8. Parallel Implementations

This chapter presents experimental verification of the network behavior predicted in chapter seven for object partitions. Tests are run on the Touchstone Delta running the NX/M message-passing kernel. A range of mesh sizes are tested and the data confirm the expected growth in redistribution size and shrinkage of redistribution time. Additionally, the object-partition algorithm described in chapter seven is implemented and tested on the Touchstone Delta and Pixel-Planes 5. There has been no other reported implementation of this algorithm to date. The algorithm's performance on Pixel-Planes 5 is compared to VVEVOL [Yoo+91], a volume rendering program for Pixel-Planes 5 that has evolved over some time and reports the fastest performance for a message-passing system to date.

8.1. System Overview

Pixel-Planes 5 and the Touchstone Delta both have i860 compute-nodes with relatively large memories and computation power. These are often classified as coarse-grain multicomputers. They both have message-passing communication models which make remote-memory accesses costly relative to local-memory accesses. Their network topologies are different. Pixel-Planes 5 uses an eight-channel ring while the Delta uses a two-dimensional mesh. There are major differences in how the systems incorporate frame buffers. Pixel-Planes 5 frame buffers interface directly to the ring network through one or two ring-channel ports. Pixel messages to a frame buffer are copied at the full ring-channel bandwidth. Messages must be formatted to cover all or part of a 128×128 pixel screen-region. The Delta frame-buffer interface is through a HIPPI port serviced by one of the I/O nodes which are on the edge of the computing mesh. Driver software for the I/O node may be customized to handle pixels in any format. Although the HIPPI bandwidth is reasonably high, feeding pixels to the I/O node is limited by the bandwidth of the mesh channels. Other characteristics of these two systems are tabulated below.

Feature	Pixel-Planes 5	Touchstone Delta
Memory per-node	8 Mbyte	16 Mbyte
Processor nodes	64×40 MHz i860	512×40 MHz i860
Network topology	8 Channel token ring	2D Mesh
CPU Message overhead	send=20 μs recv=70 μs	send+recv=60 μs (min)
Network to memory-buffer bandwidth	send = 51 Mbytes/sec recv = 62 Mbytes/sec	send = 16 Mbytes/sec recv = 16 Mbytes/sec
Peak channel-bandwidth	80 Mbytes/sec	20 Mbytes/sec

Messages costs are characterized by per-message CPU overhead and channel bandwidth

(given above for 4K byte message). Contention for a network path or a receiving node increases the latency of any transfer. Operating-system support for message passing is very similar in both systems. Transmitting nodes may use blocking or non-blocking send calls. Blocking sends enforce strict ordering of all messages by forcing a send call to wait for the message to completely enter the network before returning; thus any network contention effectively increases the CPU's message sending overhead. If a non-blocking send is used, contention increases the CPU's sending overhead only slightly, but the message itself is delayed by an arbitrary period of time. Pixel-Planes 5 and the Delta provide both blocking and non-blocking sends for flexibility of programming. Unlike the Delta however, Pixel-Planes 5 enforces the strict ordering of non-blocking sends to different nodes; all messages leaving a node are transmitted in the order the program issues them. The Delta allows non-blocking messages to different nodes to enter the network in arbitrary order; all messages to the same node are strictly ordered. These subtle differences in message-passing semantics are important to consider when porting applications from one system to another.

8.2. Mesh Redistribution Time

In chapter seven, an expression is derived (Eq. 7.9) for the redistribution time for object partitions on 2D mesh (and torus) topologies. In this section, the Delta mesh is used to experimentally verify the claim that for a fixed screen-size, the redistribution time decreases as the number of nodes increases. A test program is used that does no actual rendering of an image, but computes the number of pixels in each node's local-image and redistributes the pixels according to a randomly-interleaved static-assignment of screen regions. The pixels are received and 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 are tested and the resulting 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 negligible.

Redistribution time for three screen-sizes is plotted in figure 8.1. A 3D block-partition of the object lattice is mapped onto the smallest "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 done by enumerating the blocks in x, y, z-order and assigning them to the corresponding partition node-number. For example, a $2\times2\times2$ block-partition fits into a 3×3 mesh with blocks $\langle0,0,0\rangle$, $\langle0,0,1\rangle$, $\langle0,1,0\rangle$, … , $\langle1,1,1\rangle$ 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 8.2 shows the redistribution sizes for the test cases used for figure 8.1. These two graphs verify the behavior predicted in chapter seven - as *n* increases, the redistribution size also increases, but the redistribution time *decreases*.

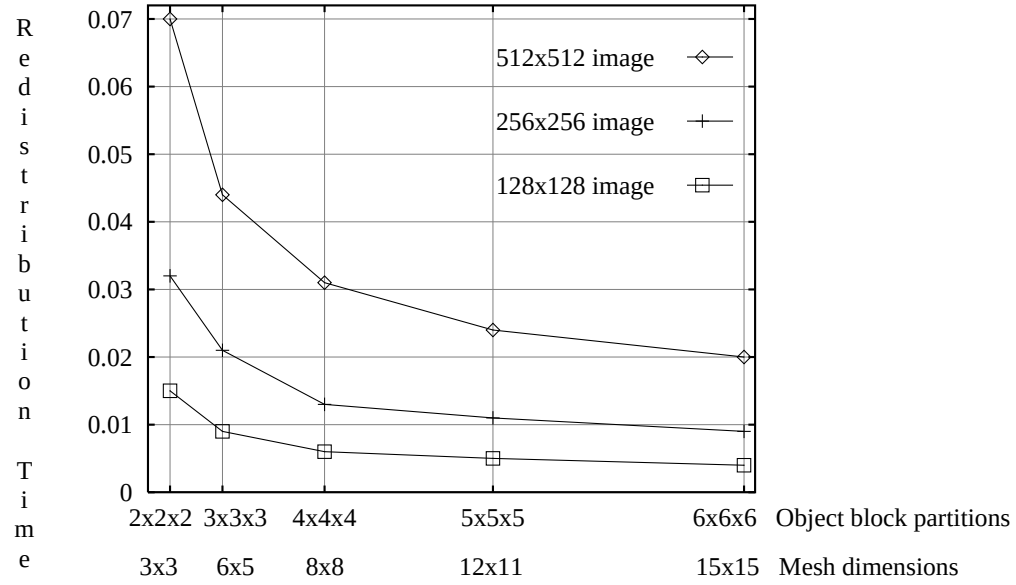

Fig. 8.1 - Redistribution times measured on the Touchstone Delta (in seconds)

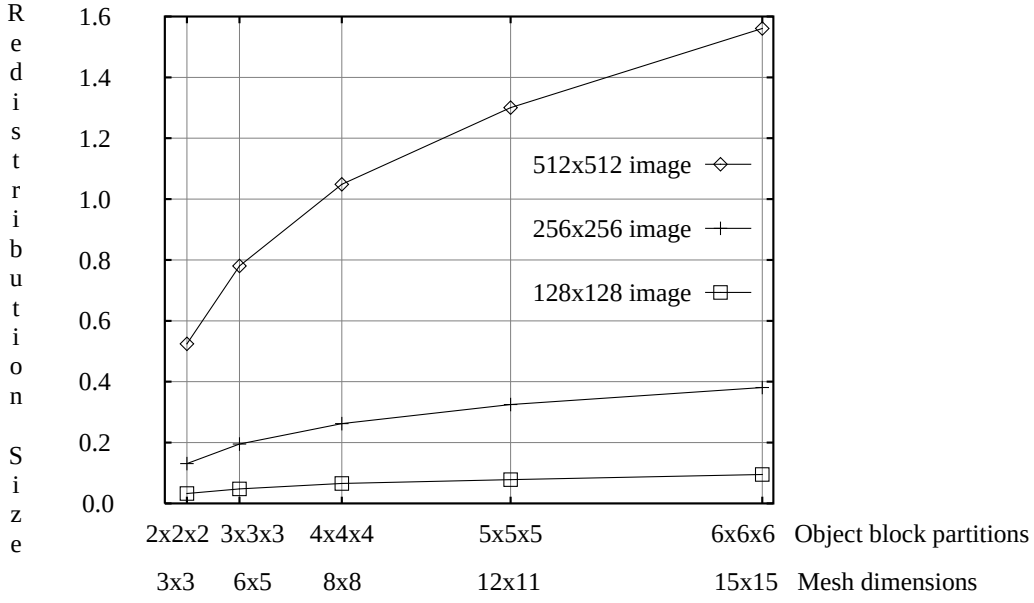

Fig. 8.2 - Redistribution sizes for tests on the Touchstone Delta

8.3. Touchstone Delta Implementation

This section describes a volume-renderer implementation for the Touchstone Delta. The algorithm is an object partition with a dynamic contiguous block distribution of the object lattice, and a static contiguous slab distribution of the image lattice. The image-lattice distribution differs from the optimal object-partition algorithm described in

chapter seven. The contiguous image-lattice distribution is used to simplify the frame buffer update which requires all pixels to be sent in one message. (Subsequent to these tests, the I/O-driver software was modified to accept a variety of pixel formats.) The image-lattice distribution is unlikely to impact the performance since the performance figures shown below indicate the redistribution time is not a bottleneck. Based on the data from section 8.2, the frame rates attained by this implementation are well-below the redistribution rate that the network can support. Software rendering-time on the nodes is the limiting factor. The performance of this implementation is tabulated below as the frame rate achieved for various data and system sizes. In all cases the screen size is 256^2 and the data set is the *mixed* data (sampled at different densities) used for the image quality tests.

Data size	**System size =** $\mathbf{2^3}$	$\mathbf{3^3}$	$\mathbf{4^3}$	$\mathbf{5^3}$	$\mathbf{6^3}$
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

The image rendered in all the performance tests is shown in figure 8.3. Although a complete image is assembled in one node, it is not sent to the HIPPI I/O node during these tests since updating the frame buffer limits the frame rate to about four updates per-second at this screen size.

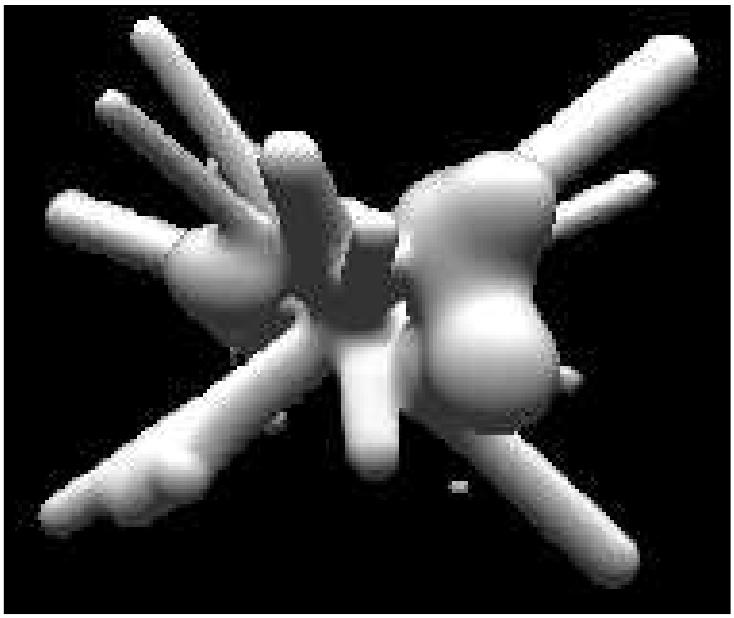

Fig. 8.3 - Isosurface rendering of *mixed* data set

Performance 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. This implementation uses ray casting with 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 image. 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; as the depth complexity of the partition goes up and the data blocks get smaller, the effectiveness of adaptive ray-termination diminishes.

Isosurface rendering is used since it is often faster than "cloud" rendering; rays terminates on the first surface they hit. The isosurface rendering method checks the minimum and maximum octree cell-values to decide whether a surface could lie in the current cell. If a surface could be in the currently-sampled cell, its position is estimated by linear interpolation of the ray's cell entry and exit values. At the intersection point, the gradient is computed by trilinear interpolation of the gradients at the eight cell corner-points. A Phong lighting model in then applied to produce the pixel's intensity. Although larger data sets have correspondingly larger numbers of cells, the octree hierarchy helps to maintain the number of cells traversed and tested fairly constant; thus, data size does not proportionally affect performance.

8.3.1. Load Balancing

Load balancing is performed by storing larger blocks of data at each node than are strictly required by the partition and adjusting the partition boundaries between nodes. This is a grid-computation load balancing method that has not previously been applied to volume rendering. This implementation replicates data near partition boundaries so that moving the partition boundary does not require communicating any data. The amount of replication determines the extent to which a load imbalance can be corrected. The test cases documented here were run with a replication factor of about two, each node had about twice the data size required by the initial partition-boundaries.

The load balancing task is distributed in order to prevent a single node from becoming a bottleneck in large systems. All rendering nodes communicate their rendering times for the last frame to a dedicated set of load-balancing nodes. These load-balancing nodes use a three-dimensional variant of a *summed-area table* [Crow84] to compute the average load on both sides of any partition plane. If the difference in the average load on both sides of a plane is greater than some threshold, then the boundary is moved to shift work to the more lightly-loaded side. This process is repeated for boundary planes perpendicular to each dimension. Once the new boundary-plane positions are established, their coordinates are communicated to the rendering nodes along with other

data that defines the next frame. The load balancing process is concurrent with the redistribution and global compositing so it adds no sequential task to slow the rendering rate. This method does not ensure perfect load-balance, but it improves it considerably in most situations. The table below shows some examples of the effects of load balancing on the Delta implementation. The load values are the times consumed to render a node's local-image. The slowdown percentage is 100 (Max. load – Avg. load) / Avg. load. All times are in seconds.

Balance	Sys/ Data	Avg. load	Max. load	Min. load	Frame rate	Slowdown
Off	2^3 / 64^3	0.48	0.72	0.32	1.3	48%
On	2^3 / 64^3	0.52	0.63	0.36	1.8	20%
Off	3^3 / 64^3	0.15	0.96	0.06	1.2	531%
On	3^3 / 64^3	0.16	0.34	0.06	2.9	112%
Off	4^3 / 64^3	0.086	0.47	0.03	2.0	454%
On	4^3 / 64^3	0.085	0.35	0.03	2.7	314%
Off	4^3 / 128^3	0.093	0.56	0.03	1.8	498%
On	4^3 / 128^3	0.091	0.40	0.02	2.5	341%
Off	4^3 / 192^3	0.098	0.51	0.03	1.5	421%
On	4^3 / 192^3	0.096	0.34	0.02	2.3	255%

8.4. Pixel-Planes 5 Implementation

This program is basically the same as the Delta program described above including the load balancing method. The major differences are due to the different frame buffer organization which causes the image-lattice distribution to be changed from slabs to shafts. By replacing the rendering loop with a simple block-fill of the local image, the performance of the ring network allows redistribution for 256^2 images to occur at > 15 Hz. for the largest test system with 40 processing nodes. This result indicates that network performance is not a bottleneck. Frame rates for 256^2 images are tabulated below. These performance figures are predictably similar to those for a comparable Delta configuration due to the similarity of the CPU performance in both systems.

Data size	**System size =** $\mathbf{2^3}$	$\mathbf{3^3}$	$\mathbf{4\times4\times2}$
64^3	2.0	3.1	2.9
128^3		3.1	2.7
192^3		2.9	2.5

This performance is comparable to that achieved by VVEVOL. VVEVOL is also an object-partition algorithm and is described in section 2.2.2. With 20 nodes rendering the *mixed* data at a ray density of 320×256, VVEVOL achieves about 2 frames per-second. At a ray density of 160×128, about 9 frames per-second are achieved. Both of these applications are processor-bound for high resolution images where the efficiency of the parallel algorithm's use of the network is not that important. However, for the lower resolutions and as more nodes are employed, the network does limit the frame rate and efficient network-utilization becomes important.