global data referencing is taking place, although performance tails off here as well. Unfortunately, the amount of testing done using the old operating system was limited, so additional results could not be obtained. It is clear from these results, though, that the operating system in a shared memory multiprocessor has significant impact on the overall performance. We feel confident that the latest version of the GP1000 operating system is better geared to the current machine and does indeed provide exceptional performance.

6.2.2. Comparison of Architectural Differences

In addition to the impact of the operating system, other factors can affect overall algorithmic performance. For instance, one would like to compare what would happen if a faster CPU or a faster switch node were to be employed in the machine. BBN has continually updated the Butterfly family of machines from the Butterfly 1, which used MC68000 processors with 1 megabyte of memory per board, to the current generation GP1000, which uses the MC68020 with 4 megabytes of memory per board. We were not able to test the algorithms on the original Butterfly, but we were able to test them on the next generation BBN multiprocessor, the TC2000. The TC2000 is a similar design to the GP1000 but there are significant differences which are illustrated in the tables on the page following the graphs. Table 6.1 shows the difference in processor characteristics, while table 6.2 shows the difference in the memory characteristics for the GP1000 and TC2000. In general, the primary differences between the two machines are the faster CPU in the TC2000, as well as a change in the basic switch node component from a 4 x 4 crossbar to an 8 x 8 crossbar.

Table 6.1: Comparison of BBN multiprocessor CPU characteristics

Machine	CPU	Clock Speed	MIPS	MFLOPS
GP1000	M68020	16 Mhz	2.5	0.6
TC2000	M88100	20 Mhz	19	20

The faster CPU in the TC2000 necessitates a faster switch with increased path width, and an 8 x 8 crossbar switch component solves this problem. One impact of the increased size of the crossbar switch

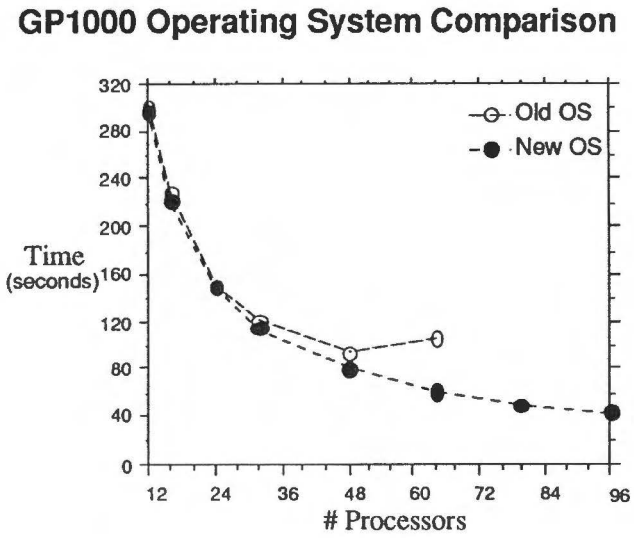

Figure 6.13: Comparison of old OS vs. new OS for mountain image, rectangular region UD

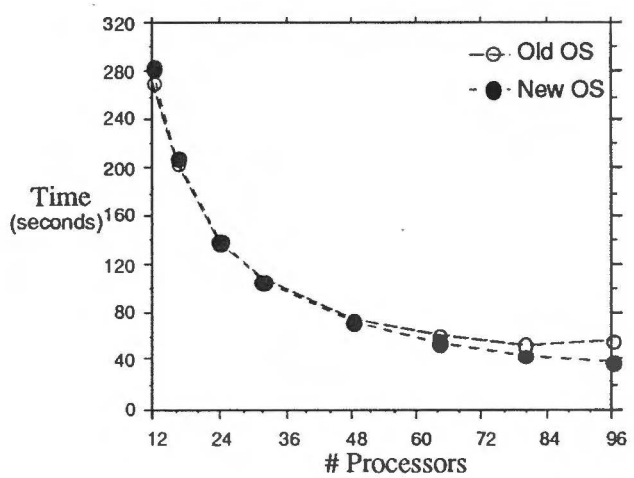

Figure 6.14: Comparison of old OS vs. new OS for mountain image, rectangular region LC

is that fewer wires are needed between the switch columns in the interconnection network. The 8 x 8 crossbar is more costly to produce than the 4 x 4 but it does allow 8 simultaneous messages to be output, whereas a 4 x 4 only supports 4 messages at a time.

Table 6.2: Comparison of BBN multiprocessor memory characteristics

Machine	Cache	Memory per Board	Switch Speed	Path Width	Basic Switch Node
GP1000	no	4 meg	8 Mhz	4 bit	4 x 4
TC2000	yes	4 or 16 meg	38 Mhz	8 bit	8 x 8

From the programmer's point of view, the TC2000 is functionally the same as the GP1000. There are several small differences regarding communication, however. The GP1000 supports the block transfer mechanism in hardware, whereby a path is held open long enough for 256 byte length messages to go from one board to another. In the TC2000, this operation is supported through software emulation rather than hardware implementation. The TC2000 does contain a memory cache which allows data to be allocated as cachable or non-cachable. Although using the cache significantly enhances performance, judicious management of this memory is required by the programmer since no cache coherence scheme is supported. The primary goal here is to compare the different algorithms under different CPU and switch characteristics, so the algorithms were not modified to take advantage of the cache.

The results, including times for the setup phase from the front end plus the tiling time, are shown in figures 6.15, 6.16, 6.17, and 6.18. A thorough analysis of the scan line algorithm was deemed unnecessary on the TC2000 due to its performance limitations noticed on the GP1000. It is, however, included for comparison purposes in the next section of this chapter.

These graphs indicate similar performance in the algorithms when compared to the previous graphs for the GP1000. The only problem with this comparison is that the results on the TC2000 were limited for most of the tests to a maximum of 48 processors, while with the GP1000, 96 processors were consistently available.[2]

[2]We have included some data obtained on the TC2000 at 96 processors in table 6.3. In general, though, due to the other users on the machine, only 48 processors were used for most of the tests.

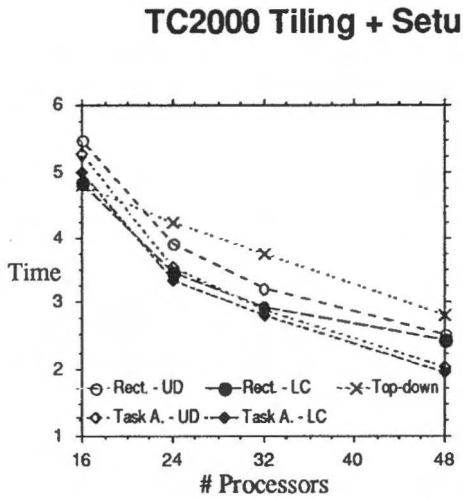

Figure 6.15: TC2000 algorithm comparison, stegosaurus image

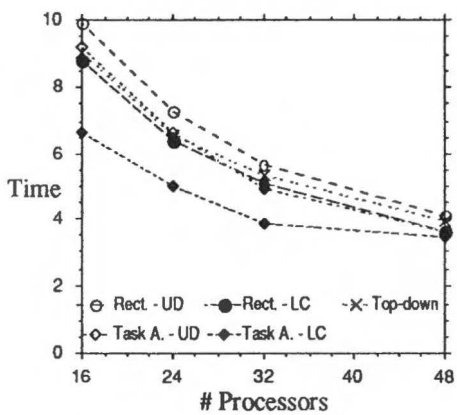

Figure 6.16: TC2000 algorithm comparison, Laser image

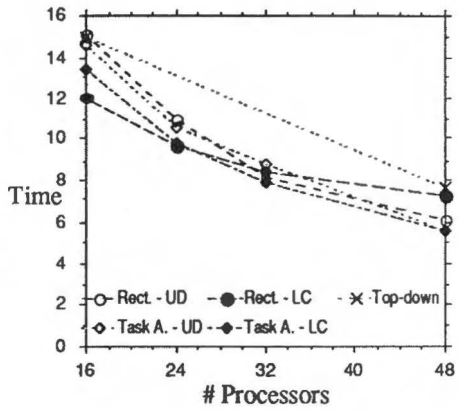

Figure 6.17: TC2000 algorithm comparison, tree image

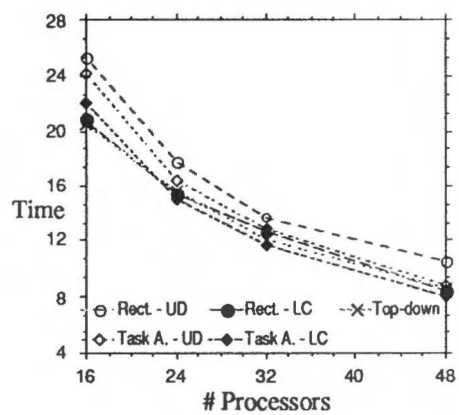

Figure 6.18: TC2000 algorithm comparison, mountain image

In order to allow a fair comparison between the two machines, the speedup was computed for each of the algorithms at 96 processors. The task adaptive algorithm is used for this comparison, and the results are shown in table 6.3.

Table 6.3: GP1000 and TC2000 speedup and time ratio comparison using 96 processors

Machine	Stegosaurus	Laser	Tree	Mountain
GP1000 Speedup	70.3	73.3	68.1	82.1
TC2000 Speedup	61.3	59.3	56.4	70.6
Ratio of Execution Times at $P = 96$: GP1000/TC2000	8.6	8.7	8.0	8.8

As can be seen from the table, the TC2000 exhibits slightly reduced speedup when compared to the GP1000 on 96 processors for most of the images. This could be caused by a number of factors, ranging from the amount of work per task to the processor-to-switch speed ratio. The last row in the table indicates the ratio of parallel execution times of the TC2000 divided by the GP1000. From this data, it appears that on 96 processors, the TC2000 is approximately 8.5 times faster than the GP1000 for this problem.

6.2.3. Relationship of Machine Parameters to Performance

In this section, we evaluate the various overheads on both machines to see their differences. The comparison involves examining the total processor-time space and comparing the results on the two machines. Here, the overheads are evaluated with respect to P and comparison values are shown to the right of each graph for the overhead percentages at 48 processors. Also, the speedup is given at each processor configuration. All of the algorithms are compared on the GP1000 and the TC2000 for the Laser image as a representative example. Due to the volume of data and the CPU time involved in the tests, only one image was used for comparison. Different results would be obtained for the different test images, but the main interest

here was to evaluate the trend in performance and directly compare the percentages on various processor configurations.

6.2.3.1. Comparison of Overheads

The next five pages provide a direct comparison of the overhead factors for all of the algorithms. The graphs include the total processor-time space for each particular processor configuration, with the overheads clearly marked as a percentage. Although the results were measured up to 96 processors for the GP1000, the overhead values given on the right side of the graph are for 48 processors so they can be compared to the values for the TC2000 below.

6.2.3.2. Analysis

These results present a number of interesting phenomena not noticed in any previous graphs. In the parallel scan line algorithm, the latency and code modification overheads constitute almost the same overhead percentage regardless of the processor configuration on both machines. This makes sense since the number of tasks is constant regardless of the number of processors in this algorithm. In the other cases, since the total number of tasks increases with the number of processors, the overhead effects increase as well. In some cases, the load balancing may go down at some point but this may be due to an increase in another factor as explained next.

With the exception of the task adaptive algorithm, the load balancing is better on the TC2000 than in the GP1000. On the other hand, the network contention, code modification, and latency/communication are significantly worse. It seems that the increased delay due to communication overheads and contention contribute to even out the load in the algorithms on the TC2000 (recall that load balancing cannot be measured independently from other factors). Since these overheads are larger in the TC2000 than in the GP1000, they contribute to an increase in the average task execution time. This changes the load balancing since it is based on dynamic scheduling of the tasks, as well as their execution time.

In the case of the task adaptive algorithm, the load balancing is a direct result of dynamic task partitioning, and it is possible that the tasks cannot be partitioned near the end of the computation due to the imposed threshold. This effect may be more pronounced in the TC2000 than in the GP1000 due to the difference in the synchronization and communication mechanisms.

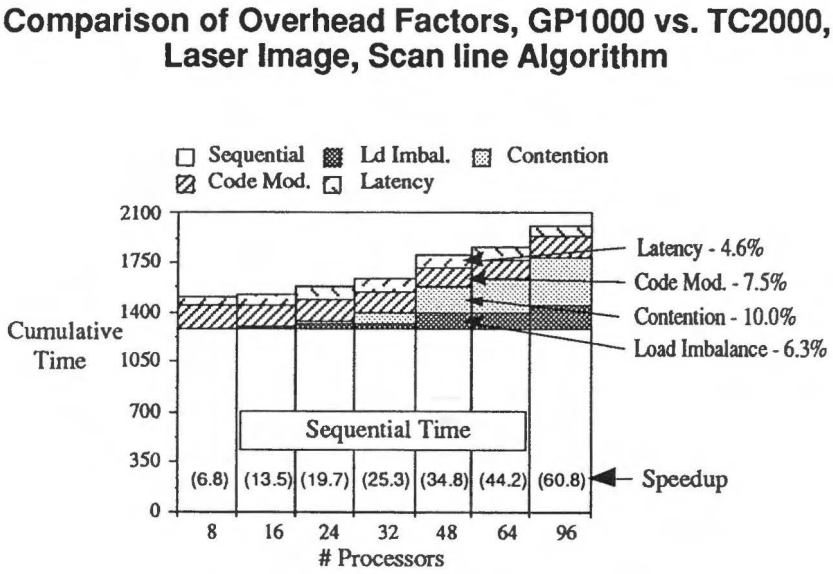

Figure 6.19: GP1000, scan line algorithm, UD, overhead comparison

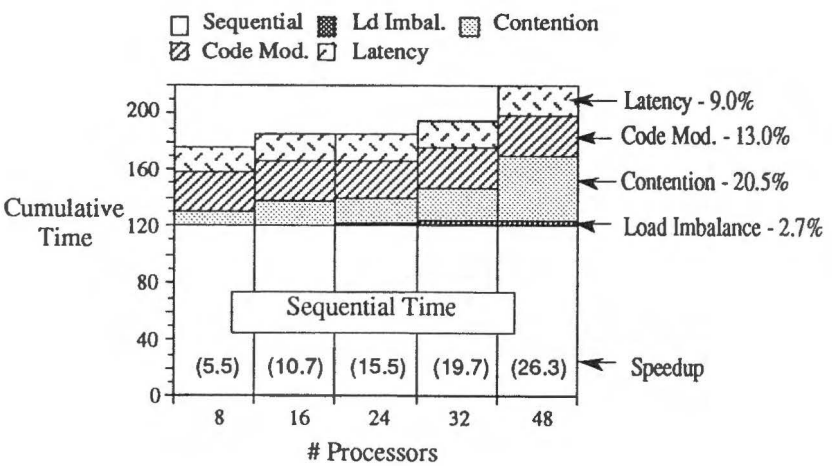

Figure 6.20: TC2000, scan line algorithm, UD, overhead comparison

Comparison of Overhead Factors, GP1000 vs. TC2000, Laser Image, Rectangular Region Algorithm, UD Scheme

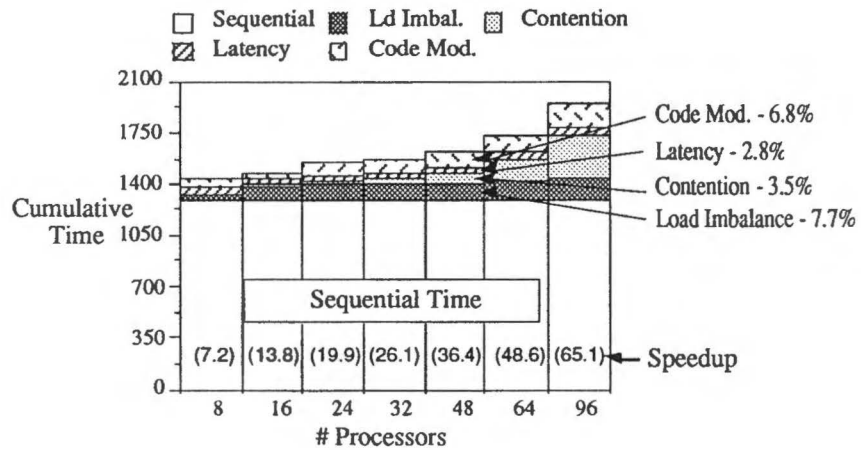

Figure 6.21: GP1000, rectangular region algorithm, UD, overhead comparison

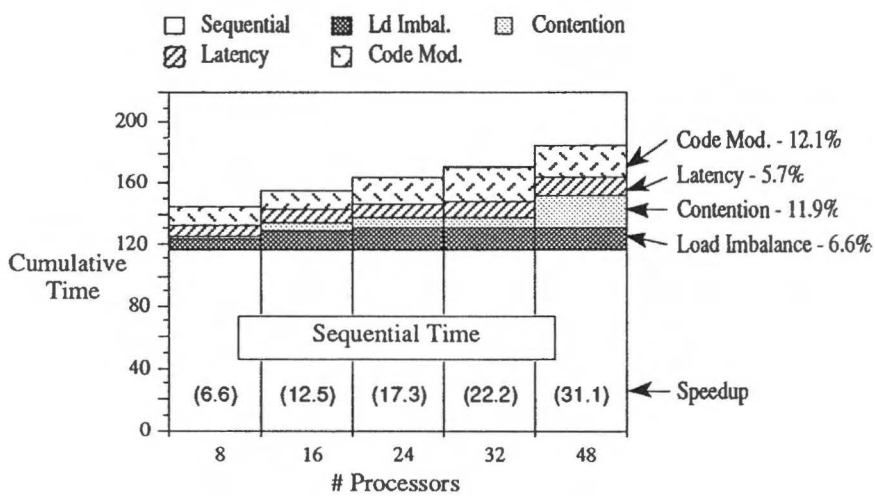

Figure 6.22: TC2000, rectangular region algorithm, UD, overhead comparison

Comparison of Overhead Factors, GP1000 vs. TC2000, Laser Image, Rectangular Region Algorithm, LC Scheme

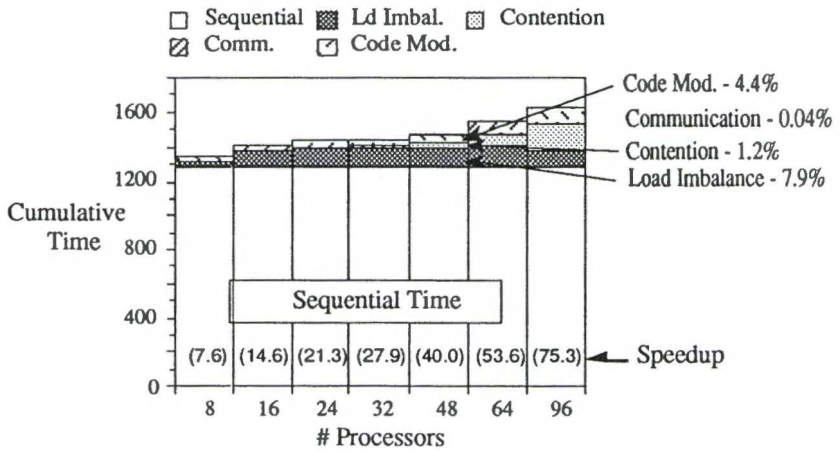

Figure 6.23: GP1000, rectangular region algorithm, LC, overhead comparison

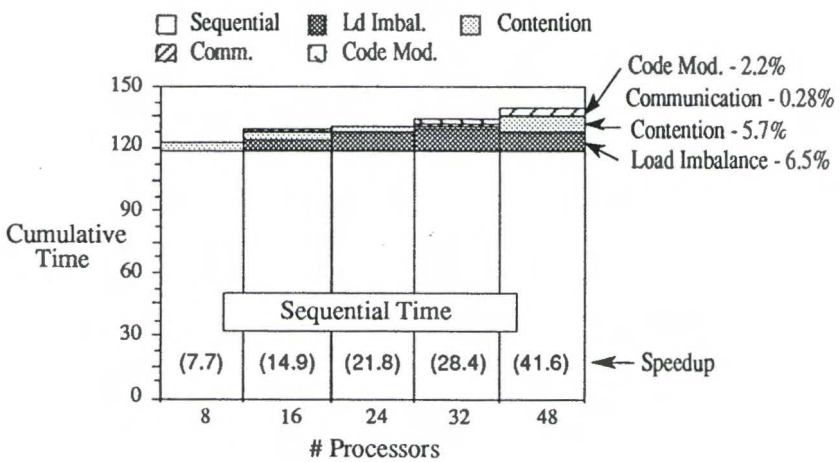

Figure 6.24: TC2000, rectangular region algorithm, LC, overhead comparison

Comparison of Overhead Factors, GP1000 vs. TC2000, Laser Image, Top-Down Algorithm

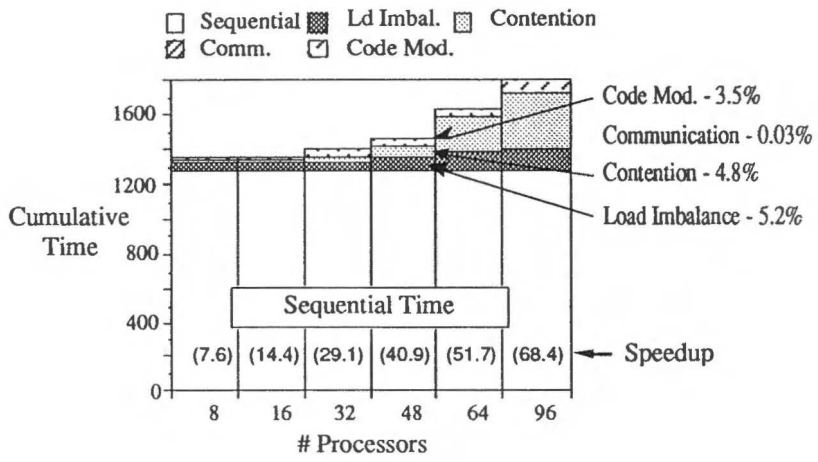

Figure 6.25: GP1000, top-down algorithm, LC, overhead comparison

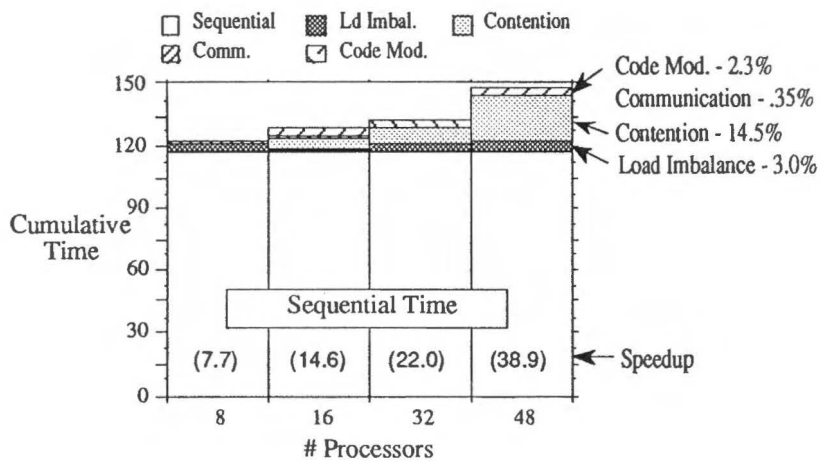

Figure 6.26: TC2000, top-down algorithm, LC, overhead comparison

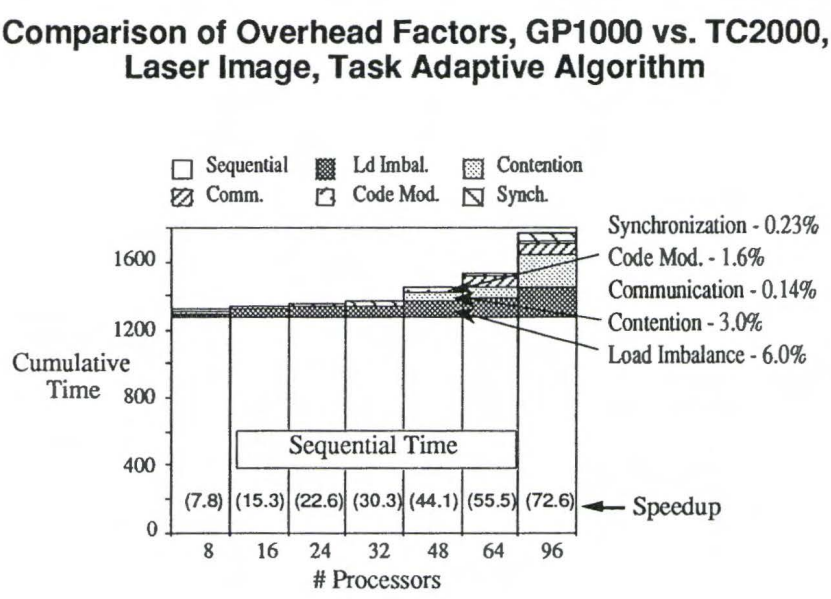

Figure 6.27: GP1000, task adaptive algorithm, LC, overhead comparison

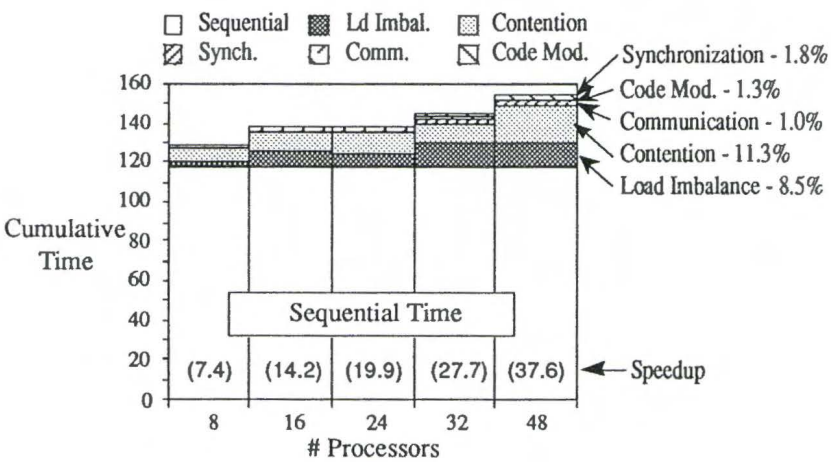

Figure 6.28: TC2000, task adaptive algorithm, LC, overhead comparison

The probable cause for the general network contention increase in the TC2000 over the GP1000 could be attributed to these factors:

1. The increase in switch speed in the TC2000 over the GP1000 does not match the corresponding increase in processor speed, therefore more collisions in the network switch are likely to occur.
2. The hardware support for block transfers in the GP1000 is not available in the TC2000 (this was used in the LC scheme).

It seems likely that the cause for the general network contention increase could be a combination of both of these reasons, especially for the LC scheme which extensively uses the block transfer mechanism. Although the TC2000 contains a cache as well as support for use of fine grained interleaved shared memory, neither of these characteristics is used in the implementation of the algorithms. The cache does not affect performance in the LC schemes, however, since the remote data is copied to local memory, where it is then cached as local data. In addition, the fine grain interleaving is not needed and would not provide better performance anyway since the shared data is already scattered among the memory modules. In the next section, the effect of enhancing the characteristics of computer graphics scenes is analyzed to determine the overall performance differential.

6.3. Scene Characteristics

One of the reasons different images are used for performance comparisons throughout this book is that it is desirable to be able to generalize these results to apply to all computer generated scenes. Of course this is an impossible task since there are always pathological cases one cannot predict. In the experiments four scenes were used which have different characteristics in screen area projection, number of data elements, and depth complexity. In this section, these same four scenes are analyzed, along with several new ones which have added scene complexity in one form or another. These break down into two categories: image complexity and object complexity .

In his thesis, Whelan analyzes several scenes which vary in complexity in terms of both of the above categories. His conclusions merely represent what most researchers intuitively realize, but they are worth repeating here:

1. Scenes are not usually composed of uniformly distributed polygons.

2. As the number of polygons increases, their size generally decreases. Somescenes may have a few large polygons which take up a significant portion of the drawn area on the screen.
3. Most polygons have few edges.
4. The aspect ratio of polygons is non-uniform, although some scenes seem to be oriented towards a particular direction.
5. The depth complexity of most pixels is fairly small (less than six), although some scenes obviously violate this rule.

It is not necessary to repeat Whelan's analysis for the scenes used here since it does not categorize scenes in terms of their difficulty in the various rendering stages. His analysis does point out that non-uniformity in scenes is the norm, so a parallel graphics algorithm must take this into account and perform well under various circumstances. The algorithms presented in this book are general purpose and are designed to handle various input scenes rather than a specific type. One cannot categorically draw a relationship between a given algorithm and say, the depth complexity of an image. Even if this were the case, does that information provide anything useful to the user community? In general, the algorithm should perform well on all imagery and should have the capability of handling pathological cases with some efficiency. This is more useful than algorithm analysis based upon depth complexity, polygon area coverage, or some other factor.

In the following sections, the different algorithms' performance is compared using an increase in image complexity in the first subsection and an increase in object complexity in the second subsection.

6.3.1. Image Complexity

Image complexity refers to the addition of features to a scene to make a higher quality image. Such additional features can include: rendering at higher resolution, advanced anti-aliasing, texture mapping, shadow generation, and bump mapping to name a few. Texturing, shadowing, or bump mapping, have not been implemented here since these features require careful planning in order to be implemented efficiently in parallel. This is primarily due to the additional memory required for each of these features, plus the desire to avoid contention for this memory. Anti-aliasing has already been incorporated into this algorithm, and all of the data presented so far includes this feature. Therefore, increasing spatial resolution is con-

centrated on here, and a comparison of the other features is left for future work.

All of the previous scenes are re-tested at double resolution (1280 x 968) to evaluate the performance of the different algorithms under these conditions. The first set of graphs involves a comparison similar to the others in this chapter, in which the setup phase as well as the tiling time is taken into account. The speedup and efficiency are given in the second set of graphs.

As a representative example of the times for the high resolution computations, the results for the mountain image are shown in figures 6.29 and 6.30, but the graphs for all the images are included in the appendix in figures A.21 through A.28. These graphs are zoomed in to show more detail at the higher processor counts. The speedup for the mountain image on the GP1000 using the task adaptive algorithm is shown in figure 6.31, and the efficiency for this image in figure 6.32. Since communication is the same as before but there is an increase in work due to the increased resolution, the algorithms are more efficient. Table 6.4 shows the speedup and efficiency for each of the images when calculated at high resolution on the GP1000, versus normal resolution using the task adaptive (LC) algorithm.

Table 6.4: Tiling section comparison of speedup and efficiency for normal resolution images vs. high resolution images on GP1000, 96 Processors

Images (#polygons)	Normal Resolution Speedup	High Resolution Speedup	Normal Resolution Efficiency	High Resolution Efficiency
Stegosaurus (9K)	71.4	86.5	0.74	0.90
Laser (46K)	73.6	80.0	0.77	0.83
Tree (106K)	70.5	84.6	0.73	0.88
Mountain (131K)	82.2	87.6	0.86	0.91

The data from the table indicates that the speedup varies widely among the images, but the single common result is that the additional work in the high resolution images provides better speedup and efficiency than in the normal resolution case. It is likely that the reason for the improved speedup is that the ratio of work to communication time has increased, thus reducing the network contention percentage.

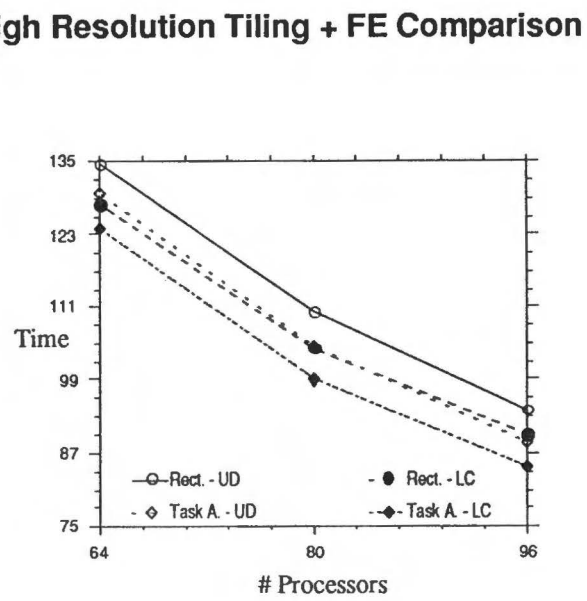

Figure 6.29: Rectangular vs. task adaptive on GP1000, mountain image, high-res

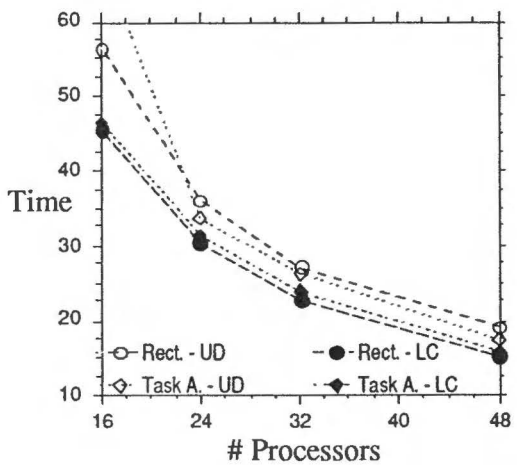

Figure 6.30: Rectangular vs. task adaptive on TC2000, mountain image, high-res

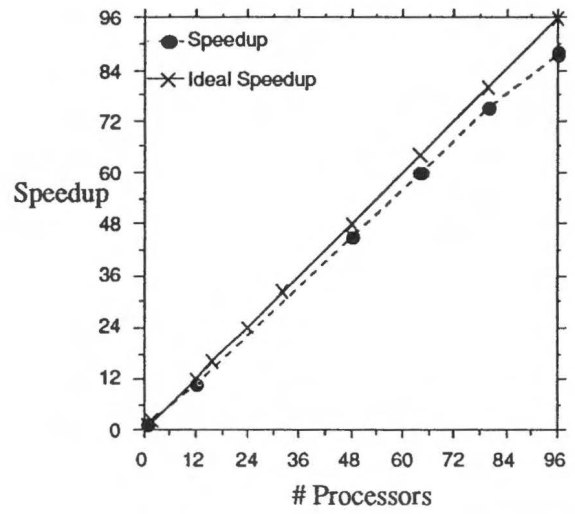

Figure 6.31: Speedup for high-res mountain image, GP1000

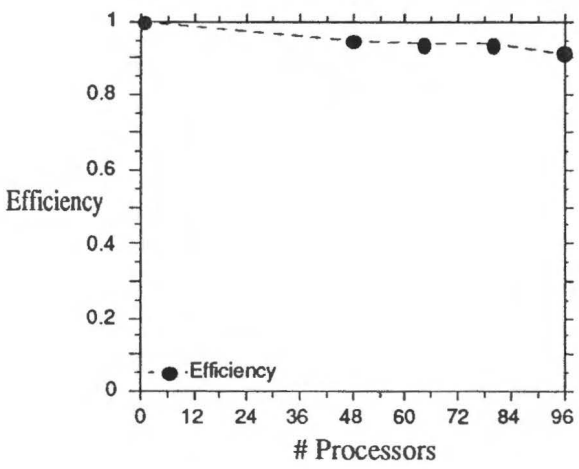

Figure 6.32: Efficiency for high-res mountain image, GP1000

This is logical since the amount of network traffic has not changed, but the amount of work has increased due to the increase in spatial resolution. In the next section, the algorithms are compared on a different set of images which involve much higher numbers of polygons to see the expected times on these datasets.

6.3.2. Object Complexity

One of the goals of this work was to present solutions for developing a highly efficient parallel rendering algorithm which allows extremely fast computation of complex imagery. To this end, new datasets are evaluated which contain a considerably greater number of polygons. Examples of two highly complex datasets are the rings image in color plate 5 and the dense tree image shown in color plate 6. The dense tree image has more polygons than the previously evaluated tree image, particularly in the twigs. Due to the amount of memory required for the tree dataset, it is not possible to evaluate speedup and efficiency since not enough physical memory was available even locally. All of the images are evaluated based on their total time, as well as the number of polygons rendered per second. Hardware manufacturers typically quote a figure of polygons per second in evaluating hardware Z-buffer graphics workstations. A typical example of this type of machine is the HP-320/SRX. Some of the images which were evaluated previously have been rendered on this machine by Eric Haines, who developed the SPD database from which the tree, mountain, and rings datasets were extracted. Table 6.5 shows a comparison of all the images and the effective number of polygons per second achieved. These results were obtained using the task adaptive algorithm on 96 processors of a BBN TC2000.

In comparison to the values in the table, Haines ran some of the same tests on the HP-320/SRX [Hain87b]. In rendering the dense tree, 4835 polygons/second was achieved. Using a denser version of the rings image (874 K polygons) than employed here, the HP-320/SRX achieved 4819 polygons/second. For comparison, a current example of a state of the art graphics superworkstation is the Silicon Graphics Iris 4D VGX [Haeb90]. The manufacturer quotes a figure of 750,000 Gouraud-shaded triangles per second for this machine.

This value for the hardware performance is based on Gouraud shaded polygons which are not anti-aliased. Our data is for Phong shaded polygons with specular highlights and stochastic sampled anti-aliasing. Although it is difficult to compare exactly, the addition of Phong shading typically might cost 3 times as much as Gouraud shading, in addition to the anti-aliasing cost which is about 4 times as

Table 6.5: Effective rendering rate and speedup using 96 processors on BBN TC2000, task adaptive algorithm

Images	# Polygons	Time (sec.)	Polygons/Second	Speedup
Stegosaurus	**9,639**	**1.14**	**8,455**	**61.3**
Laser	**46,393**	**2.02**	**22,967**	**59.3**
Tree	**106,289**	**2.94**	**32,907**	**56.4**
Mountain	**131,072**	**4.12**	**29,857**	**70.6**
Rings	**567,841**	**9.90**	**52,239**	**85.6**
Dense Tree	**851,288**	**8.81**	**80,690**	**58.9**

much for stochastic sampling as it is for a Z-buffer. Also, the amount of physical memory required to support over 500,000 polygons is most likely not available within a graphics workstation, and this will almost certainly slow down the hardware. Nevertheless, the results given here are significantly faster than a slightly older generation graphics workstation and might compare favorably with a current generation machine. When the fact that the rendering is done in software rather than hardware in this algorithm is added, the benefit is that much greater since the software version allows much more flexibility in its use. This is elaborated upon in the next chapter.

6.4. Conclusions

Based on all of the data reported in this chapter, it seems clear that the task adaptive algorithm utilizing the LC memory referencing scheme provides the best performance for all types of imagery among the methods implemented. Besides performance, other advantages of the task adaptive approach are:

1. It is unnecessary to determine an initial optimal granularity ratio for this algorithm. The number of areas chosen initially corresponds to the number of processors in the system, and high performance is achieved regardless of machine configuration. In the other algorithms, this ratio must be derived for each image independently for the best performance.

2. The load balancing in the task adaptive approach is completely dynamic, based on the amount of work left. Because of this, worst case scenarios, such as all of the data being located in one portion of the screen, can be handled effectively while the other algorithms will not perform nearly as well in this type of situation.

3. This approach has minimal time cost in the setup phase of the front end since the number of areas is very small initially. Hence, this time is much less than in the other algorithms.

In summary, the first section of this chapter presents an analysis of several different memory referencing strategies. Based on this theoretical analysis, the Uniformly Distributed and Locally Cached schemes are shown to only differ by several tenths of a second. A description regarding the implementation of both of these schemes is then presented in detail. After comparing the results of the implementations, it is clear that the LC scheme combined with the task adaptive decomposition method results in the best performance for all the test imagery. The setup time from the front end for each approach is included for the timings, in order to allow a fair comparison and substantiate this fact. The theoretical analysis of the memory reference strategies indicated that there would only be a small difference between the UD and LC schemes. Instead, the difference is much larger due to the fact that network contention is not accounted for in the theoretical analysis. This contention is an important degradation factor since it is significantly smaller in the LC scheme than it is in the UD scheme.

The second section involves a comparison of the different parallel algorithms with respect to changes in the underlying machine parameters. It is shown that a change in the GP1000 operating system which allows parallel page faults markedly improved performance over the previous version of this operating system. When using the next generation version of the BBN multiprocessor known as the TC2000, the faster CPU and network switch improves performance in all cases almost an order of magnitude over the GP1000. The task adaptive algorithm using the LC scheme proves to be the best performer for this new machine as well. A comparison of overhead factors between the two machines reveals that network contention plays a more significant role in degrading performance in the TC2000 than in the GP1000 when measured at 48 processors, however. A possible explanation is that the interconnection network speed increase from the GP1000 to the TC2000 does not match the

corresponding increase in the CPU performance. As a result, tasks execute faster, but the communication of data is more frequent, leading to a greater possibility of a blocked path in the network. The TC2000 offers enhancements to memory referencing (such as a hardware cache) that are not incorporated into the implemented algorithms. It is possible that if these are utilized, the effects of network contention could be reduced.

The third section in this chapter involves an analysis of the effect of increasing image and object complexity in the test scenes. Re-evaluating the performance of the algorithms at a resolution of 1280 x 968 on the mountain image reveals a speedup of 87.6 on 96 processors using the task adaptive algorithm on the GP1000. In addition, using a highly complex scene with over 800,000 data elements, an effective polygon rendering rate of over 80,000 anti-aliased Phong shaded polygons per second is achieved on 96 processors of a TC2000. This is most likely the fastest rendering ever realized to this point using a software algorithm on a general purpose MIMD architecture for graphics rendering.

7

Conclusion

This book has primarily concentrated on the development and analysis of various approaches to tiling three-dimensional computer generated scenes on a multiprocessor. In doing so we have presented the following:

1. A categorization of possible parallel approaches to graphics rendering into a taxonomy according to graphical task decomposition.
2. A number of methods which incorporate parallelism in all aspects of a graphics rendering program.
3. A quantitative analysis of various degradation factors encountered in a multiprocessor graphics display algorithm implementation.
4. The development of general task partitioning and memory referencing strategies which may be used in other graphics rendering algorithms, as well as non-graphics applications.

These will be described in detail in the following sections.

7.1. Summary

In the past, research in the area of parallel graphics rendering has concentrated primarily on approaches to tiling a scene. This portion of the program involves the hidden surface removal and subsequent smooth shading operations necessary to establish a realistic rendering. Although this is the most time consuming portion of a graphics display program, little work has been spent on the development of parallel approaches to the front end and back end of such a program. If this is not done, the advantage gained in parallelizing the tiling portion will be lost when the other parts are executed sequentially. Although the front end and back end portions of the programs presented here were not analyzed fully, a pipelined approach was developed t speed up significantly these segments of the algorithms. In addition, these phases are well integrated with the tiling portion of the programs, thus providing a general purpose high performance approach to parallel rendering which could be used in real-world applications. This means that the parallel programs presented here will provide faster speed than other programs developed in the past which do not incorporate parallelism into the non-tiling phases.

The results in chapter 6 indicate that the task adaptive algorithm maintains the highest performance of the image space algorithms which were implemented. By splitting tasks into areas dynamically, the maximum amount of coherence is maintained in this approach. The setup time in the front end is small since the number of areas created initially is reduced in comparison to the other approaches. This is due to the fact that the initial number of areas chosen is equal to the number of processors currently available in the system. The added advantage here is that it is unnecessary to find an optimal granularity ratio prior to tiling the scene, as opposed to the other algorithmic approaches. Another advantage of this scheme is that worst case situations, such as a high concentration of data in a small portion of the image, are handled elegantly and will not present a problem. This algorithm, when combined with the Locally Cached memory referencing scheme, offers the best overall performance on the tested datasets. In addition, based on the timings shown in both chapter 5 and chapter 6, the algorithm performs well all the way up to 96 processors on both Butterfly multiprocessors. An efficiency of over 90% was obtained with the mountain image on 96 processors on the GP1000. In addition, using the rings database which contains over 500,000 polygons, an efficiency of 84% was obtained on the TC2000 computer. This indicates that parallel processing of

computer graphics rendering is a cost effective solution for use with very complex datasets.

Based on the data presented in this book, it seems that the less complicated approaches to parallel decomposition obtain the highest performance. Many researchers have challenged this notion by developing complicated solutions to the parallel rendering problem. While this may be necessary in a hardware environment, it usually compromises performance in the software environment due to the extra overheads of synchronization, etc. The simple techniques presented here for dynamic task decomposition, along with judicious memory management schemes, combine to solve the problem in a straightforward manner.

One common misconception that might be perceived regarding parallel display algorithms is that when graphical coherence is lost, performance will suffer greatly. Indeed, coherence has played a significant role in enhancing the development of serial display algorithms. In a parallel context, however, even when a large number (2,304) of areas is created, the overhead is limited to only about 6% of the total execution time. Clearly, an approach which uses groups of small scan line areas reduces the overhead due to coherence. The loss due to lack of coherence is not as great as one might think, so parallel processing of areas does not add significant overhead to the serial approaches developed in the past.

In general, other factors play a more important role than coherence in the performance of the parallel algorithms. Most of the degradation relates to either load balancing or communication. One of the important facts brought out here is that an algorithm which strictly emphasizes load balancing does not guarantee the best performance. This is shown in the times for the data adaptive algorithm. This algorithm does exhibit the least amount of overhead due to load imbalance; however, the implementation of this approach forces additional overhead to be incurred in other parts of the algorithm that negate the gains achieved by a balanced load.

Memory usage also plays an important role in a number of ways. The Uniformly Distributed (UD) memory referencing scheme introduces latency, while the Locally Cached (LC) memory referencing scheme requires communication using block transfers. As a result of the large volume of message traffic due to retrieving data in the UD scheme, a significant amount of network contention is introduced. The LC scheme minimizes this factor, but it still plays an important role in the degradation in performance, especially on large processor configurations. Network contention is difficult to predict since it is related to the number of requests for paths in the

interconnection network at a given time. When the interval between requests is large, the contention is small, and vice versa. This explains why an analytical model that does not relate network contention to communication in the system will not accurately predict performance. Although researchers have developed a good model [Nand90] for computation in this type of environment, it is difficult to use in the context of a complex algorithm such as this one. The task adaptive algorithm uses larger tasks which require communication less frequently than the other approaches, except toward the end of the computation. Contention is not reduced in this approach, though, since the issue of how to limit the burst of communication at the end of the computation due to the large amount of task splitting has not been fully addressed.

The Butterfly interconnect is an example of a high speed network, but contention still plays an important role in the algorithms shown here. This should be taken into account in the implementation of any program involving a large amount of data movement. The problem of network contention can be resolved if the usage of the network is reduced or more switch paths are made available in the hardware. The latter is not usually a solution available to the programmer, so the former approach must be taken. This was used in the LC memory referencing strategy which was developed solely for the purpose of limiting communication in the system. Further refinements of this scheme could be added to reduce the number and size of the messages, particularly in the case of the task adaptive algorithm.

The following generalizations can be made regarding implementations on parallel machines of algorithms with high data movement, such as the ones developed here. The memory referencing strategy directly affects performance in the system since it is directly related to the communication of data. The frequency and amount of communication determines the overall degradation due to network contention. Any reduction of this factor is certain to provide high performance, especially in applications where there is a large dataset requiring frequent referencing. Another point is that global scattering of data among the memory modules in a shared memory NUMA machine is necessary to counteract the problem of hot spot contention. Using some type of caching scheme to bring in data to the local memory module prior to referencing is crucial to high performance since it minimizes network traffic and reduces latency. Finally, the coarsest granularity for tasks which allows adequate load balancing is the best approach to take in order to achieve good parallel timings. The graphs of granularity ratio versus overhead effects (figures A.5, A.6, A.7, and A.8) show that other overhead

factors are increased when the granularity becomes too fine. As an example, the task adaptive approach uses the smallest number of tasks to minimize these effects in comparison to the other decompositions.

From the results presented herein, it can be seen that high performance can be achieved in a graphics display algorithm on a parallel architecture. Since graphics applications tend to be data intensive, the memory must be managed effectively, something which has not been done in some parallel graphics algorithms developed in the past. Other graphics applications such as volume rendering, image processing, and radiosity can take advantage of some of the techniques described here for use in a parallel environment. For instance, the LC memory referencing scheme could be modified for usage in each of these instances since the data to be rendered for a particular task is known a priori. In addition, the task adaptive algorithm could also be modified for task partitioning purposes for these rendering techniques. In addition, non-graphics applications can also utilize the techniques described here. Example applications might include geographical information systems, global climate modeling, finite element simulation, and applied graph theory.

During the development of this project, many problems were encountered when implementing the parallel algorithms on the Butterfly. Several software tools available with this machine made program development much easier than it otherwise might have been. *Gist* is a performance analysis package which shows a graph of processors versus time in an X-window display. By setting events at critical time points in the program, one can evaluate the performance of the program by looking at how long a particular phase takes to execute on each processor. The graphical output of this tool aids in the programmer's understanding of the processor-time graph. A parallel profiler is also available which can generate individual processor profiles and this is also a useful tool for evaluating program performance. The primary software tool that allows greater understanding of the internal nature of the parallel aspect of the programs is a parallel debugger called *TotalView*. Without this tool, situations like race conditions, synchronization problems, and shared memory problems would have been much more difficult to debug. This environment made program debugging, testing, and analysis an easy interactive task which hastened the development of the evaluated programs.

Even with the fact that BBN is no longer building the Butterfly, scalable shared memory multiprocessors are not dead. Clearly, latency and contention issues are the primary target areas for

improvement. Tera Computer Company is designing a machine to combat both of these issues in a scalable shared memory architecture. Message passing machines seem to be growing in popularity, and it would be useful to determine how these types of machines could be used for fast graphics rendering. While the memory referencing strategies of the algorithms presented here might need to be modified for this type of machine, the work decomposition methods would not require modification. It is possible that a number of different decomposition strategies will lead to good results due to the different topologies and communication performance in these architectures. In addition, message passing architectures have distributed memory like the Butterfly so an extension of the LC memory referencing scheme might prove to be a viable solution in that environment.

The analysis and performance results given in this book should serve as a guide to the reader regarding the critical issues involved in the development of a parallel graphics rendering program. In addition, the algorithmic possibilities which are worthwhile taking into account during the development and tuning process have been presented and can certainly be modified according to the machine available to the programmer.

7.2. Future Work

There are still a number of unanswered questions which have not been addressed in this book, in addition to new questions brought out by the results reported here. Next, these are elaborated upon with regard to potential future areas of work.

In relation to machine parameters, it would be interesting to see what new issues are encountered on a very large multiprocessor containing 512 or an even greater number of processors. It is entirely possible that the algorithms will need to be changed to handle this type of situation. As the speed of microprocessors has increased over the years, the interconnection network speed has not increased as quickly. While the TC2000 network is faster than the GP1000 network, it seemed to incur greater network contention. This is most likely due to the fact that the performance of the interconnection network was not increased at the same rate that the processor speed was increased. It would be interesting to analyze this phenomenon in detail, not just for graphics algorithms, but for other applications as well. This might provide insight for hardware as well as software designers when planning a program for parallel implementation.

Another machine characteristic specifically relevant to the TC2000 is the use of the hardware cache as well as interleaved shared

memory. The cache provides fast access to read-only shared data which is stored globally. On TC2000 machines which contain 16 megabytes of memory per processor, it is possible to configure a portion of the memory on each processor as fine-grained interleaved. The programmer no longer has to make sure that the data is scattered across the memory modules since it is implicitly done in this case. In addition, the granularity of interleaving is much finer than was possible before, so that even a one-dimensional array can be scattered among the memory modules. The combination of using the cache with storing read-only data in the interleaved shared memory might produce better results than the LC scheme developed in this book, but the cachable data would need to be managed effectively. This would alleviate the programmer from the burden of programming the complex code necessary to implement the LC scheme.

Other researchers have investigated techniques by which the operating system manages the storage and access of shared memory in an NUMA architecture such as the Butterfly [Laro90]. This level of management by the operating system allows use of a memory referencing method such as the UD scheme which is easy to develop, while the operating system manages the location and storage of data and attempts to optimize it for high program performance.

Other future work in developing parallel graphics display algorithms might look at different implementations of memory referencing schemes, investigate methods to alleviate communication, and look at the handling of even larger graphics datasets. Graphics features such as shadows, texture maps, bump maps, and motion blur also present interesting challenges to a parallel implementation. Each of these features requires additional memory, and the use of the memory in some cases cannot be localized. For instance, texture mapping requires referencing a two-dimensional array of pixel values to assign a more detailed property to a particular polygon or object. If the map is scattered throughout memory, hot spot contention will be minimized, but memory latency will occur. A caching technique like the LC scheme could be used to bring in the portion of the map that is relevant for a particular polygon or object. This method might require a very large portion of the texture, and there may not be enough local processor memory to facilitate this type of approach. Another enhancement which might be worth investigating in the future involves fast update of small scenes in real-time. In fact, this is the type of work that is currently being investigated for scientific visualization at Lawrence Livermore National Lab. We hope to report on the success of this project in the future.

References

[ABRA86] Abram, G. *Parallel Image Generation with Anti-Aliasing and Texturing*, Ph.D. dissertation, University of North Carolina at Chapel Hill, 1986.

[AHUJ86] Ahuja, S.; Carriero, N.; and Gelernter, D. "Linda and Friends." *IEEE Computer 19*, 8 (August 1986) pp. 26-34.

[ALLI91] Allison, M. Private communication, July, 1991.

[AMDA67] Amdahl, G. "Validity of the Single-Processor Approach to Achieving Large-Scale Computer Capabilities." *AFIPS Conference Proceedings 30*(1967) pp. 483-485.

[APGA88] Apgar, B.; Bersack, B.; and Mammen, A. "A Display System for the Stellar Graphics Supercomputer Model GS1000." *Computer Graphics, Proceedings of Siggraph 22*, 4 (August 1988) pp. 255-262.

[ARVI86] Arvind and Ianucci, R.A. "Two Fundamental Issues in Multiprocessing. "Tech. Rept. 226-4, MIT Laboratory of Computer Science, Cambridge, Massachusetts, March, 1986.

[BADO90] Badouel, D.; Bouatouch, K.; and Priol, T. "Ray Tracing on Distributed Memory Parallel Computers: Strategies for Distributing Computations and Data." *Course Notes for Course 28, Siggraph* (1990) pp. 185-198.

[BAUM90] Baum, D.R. and Winget, J.M. "Real Time Radiosity Through Parallel Processing and Hardware Acceleration," *Computer Graphics 24*, 2, (March 1990) pp. 67-75.

[BBN84] BBN Laboratories, Inc. *Development of a Butterfly Multiprocessor Test Bed, The Butterfly Switch*, July, 1984, Report No. 5874.

[BBN89A] BBN Advanced Computers, Inc. *Programming in C with the Uniform System*, 1989.

[BBN89B] BBN Advanced Computers, Inc. *Butterfly GP1000 Switch Tutorial*, March, 1989.

[BERM87] Berman, F. and Snyder, L. "On Mapping Parallel Algorithms into Parallel Architectures." *Journal of Parallel and Distributed Computing 4*(1987).

[BLIN77] Blinn, J.F. "Models of Light Reflection for Computer Synthesized Pictures." *Computer Graphics, Proceedings of Siggraph 11*(1977) pp. 192-198.

[BURK90] Burke, A. and Leler, W. "Parallelism and Graphics: An Introduction and Annotated Bibliography." *Course Notes for Siggraph Course 28, ACM Siggraph Conference* (1990) pp. 111-140.

[CARP84] Carpenter, L.C. "The A-Buffer, an Anti-Aliased Hidden Surface Method." *Computer Graphics, Proceedings of Siggraph 18*, 3 (1984) pp. 103-108.

[CASP89] Caspary, E. and Scherson, I.D. "A self-balanced parallel ray-tracing algorithm," in *Parallel Processing for Computer Vision and Display*, P.M. Dew, T.R. Heywood, and R.A. Earnshaw, editors, Addison-Wesley, 1989, pp. 408-419.

[CATM74] Catmull, E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. dissertation, NTIS A004 968, University of Utah, December 1974.

[CHAL91] Challinger, J. "Parallel Volume Rendering on a Shared Memory Multiprocessor." UCSC-CRL-91-23, University of California, Santa Cruz, 1991.

[CHAN81] Chang, P. and Jain, R. "A Multi-Processor System for Hidden Surface Removal." *Computer Graphics 15*, 4 (December 1981) pp.405-436.

[CHEN88] Chen, M.C. "Mapping Parallel Algorithms onto General-Purpose Parallel Machines." *Proceedings of the 21st Annual Hawaii International Conference on System Sciences 1*(1988).

[CLEA83] Cleary, J.G;, Wyvill, B.M.; Birtwistle, G.M;, and Vatti, R. "Multiprocessor Ray Tracing." Tech. Rept. 83/128/17, University of Calgary, 1983.

[COOK82] Cook, R.L. and Torrance, K.E. "A Reflectance Model for Computer Graphics." *ACM Transactions on Graphics 1*(1982) pp. 7-24.

[COOK86] Cook, R.L. "Stochastic Sampling in Computer Graphics." *ACM Transactions on Graphics* (January 1986).

[COOP87] Cooper, E.C. and Draves, R.P. "C Threads." Internal Research note of the Mach Project, Carnegie Mellon University, April, 1987.

[CROC91] Crockett, T.W. and Orloff, T. "A Parallel Rendering Algorithm for MIMD Architectures," ICASE Tech. Rept. No. 91-3, NASA Langley Research Center, June, 1991.

[CROW77] Crow, F.C. "Shadow Algorithms for Computer Graphics." *Computer Graphics, Proceedings of Siggraph 11*, 3 (July 1977) pp. 242-248.

[CROW88A] Crow, F.C. "Parallelism in Rendering Algorithms." *Proceedings of Graphics Interface* (June 1988).

[CROW88B] Crow, F.C; Demos, G.; Hardy, J.; McLaughlin, J.; and Sims, K. "3D Image Synthesis on the Connection Machine." *Proceedings of the International Conference on Parallel Processing for Computer Vision and Display* (January 1988), Leeds, UK.

[DEER88] Deering, M.; Winner, S.; Schediwy, B.; Duffy, C.; and Hunt, N. "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics." *Computer Graphics, Proceedings of Siggraph 22*, 4 (August 1988).

[DIED88] Diede, T.; Hagenmaier, C.F.; Miranker, G.S.; Rubinstein, J.; and Worley, W.S. Jr. "The Titan Graphics Supercomputer Architecture." *IEEE Computer* (September 1988).

[DYER87] Dyer, S. and Whitman, S. "A Vectorized Scan line Z-Buffer Rendering Algorithm." *IEEE Computer Graphics & Applications 7*, 7 (July 1987) pp. 34-45.

[FIUM83] Fiume, E.; Fournier, A.; and Rudolph, L. "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General Purpose Ultracomputer." *Computer Graphics, Proceedings of Siggraph 17*, 3 (July 1983) pp. 141-149.

[FRAN80] Franklin, W.R. "A Linear Time Exact Hidden Surface Algorithm." *Computer Graphics, Proceedings of Siggraph 14*, 3 (1980) pp. 117-123.

[FRAN90] Franklin, W.R. and Kankanhalli, M.S. "Parallel Object-Space Hidden Surface Removal." *Computer Graphics, Proceedings of Siggraph 24*, 4 (August 1990) pp. 87-94.

[FUCH79] Fuchs, H. and Johnson, B.W. "An Expandable Multiprocessor Architecture for Video Graphics." *Proceedings of the 6th Annual ACM-IEEE Symposium on Computer Architecture* (1979) pp. 58-67.

[FUCH85] Fuchs, H.; Goldfeather, J.; Hultquist, J.P; Spach S.; Austin, J.D.; Brooks, F.P. Jr.; Eyles, J.G.; and Poulton, J. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes." *Computer Graphics, Proceedings of Siggraph 19*, 3 (July 1985) pp. 111-120.

[FUCH89] Fuchs, H.; Poulton, J.; Eyles, J.; Greer, T.; Goldfeather, J.; Ellsworth, D.; Molnar, S.; Turk, G.; Tebbs, B.; and Israel, L. "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories." *Computer Graphics, Proceedings of Siggraph 23*, 3 (July 1989) pp. 79-88.

[GARL90] Garlick, B.J.; Winget, J.M.; and Baum, D.R. "Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations," *Parallel Algorithms and Architectures for 3D Image Generation, Siggraph Course 28*, August, 1990.

[GHAR88] Gharachorloo, N.; Gupta, S.; Hokenek, E.; Balasubramanian, P.; Bogholtz, B.; Mathieu, C.; and Zoulas, C. "Subnanosecond Pixel Rendering with Million Transistor Chips." *Computer Graphics, Proceedings of Siggraph 22*, 4 (August 1988).

[GHOS86] Ghosal, D. and Patnaik, L.M. "Parallel Polygon Scan Conversion Algorithms: Performance Evaluation of a Shared Bus Architecture." *Computers & Graphics 10*, 1 (1986) pp. 7-25.

[GORD91] Gorda, B.; Warren, K.; and Brooks, E.D. III "Programming in PCP," Tech. Rept. UCRL-MA-107029, Lawrence Livermore National Laboratory, April, 1991.

[GOTT83] Gottlieb, A.; Grishman, R.; Kruskal, C.P.; McAuliffe, K.P.; Rudolph, L.; and Snir, M. "The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer." *IEEE Transactions on Computers C-32*, 2 (February 1983) pp. 175-189.

[GOUR71] Gouraud, H. "Continuous Shading of Curved Surfaces." *IEEE Transactions on Computers C-20* (June 1971) pp. 623-629.

[GREE89] Green, S. and Paddon, D. "Exploiting Coherence for Multiprocessor Ray Tracing." *IEEE Computer Graphics and Applications 9*, 6 (November 1989) pp. 12-26.

[HAEB90] Haeberli, P. and Akeley, K. "The Accumulation Buffer: Hardware Support for High-Quality Rendering." *Computer Graphics, Proceedings of Siggraph 24*, 4 (August 1990) pp. 309-318.

[HAIN87A] Haines, E. "A Proposal for Standard Graphics Environments." *IEEE Computer Graphics & Applications 7*, 11 (November 1987) pp. 3-5.

[HAIN87B] Haines, E. *Standard Procedural Database Documentation,* Unpublished, 1987.

[HU85] Hu, M.-C. and Foley, J.D. "Parallel Processing Approaches to Hidden-Surface Removal in Image Space." *Computers & Graphics 9,* 3 (1985) pp. 303-317.

[JAMI87] Jamieson, L.H. *Characteristics of Parallel Algorithms,* MIT Press (1987) pp. 65-100.

[KAPL79] Kaplan, M. and Greenberg, D.P. "Parallel Processing Techniques for Hidden Surface Removal." *Computer Graphics, Proceedings of Siggraph 13,* 2 (July 1979) pp. 300-307.

[KIRK90] Kirk, D. and Voorhies, D. "The Rendering Architecture of the DN10000VS." *Computer Graphics, Proceedings of Siggraph 24,* 4 (August 1990) pp. 299-307.

[KUCK86] Kuck, D.J.; Davidson, E.S.; Lawrie, D.H.; and Sameh, A.H. "Parallel Supercomputing Today and the Cedar Approach." *Science 231*(February 1986) pp. 967-974.

[KUMA86] Kumar, M. and Pfister, G.F. "The Onset of Hot Spot Contention." *Proceedings of the 1986 International Conference on Parallel Processing* (1986) pp. 28-34.

[LARO90] LaRowe, R.P. Jr. and Ellis, C.S. "Experimental Comparison of Memory Management Policies for NUMA Multiprocessors." Tech. Rept. CS-1990-10, Duke University, Durham, NC, April, 1990.

[MYER75] Myers, A. J. "An Efficient Visible Surface Program." Tech. Rept. The Ohio State University Computer Graphics Research Group, July, 1975.

[NAND90] Nanda, A.K; Shing, H.; Tzen, T.-H.; and Ni, L.M. "A Replicate Workload Framework to Study Performance Degradation in Shared-Memory Multiprocessors." *Proceedings of the International Conference on Parallel Processing* (1990) pp. I-161 to I-168.

[NITZ91] Nitzburg, B. and Lo, V. "Distributed Shared Memory: A Survey of Issues and Algorithms," *Computer 24,* 8 (August 1991), pp. 52-60.

[NUGE88] Nugent, S.F. "The iPSC/2 Direct-Connect Communications Technology." *Proceedings of the Third Hypercube Conference, ACM* (1988).

[PARE88] Parent, R.E. and Klasky, R.S. "Using Object Clusters for Efficient Calculation of Computer Generated Images." Tech. Rept. OSU-CISRC-TR39, The Ohio State University, Columbus, Ohio, December, 1988.

[PARK80] Parke, F.I. "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems." *Computer Graphics, Proceedings of Siggraph 14*, 3 (July 1980) pp. 48-56.

[PHON75] Phong, B.T. "Illumination for Computer Generated Pictures." *Communications of the ACM 18*, 6 (June 1975) pp. 311-317.

[PLUN85] Plunkett, D.J. and Bailey, M.J. "The Vectorization of a Ray-Tracing Algorithm for Improved Execution Speed." *IEEE Computer Graphics & Applications 5*, 8 (August 1985) pp. 52-60.

[POTM89] Potmesil, M. and Hoffert, E.M. "The Pixel Machine: A Parallel Image Computer." *Computer Graphics, Proceedings of Siggraph 23*, 3 (July 1989) pp. 69-78.

[RAO89] Rao, V.N. and Kumar, V. "Parallel Depth First Search." *International Journal of Parallel Programming 16*, 6 (1989).

[ROBL88] Roble, D.R. "A Load Balanced Parallel Scan line Z-Buffer Algorithm for the iPSC Hypercube." In *Proceedings of Pixim '88*, Paris, France, October 1988.

[ROGE85] Rogers, D. *Procedural Elements for Computer Graphics*, McGraw-Hill (1985).

[SADA87] Sadayappan, P. and Ercal, F. "Nearest Neighbor Mapping of Finite Element Graphs onto Processor Meshes." *IEEE Transactions on Computers 36*, 12 (December 1987).

[SADA88] Sadayappan, P. and Visvanathan, V. "Circuit Simulation on Shared-Memory Multiprocessors." *IEEE Transactions on Computers 37*, 12 (December 1988).

[SAMI89] Samiotakis, I. *A Thread Library for a Non Uniform Memory Access Multiprocessor*, Master's thesis, The Ohio State University, Columbus, Ohio, 1989.

[SCHR91] Schröder, P.and Salem, J.B. "Fast Rotation of Volume Data on Data Parallel Architectures," Tech. Rept. TMC-195, Thinking Machines Corporation. (Also available in *Course Notes for Advance Volume Visualization Course, Siggraph* (1991) as well as *Visualization '91 Proceedings*, San Diego, CA, Oct. 1991.)

[SLAT90] Slater, A.E. *Parallel Processing in Computer Graphics*, Master's thesis, University of Massachusetts, 1990.

[SMAL89] Smallbone, J. "Programming high performance graphics on the DAP," in *Parallel Processing for Computer Vision and Display*, P.M. Dew, T.R. Heywood, and R.A. Earnshaw, editors, Addison-Wesley, 1989, pp. 321-328.

[SUTH74] Sutherland, I.E.; Sproull, R. F;, and Schumacker, R.A. "A Characterization of Ten Hidden-Surface Removal Algorithms." *Computing Surveys 6*, 1 (March 1974).

[SUTH75] Sutherland, I. *System of Polygon Sorting By Dissection,* U.S. Patent 3,889,107, 1975.

[THEO86] Theoharis, T.A. "Exploiting Parallelism in the Graphics Pipeline." Tech. Rept. PRG-54, Oxford University Computing Laboratory, June, 1986.

[THEO89a] Theoharis, T. and Page, I. "Parallel incremental polygon rendering on a SIMD processor array," in *Parallel Processing for Computer Vision and Display*, P.M. Dew, T.R. Heywood, and R.A. Earnshaw, editors, Addison-Wesley, 1989, pp. 329-337.

[THEO89b] Theoharis, T. *Algorithms for Parallel Polygon Rendering*, Springer-Verlag (1989).

[UPSO89] Upson, C. and Fangmeier, S. "The role of visualization and parallelism in a heterogeneous supercomputing environment," in *Parallel Processing for Computer Vision and Display*, P.M. Dew, T.R. Heywood, and R.A. Earnshaw, editors, Addison-Wesley, 1989, pp. 286-297.

[WARN69] Warnock, J.E. "A Hidden-Surface Algorithm for Computer Generated Half-tone Pictures." Tech. Rept. TR 4-15, University of Utah, NTIS AD-753 671, June, 1969.

[WATK70] Watkins, G.S. A Real-Time Visible Surface Algorithm. Tech. Rept. UTEC-CSc-70-101, Univeristy of Utah, NTIS AD-762 004, June, 1970.

[WEIL77] Weiler, K. and Atherton, P. "Hidden Surface Removal Using Polygon Area Sorting." *Computer Graphics, Proceedings of Siggraph 11*, 2 (1977) pp. 214-222.

[WHEL85] Whelan, D.S. *Animac: A Multiprocessor Architecture for Real-Time Computer Animation*, Ph.D. dissertation, California Institute of Technology, 1985.

[WHIT80] Whitted, T. "An Improved Illumination Model for Shaded Display." *Communications of the ACM 23*(1980) pp. 343-349.

[WHIT88] Whitman, S. and Parent, R. "A Survey of Parallel Hidden Surface Removal Algorithms." *Proceedings of Pixim '88* (October 1988), Paris, France.

[WHIT89] Whitman, S. and Guenter, B. "The Design of Image Space Graphics Display Algorithms for MIMD Architectures." *Course Notes for Siggraph Course 16, ACM Siggraph Conference* (1989) pp. 129-155.

[WHIT90] Whitman, S. "Computer Graphics Rendering on a Parallel Processor." *Course Notes for Course 28, Siggraph* (1990) pp. 167-183.

[WHIT91] Whitman, S. and Sadayappan, P. "Computer Graphics Rendering on a Shared Memory Multiprocessor," *Proceedings of the International Conference on Parallel Processing,* (August 1991) CRC Press, pp. 191-194.

[WILL78] Williams, L. "Casting Curved Shadows on Curved Surfaces." *Computer Graphics, Proceedings of Siggraph 12*, 3 (July 1978) pp. 270-274.

Appendix

A. Information on Test Scenes

The matrix used in Eric Haines' SPD database clipped out too much of the fractal mountain when it was generated at the resolution we chose (130 K polygons). Therefore, for comparison purposes, we provide the matrix we used here:

$$\begin{bmatrix} 1.2 & 0 & 0 & 0 \\ 0 & .297 & .742 & 0 \\ 0 & .649 & -.259 & 0 \\ 0 & .185 & 3.156 & 1 \end{bmatrix} \quad \text{(A.1)}$$

Also, the reader will note that the specified screen resolution which was proposed in the SPD database was 512 x 512, while our renderings were displayed at 640 x 484. The reason for this was that the algorithm used is intended as a scene development display algorithm for animation purposes and standard video resolution is 640 x 484. We expect that a resolution of 512 x 512 would result in somewhat similar timings.

B. Data for Various Algorithms

Each of the different implementations described in chapter 4 is presented in this section with the exact overhead percentages noted for each of the four test images. The algorithms are presented in the following order: scan line parallel, rectangular region (UD), rectangular region (LC), top-down, and task adaptive.

Scan line Data Non-Adaptive Algorithm

Scheduling	
stegosaurus	0.01%
laser	0.006%
tree	0.005%
mount	0.002%

Memory Latency

	#remote refs	* 6.47 µsec	% of *Tp*P*
steg	5,765,541	37.3 sec	3.0%
laser	12,813,378	82.9 sec	4.0%
tree	15,250,407	98.7 sec	4.1%
mount	47,629,192	308.2 sec	5.6%

Switch Contention - 48 x 48 mesh

	T(96)	*T*(1)	% *Tp*P*	Calculated
steg	1036.99	834.22	16.2%	23.1%
laser	1818.50	1522.42	14.4%	17.9%
tree	2158.57	1973.95	7.8%	8.9%
mount	5199.22	4034.73	21.0%	20.5%

Load Imbalance - average over 3 runs

steg	10.4%
laser	8.1%
tree	8.7%
mountain	6.8%

Code Modification

	Time	% of *Tp*P*
steg	106.3 sec	8.5%
laser	157.7 sec	7.7%
tree	117.9 sec	4.9%
mount	543.9 sec	9.8%

Rectangular Data Non-Adaptive Algorithm (UD Scheme)

Scheduling

96 processors, 2304 areas

stegosaurus	0.013%
laser	0.009%
tree	0.007%
mountain	0.004%

Memory Latency

	remote refs	* 6.47 µsec	% of *Tp*P*
steg	2,677,256	17.32 sec	1.4%
laser	7,834,453	50.68 sec	2.6%
tree	12,395,847	80.20 sec	3.4%
mount	23,160,041	149.85 sec	3.6%

Code Modification

	Time Difference	% Total Proc-Time Space
steg	120.8 sec	8.7%
laser	164.6 sec	8.0%
tree	240.5 sec	9.6%
mount	341.6 sec	7.9%

Load Imbalance - average over 3 runs

steg	7.0%
laser	6.4%
tree	11.5%
mount	4.3%

Switch Contention - 48 x 48 mesh

	$T(96)$	$T(1)$	% $Tp*P$	Calculated
steg	1087.16	828.81	18.7%	33.1%
laser	1800.05	1494.31	14.9%	20.6%
tree	2182.92	2074.59	4.3%	5.6%
mount	4053.71	3671.33	8.8%	10.9%

Rectangular Data Non-Adaptive Algorithm (LC Scheme)

Scheduling	2,304 areas
stegosaurus	0.017%
laser	0.01%
tree	0.007%
mountain	0.004%

Communication Overhead

	# bytes transferred	Time	% $Tp*P$
steg	953,446	0.268 sec	0.028%
laser	2,288,116	0.644 sec	0.039%
tree	3,634,812	1.022 sec	0.047%
mount	6,788,070	1.909 sec	0.052%

Load Imbalance - average over 3 runs

steg	6.8%
laser	6.6%
tree	11.1%
mount	4.5%

Code Modification

	Time Difference	% *Tp*P*
steg	62.3 sec	6.4%
laser	103.0 sec	6.3%
tree	119.6 sec	5.5%
mount	196.4 sec	5.4%

Switch Contention - 48 x 48 mesh

	T(96)	*T*(1)	% *Tp*P*	Calculated
steg	881.26	753.14	13.1%	16.3%
laser	1502.99	1385.45	7.2%	8.9%
tree	1940.51	1877.89	2.9%	3.1%
mount	3485.53	3380.91	2.9%	3.2%

Data Adaptive Top-Down Decomposition

Scheduling, 960 regions

stegosaurus	0.01%
laser	0.007%
tree	0.006%
mountain	0.003%

Communication Overhead

	# bytes transferred	Time	% *Tp*P*
steg	653,060	0.18 sec	0.02 %
laser	1,833,368	0.52 sec	0.03 %
tree	3,110,170	0.87 sec	0.04 %
mount	5,654,696	1.59 sec	0.04 %

Load Imbalance - average over 3 runs

steg	1.5%
laser	6.9%
tree	4.0%
mount	3.1%

Code Modification

	Time Difference	% *Tp*P*
steg	32.3 sec	2.8 %
laser	59.1 sec	3.3 %
tree	59.9 sec	2.6 %
mount	97.9 sec	2.5 %

Switch Contention - 48 x 48 mesh

	T(96)	*T*(1)	% *Tp*P*	Calculated
steg	1109.4	723.0	34.0 %	34.9 %
laser	1649.5	1341.4	17.4 %	17.3 %
tree	2177.9	1818.2	15.7 %	16.9 %
mount	3736.6	3282.1	11.8 %	11.8 %

Task Adaptive Algorithm

Number of Tasks

	non-background tasks	total tasks
steg	209	254
laser	183	235
tree	233	245
mount	224	243

Scheduling

stegosaurus	0.00023%
laser	0.00014%
tree	0.0001%
mountain	0.00006%

Communication Overhead

	# bytes transferred	Time	% *Tp*P*
steg	3,876,924	1.1 sec	0.11 %
laser	251,191,547	70.7 sec	4.20 %
tree	21,747,484	6.1 sec	0.25 %
mount	19,765,530	5.6 sec	0.15 %

Load Imbalance - average over 3 runs

steg	14.3%
laser	10.3%
tree	22.5%
mount	9.2%

Code Modification

	Time Difference	% *Tp*P*
steg	3.9 sec	0.4 %
laser	25.6 sec	1.6 %
tree	17.6 sec	0.7 %
mount	31.5 sec	0.8 %

Switch Contention - 48 x 48 mesh

	T(96)	*T*(1)	% *Tp*P*	Calculated
steg	798.7 sec	695.6 sec	10.6 %	11.7 %
laser	1489.5 sec	1309.7 sec	10.2 %	9.2 %
tree	1900.4 sec	1781.0 sec	4.8 %	5.6 %
mount	3433.8 sec	3219.7 sec	5.7 %	5.5 %

Synchronization

	Time	% *Tp*P*
steg	21.2 sec	2.2 %
laser	39.4 sec	2.3 %
tree	4.0 sec	0.16 %
mount	7.9 sec	0.21 %

C. Supplementary Graphs

Additional graphs are given here for the purpose of providing a better evaluation of the various algorithms. These graphs, listed in order of appearance, are the following:

1. Ratio Comparison for Rectangular Region Decomposition, UD Scheme
2. Ratio Comparison for Rectangular Region Decomposition, LC Scheme
3. Duplication Factors vs. Number of Areas
4. Comparison of Operating Systems for GP1000, Rectangular Region, UD Scheme
5. Comparison of Operating Systems for GP1000, Rectangular Region, LC Scheme
6. Comparison of All Algorithms, Total Time including Tiling and Front End, High resolution on the GP1000
7. Comparison of All Algorithms, Total Time including Tiling and Front End, High resolution on the TC2000

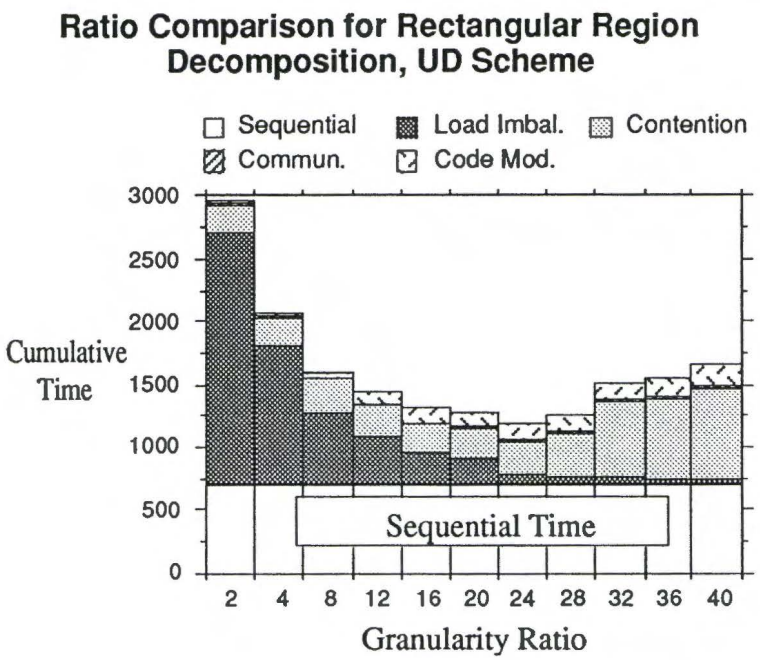

Figure A.1: Comparison of ratios for stegosaurus image, UD

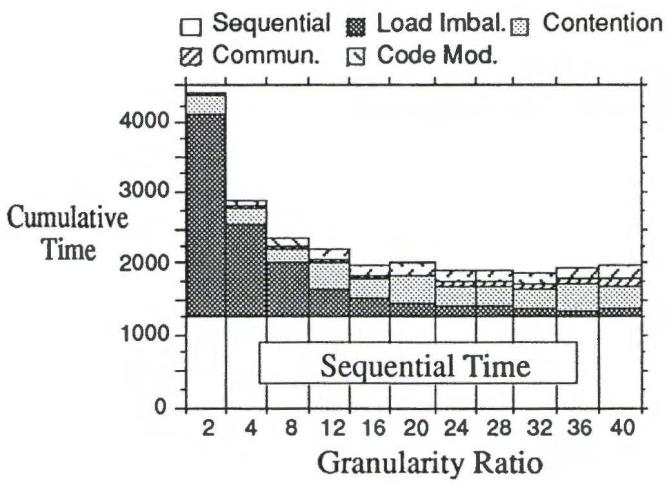

Figure A.2: Comparison of ratios for Laser image, UD

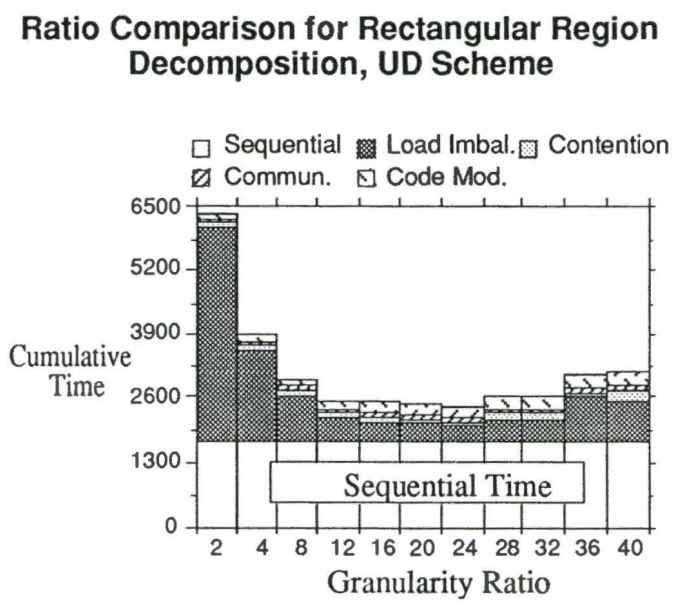

Figure A.3: Comparison of ratios for tree image, UD

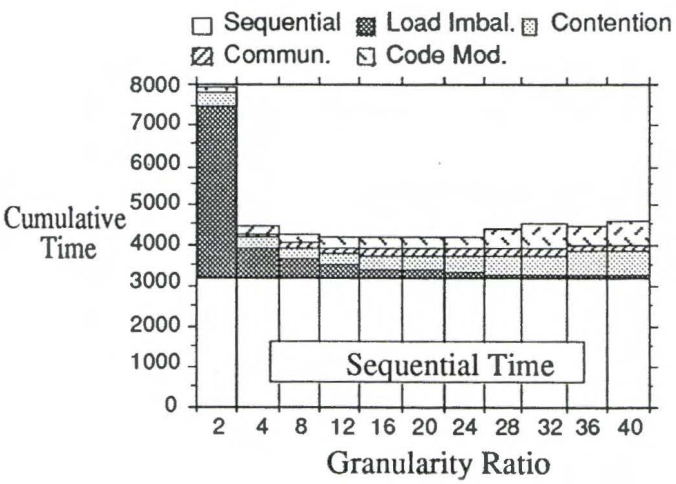

Figure A.4: Comparison of ratios for mountain image, UD

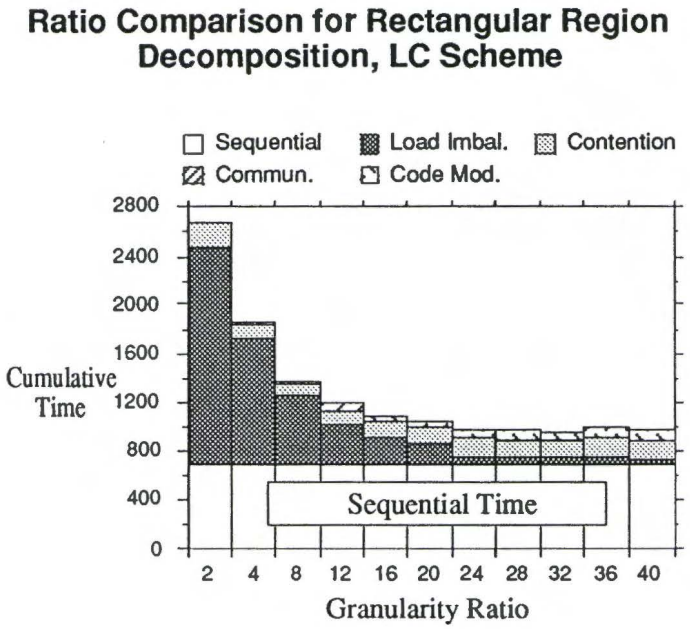

Figure A.5: Comparison of ratios for stegosaurus image, LC

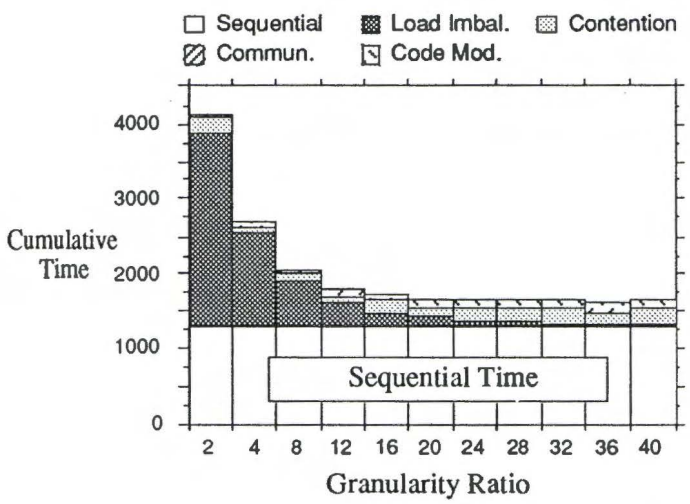

Figure A.6: Comparison of ratios for Laser image, LC

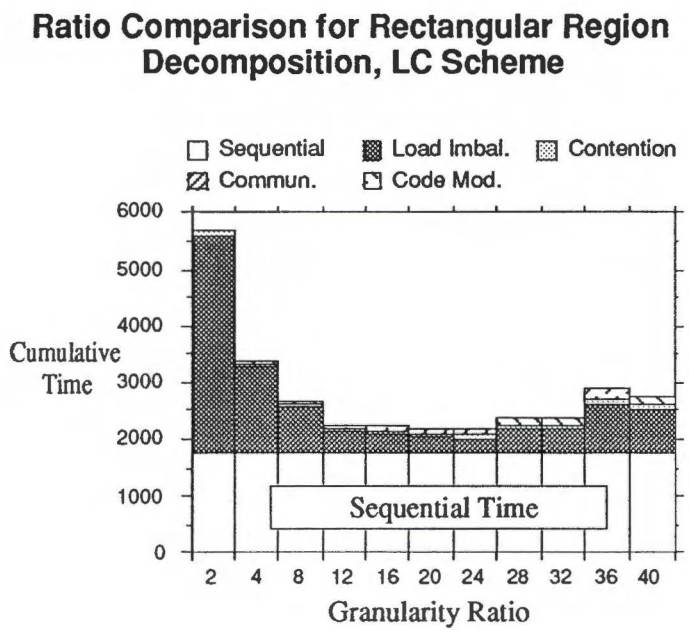

Figure A.7: Comparison of ratios for tree image, LC

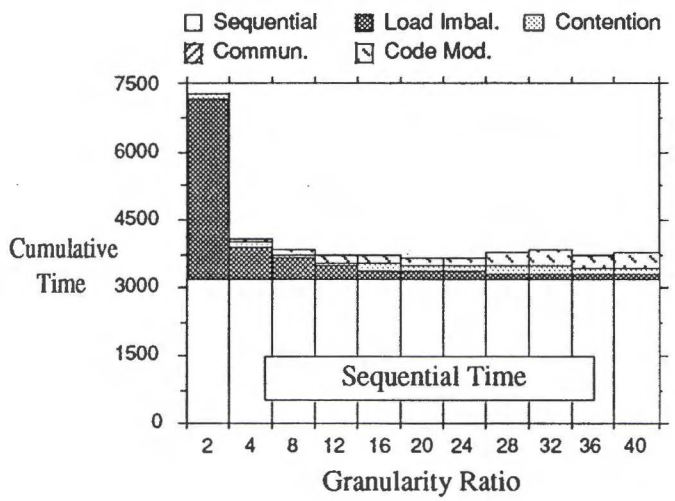

Figure A.8: Comparison of ratios for mountain image, LC

Duplication Factors vs. Number of Areas

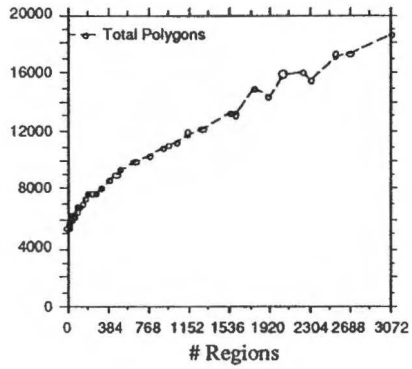

Figure A.9: Total polygons versus number of areas, stegosaurus image

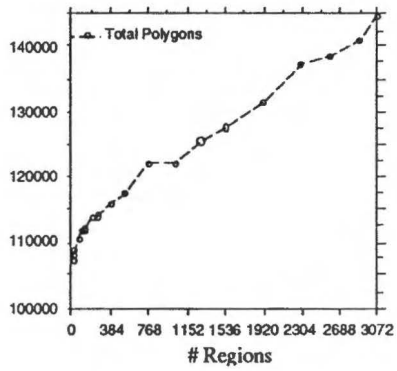

Figure A.11: Total polygons versus number of areas, tree image

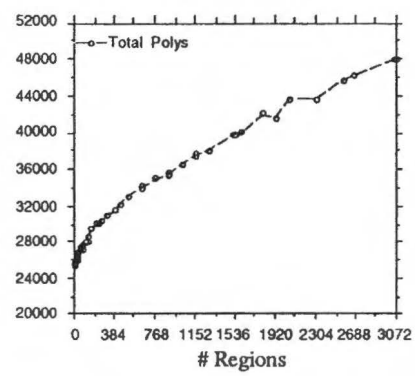

Figure A.10: Total polygons versus number of areas, Laser image

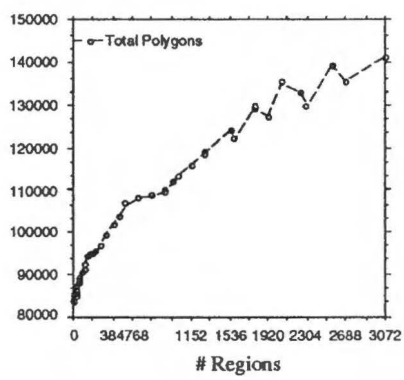

Figure A.12: Total polygons versus number of areas, mountain image

Comparison of Operating Systems for GP1000 Rectangular Region, UD Scheme

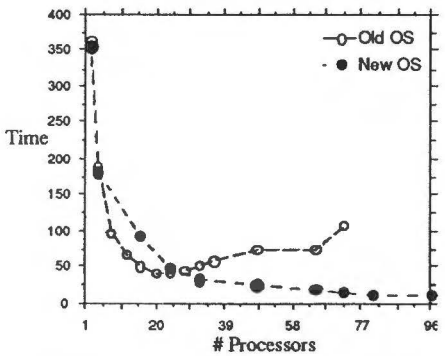

Figure A.13: Comparison of old OS vs. new OS for stegosaurus image, rectangular region UD

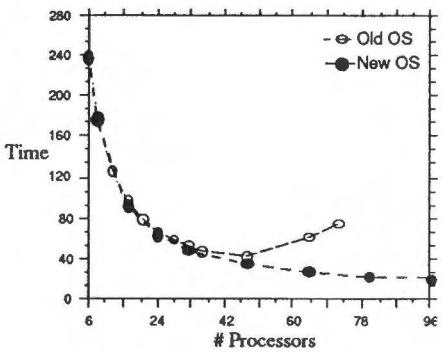

Figure A.15: Comparison of old OS vs. new OS for Laser image, rectangular region UD

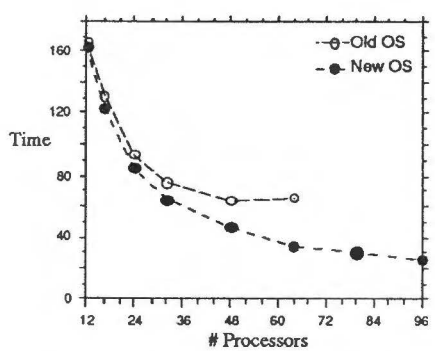

Figure A.14: Comparison of old OS vs. new OS for tree image, rectangular region UD

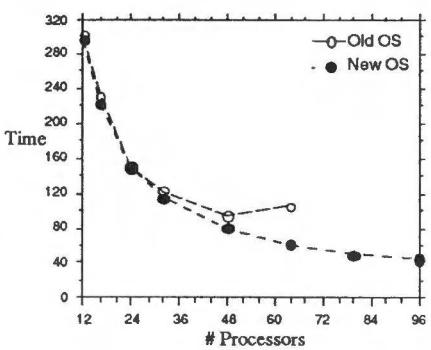

Figure A.16: Comparison of old OS vs. new OS for mountain image, rectangular region UD

Comparison of Operating Systems for GP1000, Rectangular Region, LC Scheme

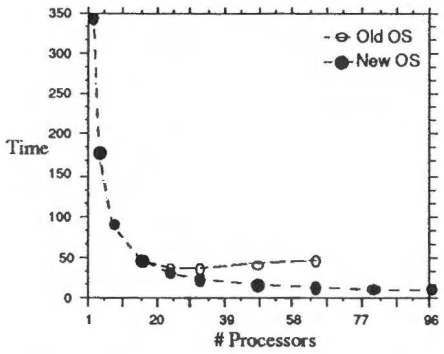

Figure A.17: Comparison of old OS vs. new OS for stegosaurus image, rectangular region LC

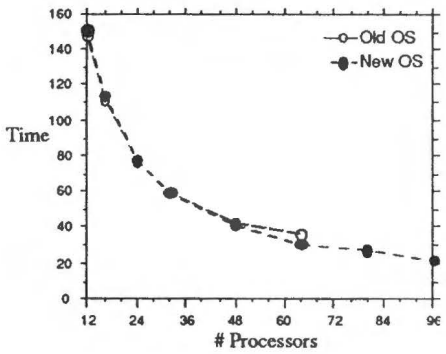

Figure A.19: Comparison of old OS vs. new OS for tree image, rectangular region LC

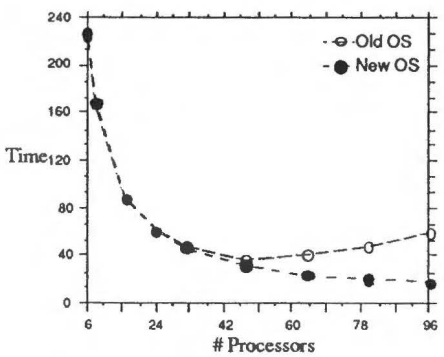

Figure A.18: Comparison of old OS vs. new OS for Laser image, rectangular region LC

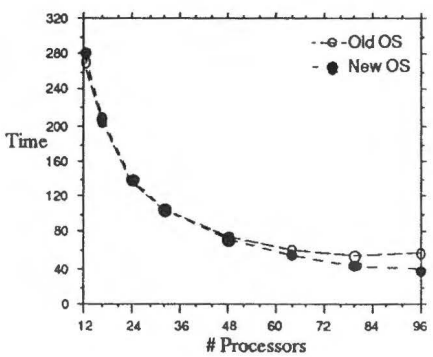

Figure A.20: Comparison of old OS vs. new OS for mountain image, rectangular region LC

Comparison of algorithms, total time including tiling + fe, high-res, GP1000

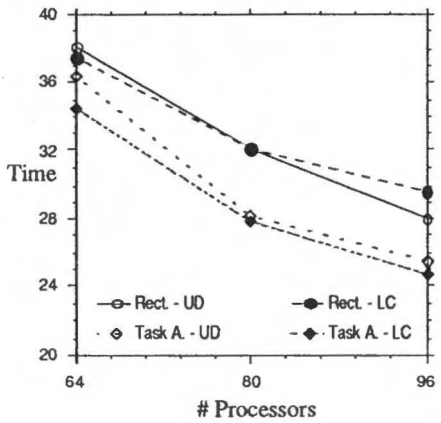

Figure A.21: Total time comparison, GP1000, stegosaurus image, hi-res

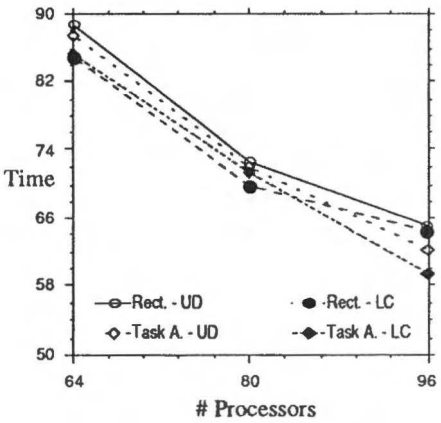

Figure A.23: Total time comparison, GP1000, tree image, hi-res

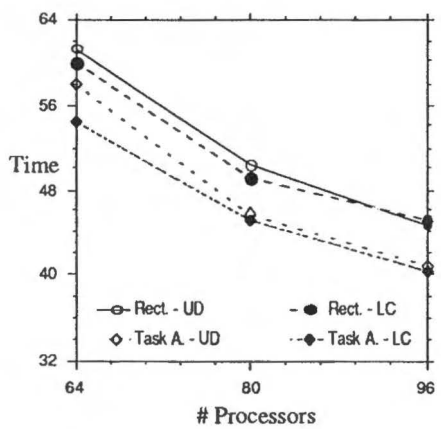

Figure A.22: Total time comparison, GP1000, Laser image, hi-res

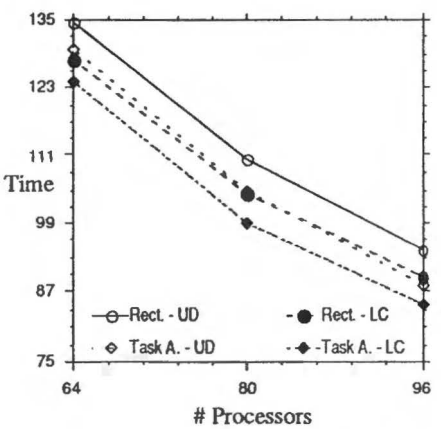

Figure A.24: Total time comparison, GP1000, mountain image, hi-res

Comparison of algorithms, total time including tiling + fe, high-res, TC2000

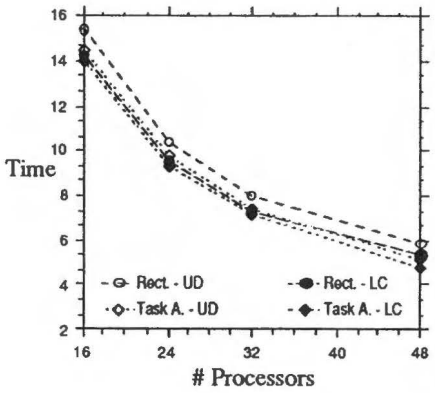

Figure A.25: Total time comparison, TC2000, stegosaurus image, hi-res

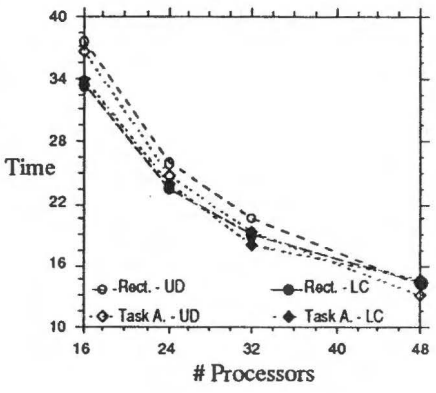

Figure A.27: Total time comparison, TC2000, tree image, hi-res

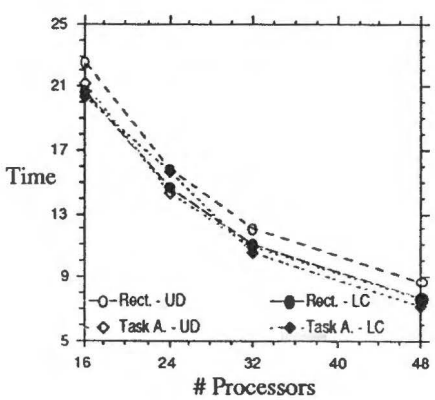

Figure A.26: Total time comparison, TC2000, Laser image, hi-res

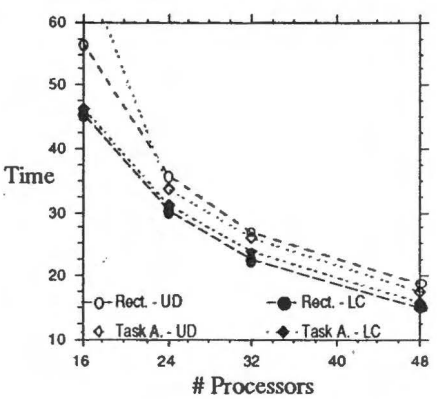

Figure A.28: Total time comparison, TC2000, mountain image, hi-res

Index

— Jones and Bartlett Books in Computer Science and Related Areas —

Barnsley, M., *The Fractal Transform*
ISBN 0-86720-218-1

Bernstein, A.J., and Lewis, P.M., *Concurrency in Programming and Database Systems*
ISBN 0-86720-205-X

Birmingham, W.P., Gupta, A.P., and Siewiorek, D., *Automating the Design of Computer Systems: The MICON Project*
ISBN 0-86720-241-6

Chandy, K.M., and Taylor, S., *An Introduction to Parallel Programming*
ISBN 0-86720-208-4

Epstein, D.B.A., *et al., Word Processing in Groups*
ISBN 0-86720-241-6

Flynn, A., and Jones, J., *Mobile Robots: Inspiration to Implementation*
ISBN 0-86720-223-8

Geometry Center, University of Minnesota, *Not Knot* (VHS video)
ISBN 0-86720-240-8

Iterated Systems, Inc., *Floppy Book: A P.OEM PC Book*
ISBN 0-86720-222-X

Iterated Systems, Inc., *SNAPSHOTS: True-Color Photo Images Using the Fractal Formatter*
ISBN 0-86720-299-8

Lee, E. S., *Algorithms and Data Structures in Computer Engineering*
ISBN 0-86720-219-X

Meyers, B.A. (ed.), *Languages for Developing User Interfaces*
ISBN 0-86720-450-8

Parke, F.I., and Waters, K., *Computer Facial Animation*
ISBN 0-86720-243-2

Ruskai, M.B., *et al., Wavelets and Their Applications*
ISBN 0-86720-225-4

Whitman, S., *Multiprocessor Methods for Computer Graphics Rendering*
ISBN 0-86720-229-7

MULTIPROCESSOR METHODS FOR COMPUTER GRAPHICS RENDERING

SCOTT WHITMAN

This book is a thorough investigation into the problem of using a massively parallel computer to render three-dimensional computer graphics scenes into images. The algorithms that are analyzed in this monograph represent several alternative approaches to image space decomposition as well as remote memory access. Pointers are given so that researchers intending to develop their own parallel rendering algorithms will be able to obtain high performance and good speedup from a variety of multiprocessor architectures.

ISBN 0-86720-229-7

JONES AND BARTLETT PUBLISHERS