with shafts on 2D network

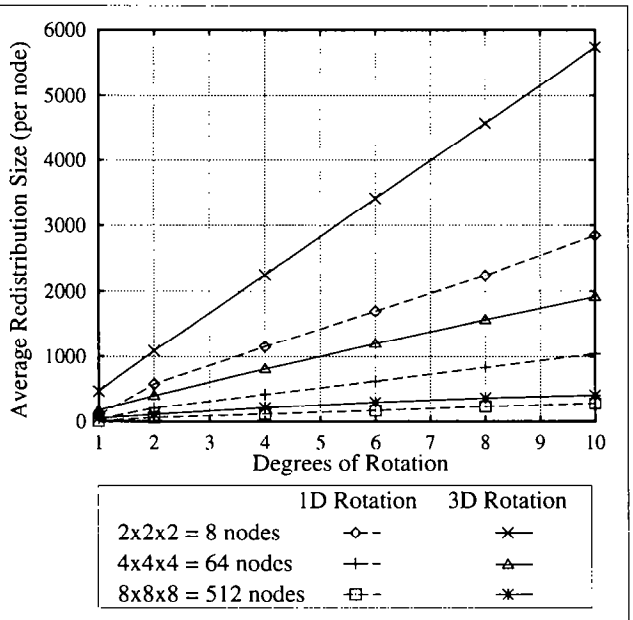

Fig. 5 - Redistribution with blocks on 3D network

Shaft (Fig. 4) and block (Fig. 5) distributions show a decrease in average redistribution size as n increases. In figures 4 and 5, the 1D and 3D rotations cause minimal and maximal redistribution size, respectively. With all dynamic distributions, the average redistribution size for a given rotation angle is proportional to the data size. When d and n increase proportionally, the net effect is still to increase the average redistribution size. For example, with a block distribution under 3D rotation, increasing the number of nodes from 8 to 64 decreases the average redistribution size to about 1/3 while the data size increases by a factor of eight; this produces a net factor of 8/3 increase in redistribution size at each node. In order to maintain a constant average redistribution size as d and n get larger, the rotation angle must decrease.

Although dynamic data distributions do not maintain a constant redistribution cost for proportionally increasing d and n, the redistribution cost does decrease as n increases for a fixed data size, and the redistribution size may be controlled by bounding the view changes. As systems grow to hundreds and thousands of nodes, this class of algorithms maintains high network utilization which is not achieved with the static data distribution characterized in section 4.2. It remains to be seen, however, whether implementations of dynamic data distribution algorithms fulfill these expectations.

4.3. Object Partition with Block Data Distribution

In object partition algorithms, nodes compute an image of their local data subset and redistribute the local images among themselves to perform the global composite that produces the final image [Challinger91] [Yoo+91]. The view point affects the redistribution size as a function of the aspect ratio of the data subsets. Slabs, shafts, and blocks vary from highly unbalanced aspect ratios to perfectly balanced ratios. As the view point changes, local images cover varying amounts of the screen, thereby varying the redistribution size. Figures 6, 7, and 8 are graphs of the average, per-node, redistribution size for different data subsets over a range of rotation angles. These graphs are experimentally obtained using a 64^3 data size and a 128^2 screen size. Rays are traced through the data subsets and the number of subsets encountered is recorded. The aggregate number of data subsets the rays pass through is the minimum redistribution size. The view transformation is affine and formulated so that a rotation of zero degrees produces a full-screen image of the data. Based on the data subset orientations in figure 1, all 1D rotations (Figs. 6, 7, 8) are specified by the abscissa and applied about the horizontal axis. The 2D shaft rotations (Fig. 7) create a worst-case by applying a constant 90° vertical axis rotation in addition to the variable horizontal axis rotation. The 3D block rotations (Fig. 8) create a worst-case by applying the abscissa angle equally about all three axes.

Figures 6, 7, and 8 show that block data distributions produce the lowest maximum redistribution size and achieve the most view-independence. The slab and shaft distributions have slightly lower best-case figures, but their strong view-dependence makes their worst-cases much higher. Therefore, blocks are considered the optimal data distribution. The local image size at each node in a block data distribution is approximately $p\ n^{-2/3}$ pixels, where p is the number of pixels in the final image. The local images must be

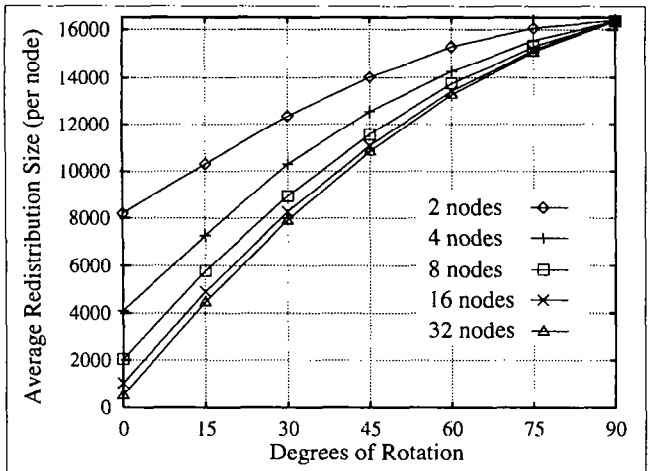

Fig. 6 - Redistribution with slab data subsets

composited properly to produce the final image. To achieve good load balance and network utilization, many small compositing regions are assigned to each node in a random or interleaved distribution. Approximately 1/n'th of each node's local image pixels are composited into the same node's assigned compositing regions so the total redistribution size is

$$m_{\text{redist}} \cong p\, n^{1/3} (1 - 1/n) \qquad (8)$$

Use of interleaved or randomized compositing regions also randomizes the redistribution network traffic, thereby matching the assumptions of the network performance model. The redistribution time for a 2D and 3D mesh under a normalized load of 0.3 is

$$t_{\text{2Dredist}} \cong m_{\text{redist}} / (0.3\, n\, q_{\text{mesh}})$$

$$\cong p\, n^{1/3} (1 - 1/n) / (0.3\, n\, 4\, c / k)$$

$$\cong p\, (1 - 1/n) / (1.2\, n^{1/6}\, c) \qquad (9)$$

$$t_{\text{3Dredist}} \cong p\, n^{1/3} (1 - 1/n) / (0.3\, n\, 4\, c / k)$$

$$\cong p\, (1 - 1/n) / (1.2\, n^{1/3}\, c) \qquad (10)$$

Toroidal topologies exhibit the same behavior except for a factor of four increase in network throughput. If the screen size p is held constant as the number of nodes increases, the redistribution size increases but the network throughput increases even faster so the time for redistribution actually decreases. Furthermore, since equations 9 and 10 are independent of d, both d and n may be increased without increasing the redistribution time. This behavior is superior to that of the image partitions, where redistribution time increases as d and n increase, and leads to the conclusion that object partitions scale well on 2D and 3D mesh network topologies. This scaling behavior is verified experimentally in the next section with tests run on the Touchstone Delta.

The above analysis holds for a constant screen size. For very large data sizes it may be necessary to increase the screen size to prevent undersampling. If we adopt the convention that $p^{1/2} = 2d^{1/3}$, then the local image size at each node becomes a function of the data set size ($4\, d^{2/3}\, n^{-2/3}$) and the total redistribution size is

$$m_{\text{redist}} \cong 4\, d^{2/3}\, n^{1/3} \qquad (11)$$

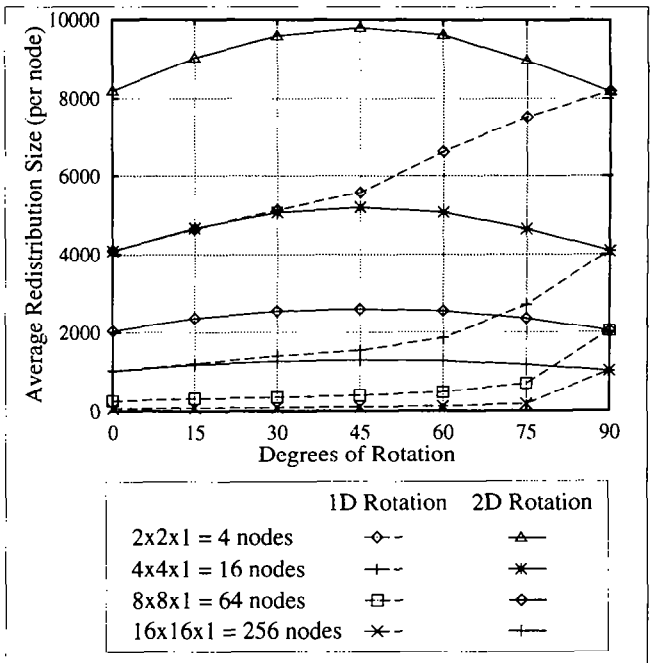

Fig. 7 - Redistribution with shaft data subsets

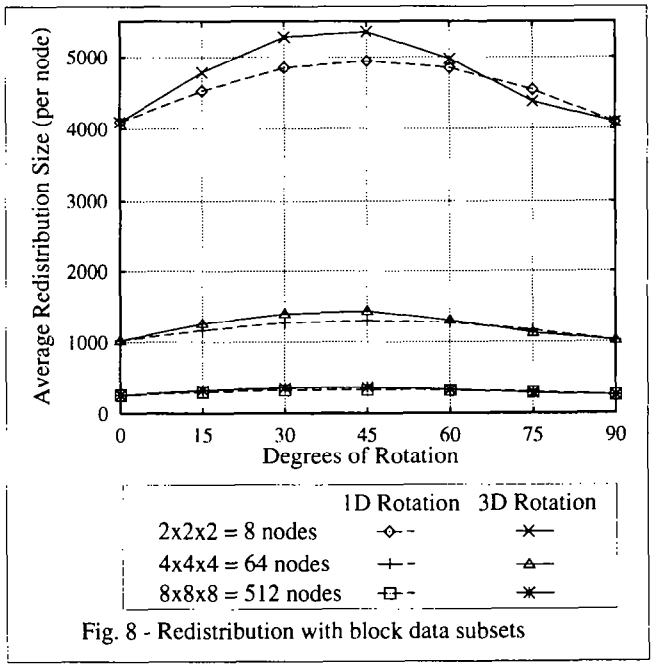

Fig. 8 - Redistribution with block data subsets

Substituting equation 11 into the expressions for redistribution time yields

$$t_{\text{2Dredist}} \cong 4\, d^{2/3} (1 - 1/n) / (1.2\, n^{1/6}\, c) \qquad (12)$$

$$t_{\text{3Dredist}} \cong 4\, d^{2/3} (1 - 1/n) / (1.2\, n^{1/3}\, c) \qquad (13)$$

When d and n are increased proportionately, these expressions exhibit the same asymptotic behavior as the image partition times given by equations 6 and 7, but for a given data set size, the redistribution time of an object partition is lower by a factor of $\sim(d/n)^{1/3}$ due to the local compositing that occurs before redistribution.

One disadvantage of object partitions is that load balance is difficult to maintain when the view point zooms in to a portion of the data set; potentially, only one node's data subset is visible making it responsible for rendering the entire image. There is a corresponding case for image partition algorithms: when the view point recedes so that all the data falls into one node's region. The application dictates the probability of either case occurring and therefore may impact the selection of a parallel algorithm.

5. Network Performance on Touchstone Delta

The Touchstone Delta with its 2D mesh network is used to experimentally verify that, for a fixed screen size, the redistribution time decreases as the number of nodes increases. The test program measures only the redistribution time, it does no actual rendering of an image. The size of each node's local image is computed, and the pixels are redistributed according to a randomly-interleaved static assignment of screen regions. The received pixels are ignored by the destination nodes, so compositing times are not included in the test times. Region assignments are varied to test for sensitivity to any pattern of assignment. Twenty different assignments were tested and the variations in redistribution time are small (< 20%) and not repeatable. These variations are likely to be due to network I/O traffic through the test partition from other user's programs. (The Delta supports multiple users in separate mesh partitions.) The sensitivity to region assignments appears to be negligible.

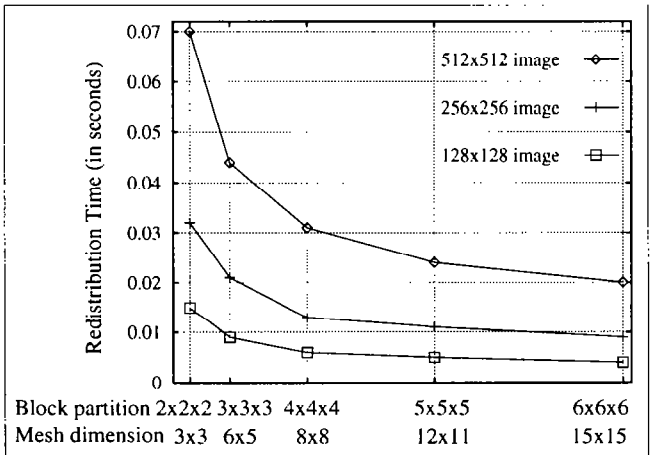

Fig. 9 - Redistribution times on the Touchstone Delta

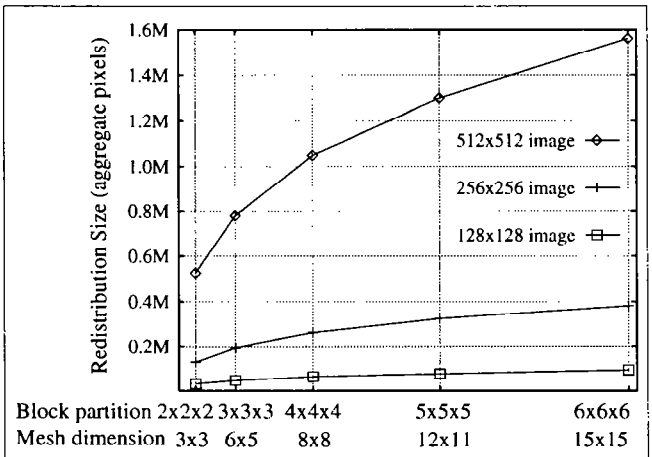

Fig. 10 - Redistribution sizes on the Touchstone Delta

Redistribution times are measured for three screen sizes and plotted in figure 9. An object partition with a block data distribution is mapped onto the smallest "near-square" 2D mesh with sufficient nodes. A square, or near-square mesh partition is used to maintain the largest bisection possible. The 3D to 2D mapping is simply done by enumerating the partition blocks in x, y, z-order and assigning them to the corresponding partition node number. For example, a 2×2×2 block partition fits into a 3×3 mesh with blocks ⟨0,0,0⟩, ⟨0,0,1⟩, ⟨0,1,0⟩, ... , ⟨1,1,1⟩ assigned to nodes 0, 1, 2, ..., 7, respectively. In this example case, the last node (node 8) is unused and doesn't contribute to the test. Figure 10 shows the redistribution sizes for the test cases used for figure 9. These two graphs verify the predicted behavior - as n increases, the redistribution size also increases, but the redistribution time *decreases*.

6. Volume Rendering on Touchstone Delta

An object partition volume rendering algorithm was implemented on the Touchstone Delta. The algorithm uses a contiguous block data distribution. Local images are rendered by ray casting to provide perspective views. The performance of this implementation is tabulated in figure 11 as the frame rates achieved for various data and system sizes. In all cases the screen size is 256^2 pixels. The data sets are analytically generated from Gaussian point and line sources (sampled at different densities). The image rendered in all the performance tests is shown in figure 12.

The Delta provides access to a frame buffer through an I/O node that feeds a HIPPI channel. Although the renderer assembles a complete 256^2 image in one node, it is not sent to the HIPPI frame buffer I/O node during these tests since updating the frame buffer limits the frame rate to about four Hertz at this image size.

Data size	System 2^3	3^3	4^3	5^3	6^3
64^3	1.8	2.9	2.7	5.0	
128^3	1.6	2.6	2.5	4.2	5.1
192^3			2.3	4.1	4.9

Fig. 11 - Touchstone Delta rendering performance (frames per second)

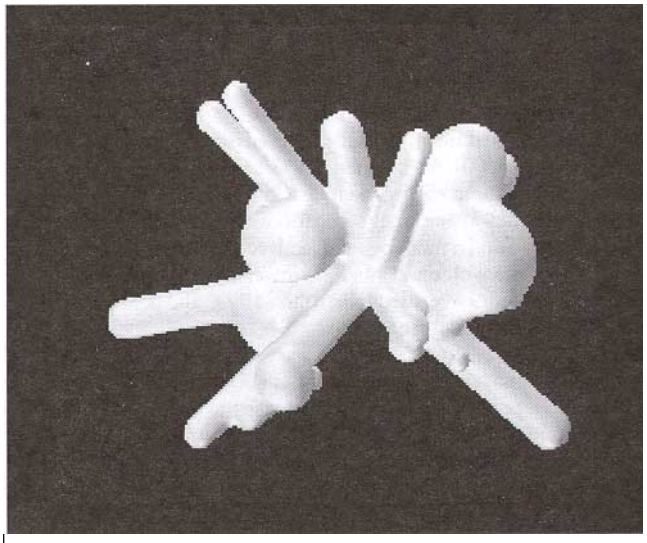

Fig. 12 - Isosurface rendering of test data

From figure 11, it is apparent that although this performance is as fast or faster than any reported for general purpose message passing multicomputers, ray casting time on the nodes is the performance bottleneck. Performance scaling is very nonlinear due to the effects of the ray casting speedups. Note the slower frame rate of the 4^3 system relative to the 3^3 system size and the low sensitivity to data size. This implementation uses adaptive sampling, adaptive ray termination, and an octree encoding of the minimum and maximum data value in each octant [Wilhelms+90]. The effectiveness of the speedups vary for different data block sizes and numbers. The nodes perform adaptive sampling with the isosceles-triangle recursive subdivision method [Shu+91] to render their local images. Each node constructs a unique octree for its data block. The octree "fit" of the features in the data will vary with the block dimensions and placement. Adaptive ray termination only effects local image rendering, so as the depth complexity of the partition goes up and the data blocks get smaller its effectiveness diminishes. This effect also occurs in object partition algorithms that use ray casting with adaptive ray termination on Pixel-Planes 5 [Yoo+91]. Although adaptive ray termination becomes less efficient as the number of blocks (and nodes) increases, its low computational overhead makes it worthwhile in all the cases tested.

Since the focus of this paper in on the redistribution costs of parallel algorithms, and not on rendering methods, the reader is referred to [Neumann93] for further details about the isosurface shading and load balancing used in the DELTA implementation. The implementation is described here only to illustrate a case where an object partition algorithm succeeds in reducing the redistribution costs to an insignificant level. A 6^3 system computes a 256^2 image in about 200 ms. The measured redistribution time for the same case is about 10 ms. - only five percent of the total frame time.

7. Summary and Discussion

Parallel volume rendering algorithms inherently require communication of data independent of the rendering method used at each node. The data redistribution sizes for three classes of parallel algorithms are derived from analysis and simulations. A network model is used to predict the time required for redistribution on mesh networks. The class of object partition algorithms with contiguous block data distributions exhibits the lowest redistribution time of any algorithm and scales well on mesh topologies - for a fixed image size, the redistribution time decreases as the data and system sizes increase. This scaling behavior is verified by tests on the Touchstone Delta. An implementation of an object partition algorithm is presented for the Touchstone Delta. For images of size $\geq 256^2$, frame rates are not limited by the network performance, but rather by the reconstruction and resampling required to render local images. The modest bandwidth (20 Mbyte/sec peak) of the communication links in the Delta are sufficient for an object partition algorithm to perform at real time (> 20 Hz.) rates. Unfortunately, the rendering speed of the nodes is too slow to produce local images at those rates.

Further rendering speedups and hardware accelerators are clearly important areas of future research. A large portion of software rendering time is consumed to reconstruct and resample the volume. Hardware acceleration of this process is possible with the 3D texture hardware provided on new graphics systems [Cullip+93]. If similar hardware were available on multicomputer nodes, real time volume visualization of large data sets would be possible.

When ray casting is used as the rendering method for an image partition algorithm, the efficiency of adaptive ray termination is preserved. As a result, implementers frequently report more linear speedups than those in figure 11. Despite this advantage, implementations on message passing multicomputers suffer from the high latency of remote memory accesses [Corrie+92] [Yoo+92] which makes redistribution time significant and lowers overall performance below that given in figure 11. A low latency network is necessary for ray casting to be efficient in an image partition algorithm; such a network is provided on the Stanford DASH [Nich+92] and results in the highest reported frame rates for a general purpose multicomputer - 48 nodes render a 209^2 image of a 128^3 data set at over 10 frames per second. This figure is quoted for an isosurface rendering mode which is the case where adaptive ray termination is most effective in reducing the rendering time and redistribution size. Such performance has not been achieved on message passing systems with high latency networks.

8. References

[Challinger91] Judy Challinger. "Parallel Volume Rendering on a Shared-Memory Multiprocessor." *Computer and Information Sciences, UC Santa Cruz*, Tech Report CRL-91-23, Revised March 1992.

[Corrie+92] Brian Corrie and Paul Mackerras. "Parallel Volume Rendering and Data Coherence on the Fujitsu AP1000." *Department of Computer Science, The Australian National University*, Tech Report TR-CS-92-11, August 1991.

[Cullip+93] Timothy Cullip and Ulrich Neumann. "Accelerating Volume Reconstruction with 3D Texture Hardware." *Department of Computer Science UNC Chapel Hill*, Tech Report TR93-027, July 1993.

[Danskin+92] John Danskin and Pat Hanrahan. "Fast Algorithms for Volume Ray Tracing." *1992 Workshop on Volume Visualization*, 91-98, October 1992. Workshop Proceedings.

[Dash] Daniel Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam. "The Stanford DASH Multiprocessor." *IEEE Computer*, 25(3):92-103, March 1992.

[Delta] Intel Corp. "Touchstone Delta System User's Guide." October 1991. Order Number: 312125-001.

[Hubbold93] R.J. Hubbold. "Experiments with Parallel Graphics Architectures and Algorithms." *Parallel Processing for Graphics and Scientific Visualization*, University of Edinburgh, May 1993. Workshop Proceedings.

[JMach] Michael Noakes and William J. Dally. "System Design of the J-Machine." *Proceedings of the Sixth MIT Conference of Advanced Research in VLSI*, 179-194, MIT Press, 1990.

[Levoy90] Marc Levoy. "Efficient Ray Tracing of Volume Data." *ACM Transactions on Graphics*, 9(3):245-261, July 1990.

[Montani+92] C. Montani, R. Perego, and R. Scopigno. "Parallel Volume Visualization on a Hypercube Architecture." *1992 Workshop on Volume Visualization*, 9-16, October 1992. Workshop Proceedings.

[Mosaic] Charles L. Seitz et at. "Submicron Systems Architecture - Semiannual Technical Report." *Department of Computer Science, California Institute of Technology*, Tech Report CS-TR-90-05, March 1990.

[Neumann93] Ulrich Neumann. "Volume Reconstruction and Parallel Rendering Algorithms: A Comparative Analysis." *Department of Computer Science, UNC at Chapel Hill*, Tech Report TR93-017, May 1993. Ph.D. Dissertation.

[Ngai89] John Y. Ngai. "A Framework for Adaptive Routing in Multicomputer Networks." *Department of Computer Science, California Institute of Technology*, Tech Report CS-TR-89-09, May 1989. Ph.D. Dissertation.

[Nieh+92] Jason Nieh and Marc Levoy. "Volume Rendering on Scalable Shared-Memory MIMD Architectures." *1992 Workshop on Volume Visualization*, 17-24, October 1992. Workshop Proceedings.

[Paragon] Intel Corp. "Paragon Operating System." January 1992. Document 312286-001.

[Shu+91] Renben Shu and Alan Liu. "A Fast Ray Casting Algorithm Using Isotriangular Subdivision." *IEEE Visualization'91*, 232-237, October 1991. Conference Proceedings.

[Vézina+92] Guy Vézina, Peter A. Fletcher, and Philip K. Robertson. "Volume Rendering on the MasPar MP-1." *1992 Workshop on Volume Visualization*, 3-8, October 1992. Workshop Proceedings.

[Westover89] Lee Westover. "Interactive Volume Rendering." *Chapel Hill Workshop on Volume Visualization*, 9-16, May 1989. Workshop proceedings.

[Wilhelms+90] Jane Wilhelms and Alan Van Gelder. "Octrees for Faster Isosurface Generation (extended abstract)." *San Diego Workshop on Volume Visualization*, 57-62, November 1990. Workshop Proceedings.

[Yoo+91] Terry S. Yoo, Ulrich Neumann, Henry Fuchs, Stephen M. Pizer, Tim Cullip, John Rhoades, Ross Whitaker. "Achieving Direct Volume Visualization with Interactive Semantic Region Selection." *IEEE Visualization'91*, 58-65, October 1991. Conference Proceedings.