7. Parallel Algorithm Performance

This chapter examines the performance of the parallel algorithms described in chapter six. Performance is measured in terms of the per-frame network-communication time-requirement for redistribution. First, the amount of data communicated, or the redistribution size, is derived for each algorithm. By itself, the redistribution size is only a gross indicator of an algorithm's performance with a given system. In addition to the amount of data transferred, the network characteristics, the communications model, and the distribution of data accesses determine how much time is consumed by the redistribution process. The redistribution time of each class of algorithms is analyzed for two and three-dimensional mesh and toroidal network topologies. Expressions are derived for the time consumed by the redistribution process on these networks. These estimates are verified experimentally in chapter eight. This analysis and experimental verification are the the basis of the thesis claim that "*An object partition has the lowest communication costs and, for a constant image size, scales well on 2D mesh network topologies....*"

7.1. Network Performance Model

A model of network behavior is necessary to make the algorithm performance analysis general and applicable to a range of systems. Our focus will be the current generation of mesh and toroidal networks employing *virtual cut-through oblivious wormhole* routing techniques [Delta] [Paragon].

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 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] has 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 7.1. 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 we consider. These results are used to model network behavior and predict performance.

The major performance aspects of these networks are the *throughput* and average *latency* of messages as a function of *applied load* and *bisection bandwidth*.

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

$$b_{\text{bis-mesh}} = b_{\text{link}}\, n / k \qquad (7.1)$$

where b_{link} is the bandwidth of a single communication-channel.

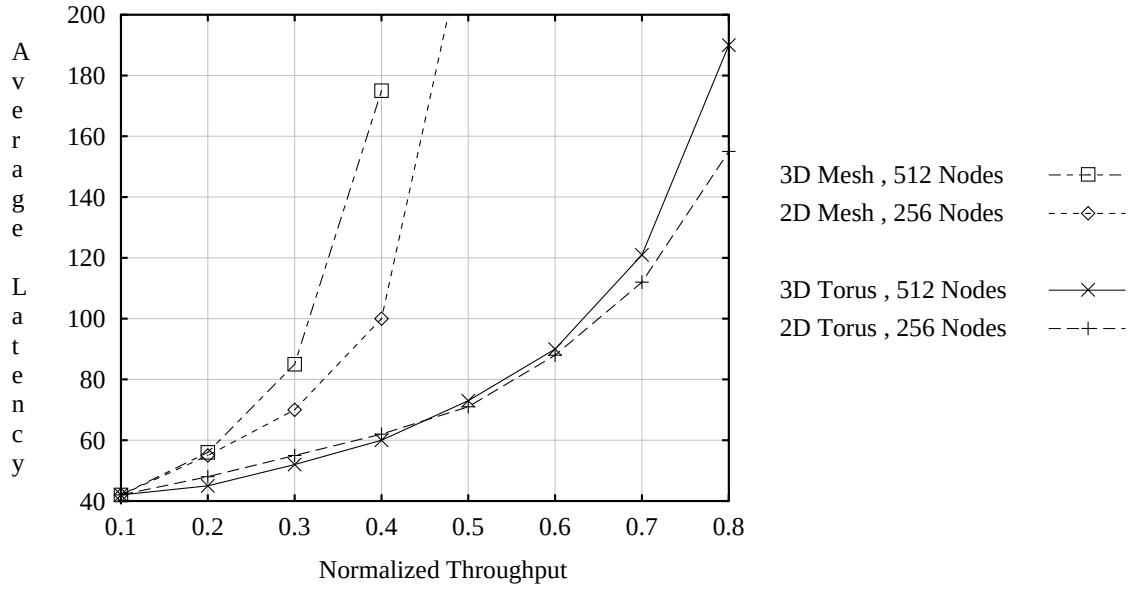

Fig. 7.1 - Average latency vs. normalized throughput (adapted from [Ngai89])

Toroidal topologies have additional wrap-around connections that double the bisection for a given k and n.

$$b_{\text{bis-torus}} = 2\, b_{\text{link}}\, n / k \qquad (7.2)$$

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 7.1 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. Assume a bidirectional network 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

$$q_{\text{mesh}} = 4\, b_{\text{link}} / k \qquad (7.3)$$

$$q_{\text{torus}} = 8\, b_{\text{link}} / k \qquad (7.4)$$

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 = $n\, q$, which represents a normalized load of 1.0. The normalized load and normalized throughput are a fraction of the saturation load.

Redistribution times are estimated under the assumption that b_{link} 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 any network of size n. Given a mesh of size n with channel bandwidth b_{link}, figure 7.1 indicates that adding connections to change the mesh to a torus doubles the normalized throughput for about the same latency, and since the saturation load also doubles, network performance actually increases by a factor of four. Comparing n nodes in a 2D mesh to n nodes in a 3D mesh, the saturation load of the 3D network is a factor of $n^{1/6}$ larger than its 2D counterpart. Since the normalized throughput of 2D and 3D networks are comparable, network performance increase is a factor of $n^{1/6}$. This leads to the conclusion that for a 2D mesh with $n < 4096$, the performance gained by adding toroidal connections is greater than that gained by raising the dimension of the network.

7.2. Image-Partition Redistribution Costs

7.2.1. Contiguous Static Data Distribution

Contiguous static data distributions are sub-optimal due to their view-dependent redistribution routing-patterns. For a given view, a node may require only local data; for another view, it may require only remote data from the most distant node in the network. This variation in network access increases the probability of imbalance in the redistribution cost for each node. Hot-spots may develop, increasing the latency of the network; two nodes may need most of their data from two other nodes positioned such that contention for some channel exists along the routing paths. Interleaving the image lattice reduces the view-dependence, but recall that an interleaved image-lattice in an image partition increases the redistribution size. For these reasons the contiguous static data distribution is not an optimal algorithm choice.

7.2.2. Interleaved Static Data Distribution

Interleaved static data distributions can produce redistribution routing patterns that approximate the uniformly-random distribution used to characterize network performance. A fine-grain, randomly-interleaved data distribution achieves this. As the granularity of the data subsets gets finer, the redistribution size becomes view-independent.

Redistribution size may be lowered by replication of the data set. We define a data set replication factor $r \mid (r \leq n)$, and assume that each node needs $1/n$ of the data to render its assigned region. Each node has $(r\ d/n)$ randomly-located data points in its local memory, and of those, $(r\ d/n^2)$ points are needed for their assigned region. Redistribution size is

$$m_{\text{redist}} = d - r\ d/n \tag{7.5}$$

If $r = n$, every node has a complete copy of the data and the redistribution size is zero. If memory size constraints make it impractical to raise r as n increases, redistribution size approaches d. The redistribution time using a 2D mesh under a normalized load of 0.3, is

$$\begin{aligned} t_{\text{2Dredist}} &= m_{\text{redist}} / (0.3\ n\ q_{\text{mesh}}) \\ &= (d - r\ d/n) / (1.2\ n\ b_{\text{link}}/k) \\ &= (d - r\ d/n) / (1.2\ n^{1/2}\ b_{\text{link}}) \end{aligned} \tag{7.6}$$

Network throughput is $\boldsymbol{O}(n^{1/2})$; 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.

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

$$t_{3Dredist} = (d - r\ d/n) / (1.2\ n^{2/3}\ b_{link}) \qquad (7.7)$$

The 3D topologies scale only slightly better than their 2D counterparts.

7.2.2.1. Implementation Issues and Summary

Static data distributions are shown in figure 6.6 as compatible with image, or object-order rendering methods. Object-order rendering requires a static task (image-lattice) distribution that provides a stable mapping between object-lattice points and the regions into which they fall under a view transformation. Load balancing a static task distribution is achieved by interleaving it at the expense of increased redistribution size. With a shared memory communications model, nodes may simply access the points that transform to their regions. In a message passing system, nodes must transform their data subsets and send them to the appropriate nodes for rendering. Using image-order rendering, increased redistribution size can be minimized or avoided by using a dynamic task distribution. Message passing systems are inefficient with image-order rendering methods since explicit requests for data must be sent, resulting in high latency for remote accesses. Shared memory is the preferred model when using image-order rendering.

The chart below summarizes the reasoning done in chapters six and seven about image partitions with static data distributions. The relative strengths and weakness of the options are given by the three columns on the right which indicate the expected performance of the algorithm with respect to the load balancing performance, the redistribution size (more data is "poor"), and the network throughput. This is only expected behavior - not measured or simulated. Based on the discussions in this work, the optimal algorithm in the class of image partitions with a static data distribution combines:

- static interleaved blocks of the object lattice,
- dynamic contiguous slabs or shafts of the image lattice,
- an image order rendering method, and
- a shared memory communications model.

Data distribution	Task distribution	Rendering	Communication	Load	Redist	Network
1) interleaved - static	interleaved - static	image	shared memory	Avg	Poor	Good
2) interleaved - static	interleaved - static	object	message passing	Avg	Poor	Good
3) interleaved - static	interleaved - dynamic	image	shared memory	Good	Avg	Good
4) interleaved - static	**contiguous - dynamic**	**image**	**shared memory**	**Good**	**Good**	**Good**
5) contiguous - static	interleaved - static	image	shared memory	Avg	Poor	Avg
6) contiguous - static	interleaved - static	object	message passing	Avg	Poor	Avg
7) contiguous - static	interleaved - dynamic	image	shared memory	Good	Avg	Avg

7.2.3. Dynamic Data Distribution

In a dynamic data distribution, data blocks migrate among the nodes in response to view transformation changes. By equalizing the network and task partition-dimensions, and by mapping neighboring image-regions to neighboring nodes on the network, communication can be limited to *adjacent* nodes. Define an *adjacent* node to be any node whose maximum network-distance from a start node is one along all dimensions of the network. In a 2D torus, there are 8 nodes adjacent to any node. In a 3D torus each node has 26 adjacent nodes. Communication with adjacent nodes is not nearest-neighbor communication since contention can occur. However, because of the bound on the message distance, the network throughput is lower by only a constant factor. Network throughput for nearest-neighbor communication is limited only by the channel bandwidth since there is no contention for paths. If nodes send and receive data simultaneously, the maximum network throughput is $= n\, b_{\text{link}}$, and is achieved with minimum latency. Nearest-neighbor and adjacent-neighbor network behavior is similar since both have throughput proportional to the number of nodes n.

Redistribution size is proportional to the change in view-point for successive frames. View-point changes are composed of rotations, translations, and scaling. Rotation is the most-common view change when visualizing data. Figures 7.2, 7.3, and 7.4 plot the average per-node redistribution size for different subsets of data over a range of system sizes and rotations. These graphs are experimentally obtained using a 64^3 data set, and the results are scalable to any data size. Each point is transformed twice, once by a view matrix with no rotation, and once by a matrix with the rotation specified by the abscissa. The image lattice is partitioned into equal subsets as indicated by the graph legend. If a point falls into the same image-lattice subset under both transformations, no contribution to redistribution size is incurred. If a point transforms to different image-lattice subsets, then it would have to migrate over the network, and therefore contributes to the redistribution size. Rotations are applied successively about the screen x, z, and y-axes. Image-lattice slabs are horizontally oriented to encompass a set of scan lines; shafts are rectangular screen-regions extending away from the viewer along the screen z-axis. The redistribution size given for the block distribution does not include the communication needed for the final compositing step between layers of blocks.

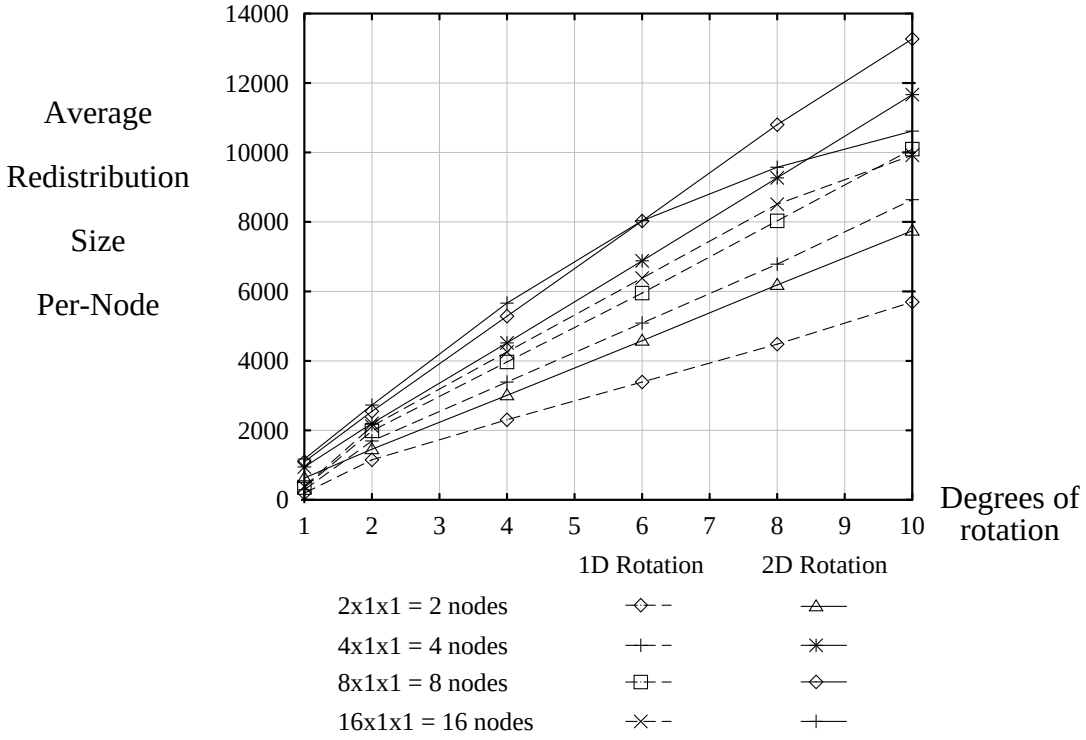

Fig. 7.2 - Redistribution with slab image-lattice distribution

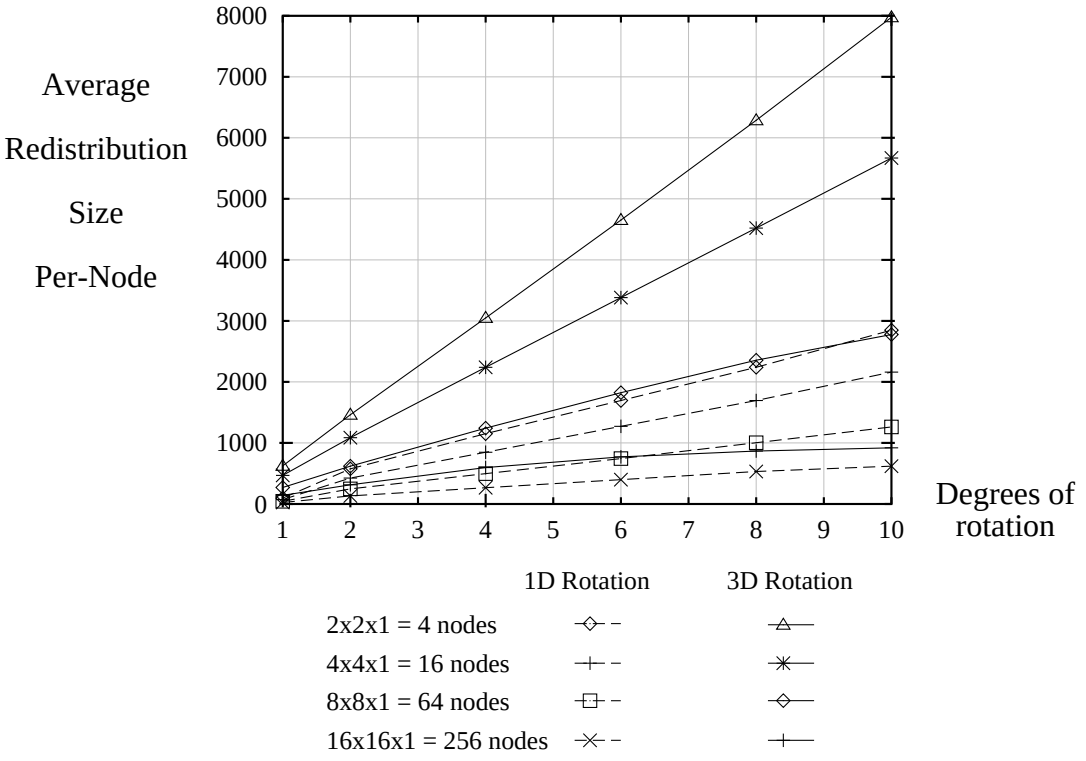

Fig. 7.3 - Redistribution with shaft image-lattice distribution

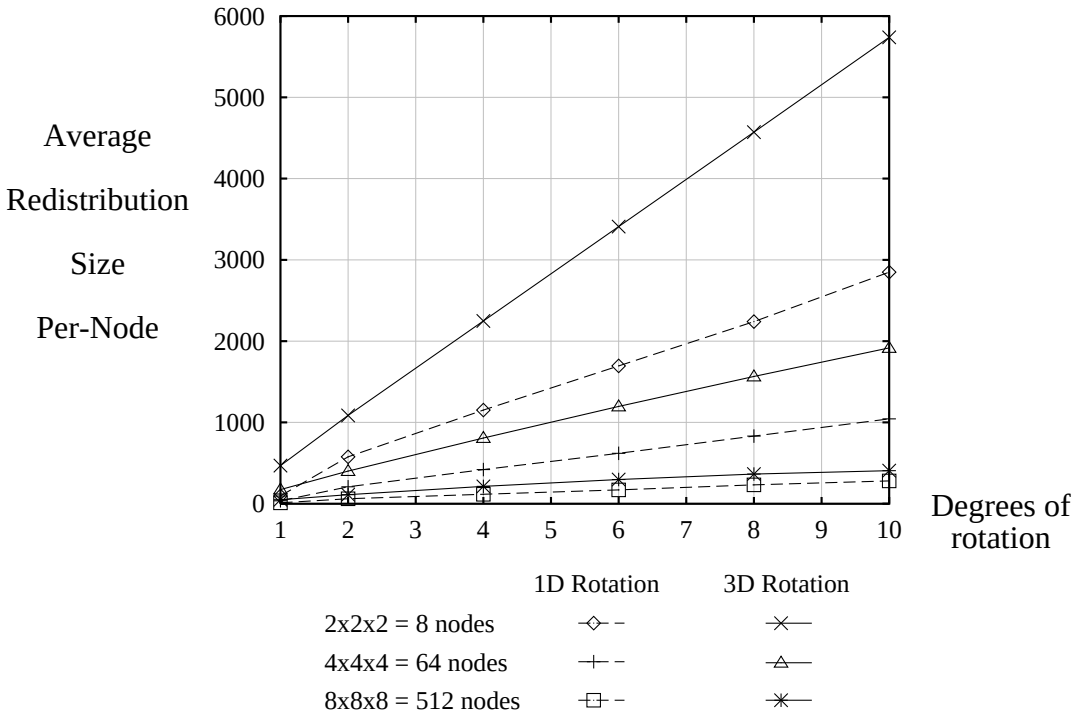

Fig. 7.4 - Redistribution with block image-lattice distribution

7.2.3.1. Scaling and Speedup

Slab distributions show an approximate doubling of average redistribution size as the number of nodes increases from two to sixteen. This is mainly due to the fact that for *n* nodes there are *n*–1 boundaries for data to migrate across. For large *n*, the average per-node redistribution size remains constant with respect to *n*. The downward curve for the case of sixteen nodes with 2D rotation occurs because the redistribution size is not a linear function of rotation angle and the angle has increased sufficiently to cause points to migrate beyond neighboring regions. To preserve adjacent-node network behavior, the rotation angle change should be constrained to limit data migration to neighboring nodes. Shaft and block task-distributions show a decrease in average redistribution size as the number of nodes increases for a fixed data-size *d*. These dynamic task distributions provide very good speedup since adding nodes causes the redistribution time to decrease.

For all dynamic data distributions, the average redistribution size is proportional to data size. If *d* and *n* are increased proportionally, the graph data indicates that average redistribution size will increase. For example, with a block task 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 8 - producing a net factor of 8/3 increase in redistribution size at each node. We conclude that dynamic data distributions do not scale well with respect to increasing *d* and *n*, but they do provide good speedup for fixed data sizes.

7.2.3.2. Data Block Size

Figures 7.2 - 7.4 represent a lower bound for redistribution size. To achieve this minimum, the data distribution block size must be one point, and each point must be transformed - an unnecessary computational expense. A single point is also an inefficient block-size from the network's perspective since a separate communication event occurs for each migrating block. Message passing systems allow selection of the block size. A coarser block granularity allows transformations of the eight corner points to determine the entire block's position in the image lattice. The block is sent to all nodes whose image lattice subset intersects the block's transformed position. With this approach, lower computation cost is achieved at the expense of increasing the redistribution size by a factor *g*, where

g = total number of blocks communicated / number of unique blocks moved (7.8)

For a given task distribution, if we limit the maximum data-block size so that it fits within any node's image-lattice subset, then there is a maximum rotation-angle between frames for which no block migrates farther than an adjacent node. For slab task distributions, blocks can only move to one neighbor, so $g = 1$ always. With shaft task distributions, data blocks can move to possibly three adjacent nodes, and for block task distributions, data blocks can move to up to seven adjacent nodes. As block size increases to fill an image lattice subset, $g = 3$ and $g = 7$ for shaft and block task distributions, respectively.

A trade-off must be made between lowering *g* by using small blocks and using the network efficiently with large blocks. Shared memory systems are likely to transfer remote data in small blocks so their *g* factor will be small. In message passing systems, high message overhead and transformation time for small blocks is traded against a high *g* factor for large blocks. Message passing systems also must contend with replicated blocks all being forwarded to the same set of nodes, further increasing redistribution size. One possible solution is to designate one block as "original" when copies are made, and allow only "original" blocks to migrate each frame.

7.2.3.3. Implementation Issues on Message Passing Systems

With a dynamic data distribution, image-order rendering with a message passing system is complicated by high remote-data access-latency, or complex indexing of data blocks. High latency is a problem if the rendering process requests data blocks as they are needed. While the latency effects can be minimized by overlapping other tasks, the necessary context and schedule management adds overhead and complexity to the resampling process. Latency is less of an issue if all the data blocks are transmitted before, or during the start of rendering. Any data block may transform to a given node,

so complex indexing is required to store and access blocks in less memory than the whole data set requires.

An interleaved static task distribution (Fig. 6.6) is the only dynamic data option for systems with message passing communication models. The interleaved distribution increases the *g* factor, but that must be accepted to accomplish any load balancing. Since the mapping of tasks to nodes is static, message passing systems can use object-order rendering methods to avoid the latency problems described above. Using volume shearing for example, nodes transform their local data blocks and send them, as necessary, to adjacent nodes. A message-handler process receives transformed blocks (in arbitrary order) and resamples them into the image lattice by performing the first shear pass. When all the required blocks have arrived, the remaining shear pass completes the resampling. In this example, the redistribution message latency is overlapped with the transformations and the first pass shear. No explicit user code is needed to switch context from sending blocks to receiving them since handlers are a common feature of message passing kernels. We conclude that the optimal algorithm for message passing systems in the class of image partitions with a dynamic data distribution combines:

- static interleaved slabs or shafts of the image lattice,
- dynamic interleaved blocks of the object lattice, and
- an object order rendering method.

7.2.3.4. Implementation Issues on Shared Memory Systems

Systems with a shared memory communications model can use image-order rendering methods with a contiguous-dynamic, or interleaved-static task-distribution. Dynamic task distributions offer minimal redistribution size but require active load balancing. Static distributions pay for load balancing with increased redistribution size. Since the message size is typically small for shared memory systems, image-lattice subsets can be small, allowing good load balance to be achieved by interleaving. A major advantage of the contiguous dynamic option is that all the ray-casting speedups may be applied efficiently. Recall that ray-casting speedups are adaptive ray-termination, adaptive screen-sampling, and presence structures like octrees and distance-transform volumes. Dynamic data distributions with slab or shaft task-distributions can use adaptive ray-termination effectively. Block task distributions are less efficient since nodes compute ray samples in rear blocks that may be fully occluded by opaque front blocks. Adaptive screen-sampling is inefficient with interleaved static task-distributions since there are shared rays at the boundaries of screen regions that are redundantly computed [Nieh+92]. Presence structures may be used with any dynamic data option but must be replicated on each node. We conclude that the optimal algorithm for shared memory systems in the class of image partitions with a dynamic data distribution combines:

- dynamic contiguous slabs or shafts of the image lattice,
- dynamic contiguous blocks of the object lattice, and
- an image order rendering method.

7.3. Object-Partition Redistribution Costs

In an object partition, nodes compute an image of their local data and redistribute the local images among themselves to perform the global composite that produces the final image. 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 varies, local images cover varying amounts of the screen, thereby varying the redistribution size. Figures 7.5, 7.6, and 7.7 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 intersected 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 zero rotation produces a full-screen image of the data. All 1D rotations are specified by the abscissa and applied about the image x-axis. The 2D shaft rotations create a worst-case by applying a constant 90° y-axis rotation in addition to the variable x-axis rotation. The 3D block rotations create a worst-case by applying the abscissa angle equally about the image x, y, and z-axes. Data slabs lie parallel to the screen x and z-axis under zero rotation. Data shafts cover a rectangular screen-region extending along the screen z-axis under zero rotation.

A block data-distribution produces the lowest maximum-redistribution-size and is most view-independent. The slab and shaft distributions have slightly-lower best-case figures,

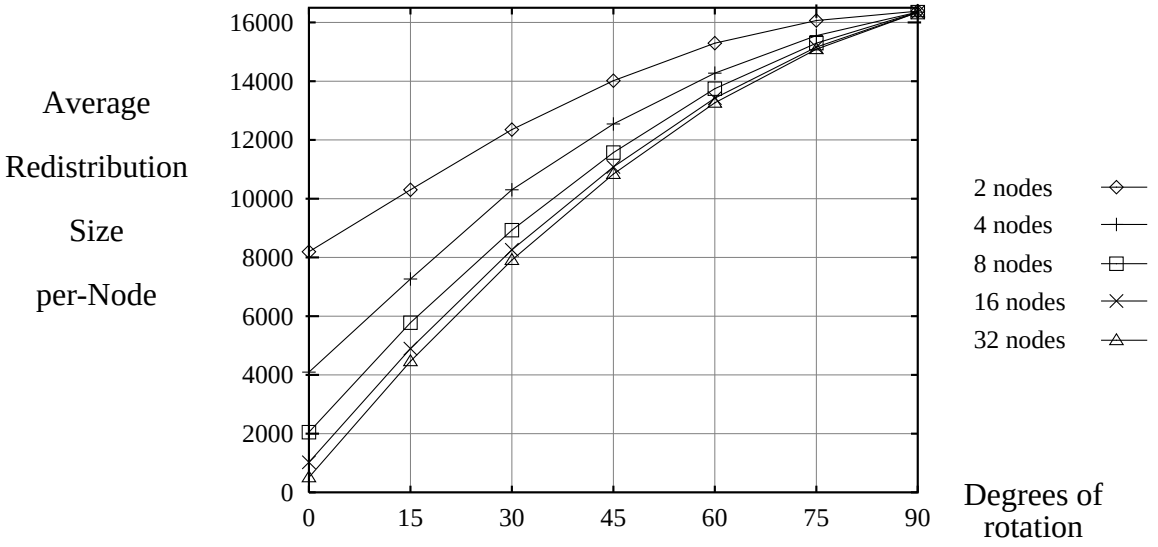

Fig. 7.5 - Redistribution with slab object-lattice distribution

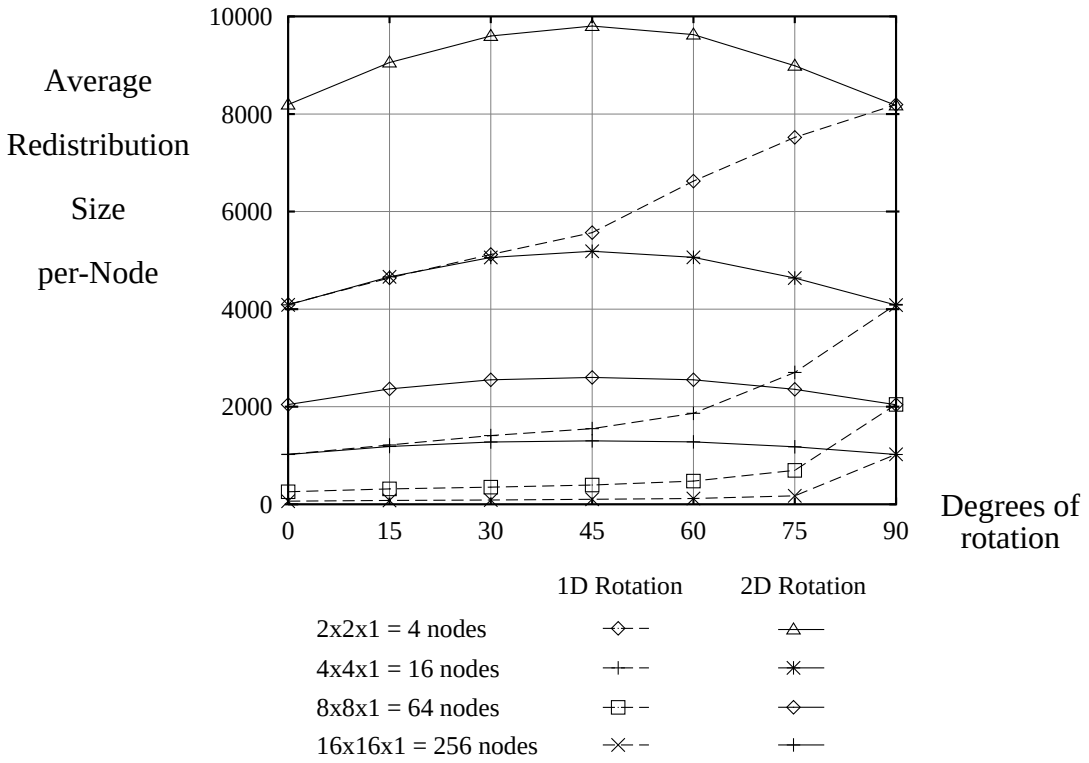

Fig. 7.6 - Redistribution with shaft object-lattice distribution

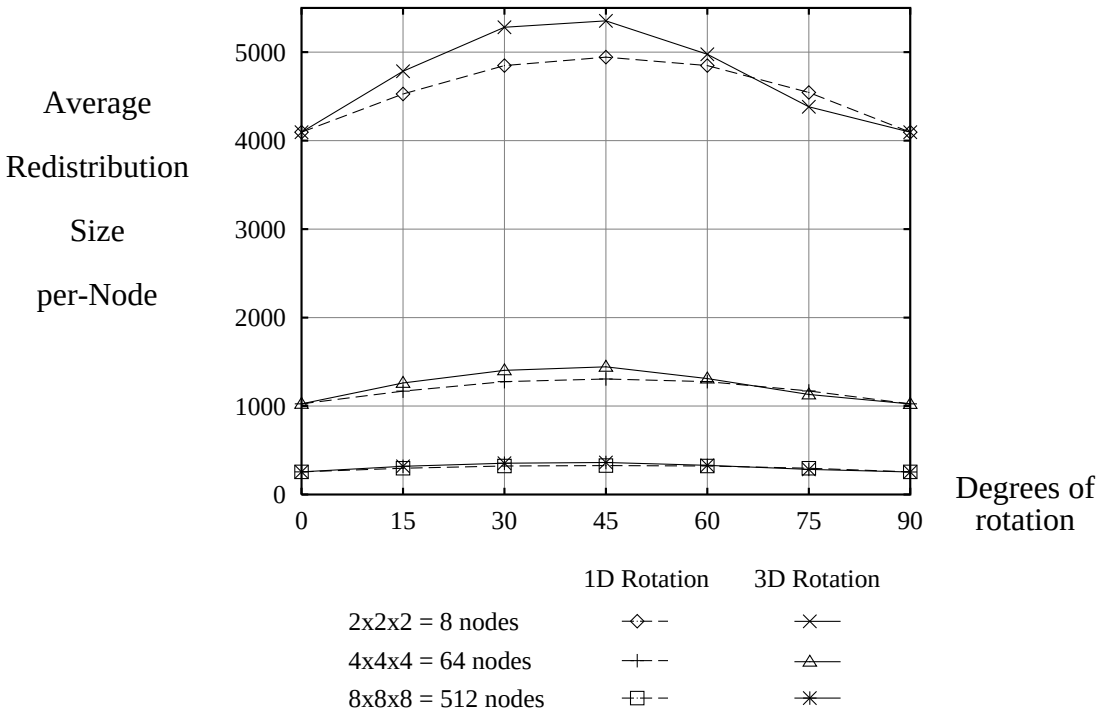

Fig. 7.7 - Redistribution with block object-lattice distribution

but their strong view-dependence makes their worst-cases much higher. Therefore, we argue that blocks are the optimal data-distribution. The local-image size at each node in a block data-distribution is $\cong p / n^{2/3}$.

Assuming an interleaved slab or shaft image-lattice distribution, approximately 1/nth of each node's local-image pixels are composited into the same node's locally-assigned regions so the total redistribution size is

$$m_{\text{redist}} \cong p\, n^{1/3} (1 - 1/n) \tag{7.8}$$

Use of an interleaved image-lattice distribution also randomizes the redistribution network-traffic thereby matching the assumptions of the network-performance model. The redistribution time using a 2D and 3D mesh under a normalized load of 0.3 is

$$\begin{aligned} t_{\text{2Dredist}} &\cong m_{\text{redist}} / (0.3\, n\, q_{\text{mesh}}) \\ &\cong p\, n^{1/3} (1 - 1/n) / (0.3\, n\, 4\, b_{\text{link}} / k) \\ &\cong p\, n^{1/3} (1 - 1/n) / (1.2\, n^{1/2}\, b_{\text{link}}) \\ &\cong p\, (1 - 1/n) / (1.2\, n^{1/6}\, b_{\text{link}}) \end{aligned} \tag{7.9}$$

$$\begin{aligned} t_{\text{3Dredist}} &\cong p\, n^{1/3} (1 - 1/n) / (1.2\, n^{2/3}\, b_{\text{link}}) \\ &\cong p\, (1 - 1/n) / (1.2\, n^{1/3}\, b_{\text{link}}) \end{aligned} \tag{7.10}$$

Toroidal topologies exhibit the same behavior except for a factor of four increase in network throughput. 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 7.9 and 7.10 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. This is the basis of the third thesis claim that, *"An object partition ..., for a constant image-size, scales well on 2D mesh network topologies. The network-channel bandwidth-requirement actually decreases as the problem is scaled to larger systems and volume data sets."* This scaling behavior is verified in chapter eight with test runs on the Touchstone Delta.

The above analysis holds for the usual case where the screen size is constant. For very-large data sizes it may be necessary to increase the screen size to prevent undersampling. Maintaining the convention adopted in chapter four that $p^{1/2} = 2d^{1/3}$, the local-image size at each node becomes a function of the data size $\cong 4\, d^{2/3}\, n^{-2/3}$, and the total redistribution size is

$$m_{\text{redist}} \cong 4\, d^{2/3}\, n^{1/3} \tag{7.11}$$

Substituting equation 7.11 into the expressions for redistribution time yields

$$t_{2\text{Dredist}} \cong 4\, d^{2/3} (1 - 1/n) / (1.2\, n^{1/6}\, b_{\text{link}}) \qquad (7.12)$$

$$t_{3\text{Dredist}} \cong 4\, d^{2/3} (1 - 1/n) / (1.2\, n^{1/3}\, b_{\text{link}}) \qquad (7.13)$$

When d and n are increased proportionately, these expressions exhibit the same asymptotic behavior as the image-partition times given by equations 7.6 and 7.7, but for a given data size, the redistribution time of an object partition is lower by a factor of $\sim d^{1/3}$ due to the local compositing that occurs before redistribution. This is the basis for the thesis claim that, *"An object partition has the lowest communication costs...."*

7.3.1. Composite Sequencing

For object partitions, the sequence of redistribution is important since the final compositing must be done in view order. Either, nodes must buffer image data that can not immediately be composited, or data transmission must be sequenced so local-images arrive at each node in a defined order. In the buffered case, sufficient memory must be available on each node to allow the worst-case arrival sequence which requires buffering all the image data for a region. Buffer size is the product of the region size and the depth complexity. Depth complexity c is the maximum number of local-image pixels overlapping at a pixel in the final image. For block data distributions,

$$c = 2\, n^{1/3} \qquad (7.14)$$

If the screen size is kept constant and equal-size image regions are assigned to each node for compositing, then region size is = p / n. The maximum buffer size needed at each node is

$$\begin{aligned} \text{Buffer size} &= c\, p / n \\ &= 2\, n^{1/3}\, p / n \\ &= 2\, n^{-2/3}\, p \end{aligned} \qquad (7.15)$$

Note that the buffer size at each node actually decreases as nodes are added to the system since region-size decreases faster than depth-complexity increases.

In the sequenced approach, local-images must arrive at the nodes in view order so that they can always immediately be composited. The compositing sequence is like a planar wavefront moving through the volume in discrete steps. The number of steps required for complete compositing is the depth complexity c given in equation 7.14. During each step, only a subset of the nodes are sending data. The number of nodes sending data at any time is a function of the redistribution sequence. The most likely redistribution sequence occurs when a corner of the data set is closest to the view point. The

redistribution sequence (for front to back compositing) starts at the closest corner and moves diagonally to the farthest corner. Nodes that complete their redistribution-step trigger other nodes across their face-boundary. When starting from a corner, the number of nodes *u* sending local-image data at step *s* is

$$u = (s^2 + s) / 2 \qquad | \ 1 \le s \le n^{1\backslash 3} \tag{7.16}$$

The maximum value for *u* is reached when $s = n^{1/3}$. Because of the symmetry of the data's cubic shape, u decreases in the reverse sequence for $n^{1/3} < s \le 2n^{1/3}$. Under a perspective projection, the closest data point to the viewer may be on the face of the volume. For this condition, the wavefront is not planar but rather like a pyramid, and the number of nodes sending data is

$$u = 2s^2 - 2s + 1 \qquad | \ 1 \le s \le (n^{1\backslash 3} / 2) \tag{7.17}$$

For $s > n^{1/3} / 2$, the number of nodes sending data continues to increase, but less quickly since the wavefront is clipped by the volume dimensions. The maximum *u* value occurs at step $s = n^{1/3}$, where $u = n^{2/3}$. For many of the steps, *u* is too small to fully utilize the throughput of the network. This causes the redistribution time to be greater than for the buffered approach. Even with the optimistic assumption that all steps take constant time due to high network-throughput, redistribution time still grows as the depth complexity. The buffered approach is superior for large systems since it does not force such growth due to its higher utilization of the network.

7.3.2. Implementation Issues

Figure 6.9 illustrates the useful object-partition options. Section 7.3 argues that the optimal object-lattice distribution is blocks. Section 6.3.2 indicates that an interleaved image-lattice distribution is desirable since it has predictable network-performance and provides some load balancing of the compositing task. Fortunately, interleaving the image lattice in an object partition has no associated penalty. Slab or shaft image-lattice distributions are equally-efficient options although shafts allow a finer granularity of regions for interleaving.

Object partitions are very flexible in supporting the use of any rendering method. Object-order splatting could be used for the sake of its speed. Image-order ray casting is attractive since it supports perspective transformations and offers many speedup techniques. However, unlike image partitions, object partitions are not efficient at using adaptive ray termination. Each node renders its data block into a local-image regardless of whether it is fully occluded by others. The lack of true adaptive ray-termination can limit the speedup obtained with ray casting when *n* is increased for a fixed data size.