Hierarchical Radiosity on Multicomputers: a Load–Balanced Approach

E. Padrón, M. Amor, J. Touriño, and R. Doallo[*]

1 Introduction

The illumination model is one of the most important factors to obtain a realistic appearance of synthetic computer images. Almost all current image synthesis environments are based on a sequence of stages to represent an image (*pipeline rendering*); one of these stages is a global illumination algorithm. It is based on mathematical models about the interaction of rays of light with surfaces and their propagation through the environment. The radiosity method is one of the most popular global illumination models. It is the most realistic approach because it is based on the real behavior of light in a closed environment in a view-independent way.

The radiosity equation derives from the rendering equation [9] under the assumption that all surfaces and light sources exhibit Lambertian diffuse reflection and emission, respectively. The radiosity $B(x)$ emitted from a surface of a domain S has the form:

$$B(x) = E(x) + \rho(x)\int_S B(x')G(x,x')dA' \tag{1}$$

where $E(x)$ is the emission energy, $\rho(x)$ is the diffuse reflectivity, $G(x,x')$ is a function of geometry relationship between two surface points x and x', and A' is the area of the surface that contains x'. $G(x,x')$ is given by

$$G(x,x') = \frac{cos\ \theta\ cos\ \theta'}{|x-x'|^2}V(x,x'). \tag{2}$$

This term consists of the cosines made by the vector connecting x and x' with their

[*]Department of Electronics and Systems, University of A Coruña, 15071 A Coruña, Spain ({emilioj,margamor,juan,doallo}@des.fi.udc.es).

respective surface normals, the distance between these two points, and the visibility function V, whose value is 1 if x and x' are mutually visible, or 0 otherwise.

In order to solve a global illumination problem, the radiosity method computes an approximation to (1) by discretizing the domain. Thus, the surfaces of the domain are broken down into a collection of N disjoint polygons or patches $(P_i)_{i=1,\cdots,N}$. In order to further simplify the problem, an assumption is made: radiosity takes a uniform value across the surface of each patch. Consequently, each patch P_i is assumed to have a uniform radiosity B_i, a uniform emission of light E_i, and a constant reflectivity ρ_i. The discrete version of the radiosity equation is derived from (1):

$$B_i = E_i + \rho_i \sum_{j=1}^{N} B_j F_{ij} \tag{3}$$

where

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{cos\theta_i cos\theta_j}{\pi d^2} V(P_j, P_i) dA_j dA_i. \tag{4}$$

F_{ij} is called the form factor between patches P_i and P_j. It is the proportion of the radiosity leaving patch P_i that is received by patch P_j (d is the distance between these two patches). There is one radiosity equation (3) for each patch. The resultant equation system can be expanded into matrix form:

$$\begin{pmatrix} 1-\rho_1 F_{11} & \dots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & \dots & -\rho_2 F_{2N} \\ \vdots & \ddots & \vdots \\ -\rho_{N-1} F_{N-1,1} & \dots & -\rho_{N-1} F_{N-1,N} \\ -\rho_N F_{N1} & \dots & 1-\rho_N F_{NN} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_{N-1} \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_{N-1} \\ E_N \end{pmatrix}. \tag{5}$$

As the radiosity equation system is liable to be very large and relatively dense, iterative methods (like Gauss-Seidel, Jacobi, Southwell ...) are more appropriate than direct methods to solve it. Still, they have a high computational and memory cost and, thus, alternative strategies have been developed to reduce the size and complexity of the radiosity problem, as follows.

Cohen et al. [3] describe an algorithm called *progressive refinement* or *progressive radiosity.* The physical interpretation is that, during one iteration, the patch with the greatest unshot radiosity is chosen and its radiosity is shot through the domain. One shooting step takes $\mathcal{O}(N)$ operations and can be viewed as multiplying the scalar B_i by a column of the form factor matrix (5). Cohen et al. show that in many cases only a small fraction of N shooting steps is required to approximate a solution closely. *Wavelet radiosity* [6], [12] employs multilevel meshes to represent the radiosity function, and allows inter–patch interactions to take place between arbitrary levels of the mesh hierarchy.

Hierarchical algorithms [7] are based on an adaptive subdivision of the scene. Starting from the input polygons (patches), a hierarchy of elements with different refinement levels depending on the required precision is built. Therefore, the interactions between patches can involve elements of different levels in the hierarchy. This approach condenses entire blocks of the form factor matrix, resulting in a $\mathcal{O}(k^2 + N)$ complexity (k is the number of top-level elements, which are finally meshed into N elements). The hierarchical method is more accurate than the progressive approach and faster than wavelet radiosity. Although the hierarchical method drastically reduces the $\mathcal{O}(N^2)$ complexity of the classical radiosity algorithm of (3), it still has a significant computational cost, which justifies the use of parallel computing techniques. In this work, we propose a parallel implementation of a hierarchical radiosity method on multicomputers, where the load is dynamically balanced to avoid idle processors.

The paper is organized as follows: next section reviews related work about parallel radiosity algorithms, while Section 3 describes our parallel approach. Experimental results in terms of speedups and numerical errors are presented in Section 4. Finally, conclusions and future work are discussed in Section 5.

2 Related work

In the literature, several parallel approaches have been proposed to speed up the radiosity calculation. Parallel algorithms based on the progressive method are proposed in [1] (on a distributed-memory computer, Intel Paragon XP/S), and in [11] (on a distributed-shared memory machine, SGI Origin 2000). Although the parallel programming paradigms of both works are different, they use the same strategy: the domain is divided into subdomains using *Virtual Interfaces* and *Visibility Masks* to achieve data locality and, thus, reduce data traffic both in the local memory hierarchy and between the processors. In [2], a parallel wavelet radiosity algorithm is presented for the Origin 2000.

Focusing on the hierarchical method, good results have been reported on shared-memory multiprocessors [14], where all the processors have access to the whole scene. However, the results on distributed-memory are not so encouraging, mainly due to the communication overhead. In [15] a fine–grain parallelism was applied using a master-slave paradigm, where each slave calculates ray-polygon interactions on the corresponding subset of elements of the scene. In this case, the speedup of the algorithm was restrained by the bottleneck of having a master processor and the large number of communications required.

Other multicomputer implementations also follow a master-slave model, but using a coarse–grain parallelism. In that case, each slave performs the whole computation of the radiosity on a group of patches of the scene, and the master takes charge of the dynamic patch distribution, as well as the convergence analysis. Among these implementations, one approach is to store the complete scene into the local memory of each processor [4], in order to minimize communications, although large scenes cannot be processed due to memory requirements. Another approach is to distribute the scene among the processors [5], which allows to work with larger scenes, although communication overhead increases to a great extent. Our approach follows an SPMD paradigm, that is, we do not waste one processor on load distribution tasks. The scheduling is, therefore, distributed.

Finally, a coarse–grain hierarchical approach based on wavelets is described in [10] (on a heterogeneous computer network). Nevertheless, unlike our algorithm, it is not suited for all kinds of scenes (e.g., for scenes with objects of highly detailed geometry).

3 Parallel implementation

Our parallel version of the hierarchical method is based on the sequential algorithm described in [7], which consists of an iterative process to compute the radiosity of all the elements of the scene, as well as a preprocessing stage to build a BSP (Binary Space Partition) tree and to compute the initial interactions between the patches. The message-passing algorithm has been implemented using MPI and following a coarse-grain approach, that is, each processor performs the whole computation of the radiosity for a set of patches of the scene. Then, the communication between the processors only takes place at the end of each iteration, in order to exchange the updated radiosity values of the elements of the scene. The stages of the parallel algorithm are described in the next subsections.

3.1 BSP construction and initial patch allocation

Each processor builds its own BSP tree with the patches of the scene. This tree will be useful to determine the visibility between two patches in an efficient way. For each patch inserted in the BSP tree, a list of initial interactions (or links) is computed. Each entry of this list has as destination other patch of the scene, potentially visible from the current patch (we consider that two patches are potentially visible if their positive sides are face to face). The form factor between the two patches involved is computed for each interaction. Besides, once the patches are inserted in the BSP tree, they are sorted in decreasing order of area. The sorted patches are cyclically assigned to the processors, in order to achieve an initial load balancing.

3.2 Radiosity computation

During the iterative process in which radiosity is computed, each processor takes charge of its assigned patches. The radiosity obtained from all the visible interactions of each patch is calculated (gathering process). If the radiosity emitted by a certain link exceeds a given threshold, the interaction must be refined (in this work, we have used a BF refinement [7]). To perform this task, either the source element or the destination element of the interaction (depending on which of them has the largest area) is subdivided into a quadtree, where the children inherit the current radiosity of the father. Besides, in each iteration, each processor keeps a record of the destination elements that correspond to patches assigned to a different processor.

The irregular and unpredictable behaviour of the refinement makes a static load distribution inappropriate, due to the appearance of workload imbalance, which results in poor speedups; this fact is more critical as the number of processors increases. Although we tried to overcome this problem by assigning cyclically a list of patches in decreasing order of area, as described in the previous subsection, it is not enough. Thus, we have implemented a dynamic load distribution as follows. In the first iteration of the algorithm, if a processor finishes its corresponding computations, the next step is to check the presence of non-processed (free) patches in the ordered global list. If so, the processor takes the last free patch from the list and computes its radiosity. This procedure is repeated until the list of non-processed patches is empty. We have applied the scheduling only in the first iteration (although the resultant load distribution is also used in the next iterations) because we have experimentally checked that most of the execution time of the algorithm is consumed in the first iteration, and so it is critical to achieve load balancing during its execution.

The main drawback of this scheduling lies in the fact that two or more processors could compete for the same patch. In Subsection 3.4, we describe the protocol developed to overcome this problem.

3.3 Communication stage and convergence test

Once the local calculation of radiosity in one iteration is completed, the processors start a global communication phase to update both radiosity values and tree structures. In this phase, each processor sends and receives data from the other processors, following an all-to-all communication pattern implemented by MPI total exchange routines. First, the processors exchange the lists containing the identifiers of the elements they need: `MPI_Alltoall` is used to communicate the exact number of elements that each processor requires of the other processors, and `MPI_Alltoallv` is used to exchange those element identifiers according to the information provided by the previous routine. Next, each processor updates the hierarchical structures of the assigned patches: if a processor receives a request about an element that corresponds to a patch that had not been still refined up to the level of that element, it is refined until the required level is achieved. Then, each processor sends the updated radiosity values of the requested elements (see the example of Fig. 1).

After the communication stage of each iteration, the complete radiosity of the scene is summed up and the convergence is checked in parallel by means of a reduction operation (`MPI_Allreduce`). Each processor contributes the partial radiosity of its set of assigned patches to the reduction and, this way, the whole radiosity of the scene is obtained in all the processors. Next, each processor compares this value with the radiosity in the previous iteration. If the convergence criterion is fulfilled, the iterative algorithm ends; otherwise, a new iteration to compute radiosity begins. For the next iterations, each processor uses the same patches as in the first iteration (both patches assigned statically and those ones taken from the list dynamically).

3.4 Load balancing protocol

In order to carry out a dynamic patch allocation, each processor must keep updated information about the patches that have not been still processed. This information is stored in the ordered list of patches and must be available in every processor. Thus, before processing a patch, each processor communicates this state to the rest of processors. A drawback arises when two or more processors compete for the same patch. To avoid the assignment of the same patch to different processors, we have

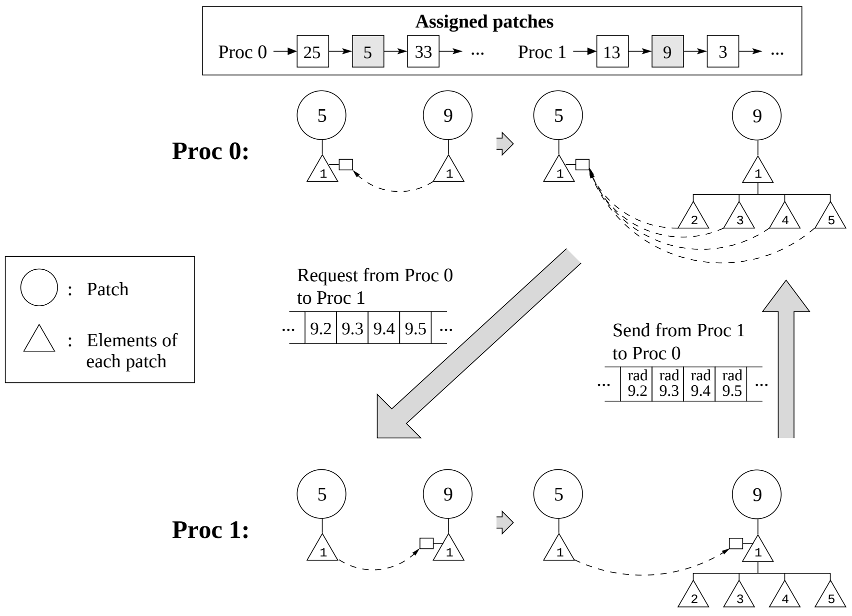


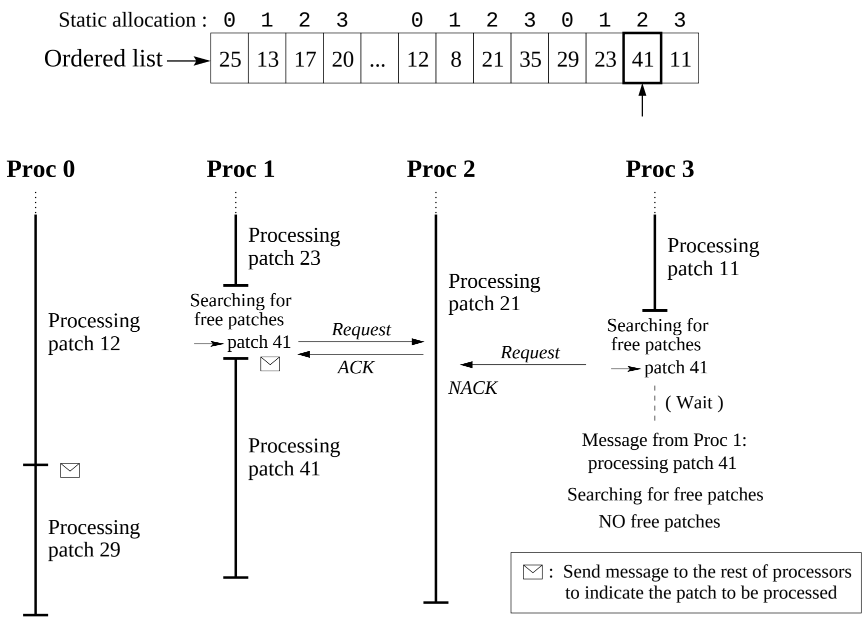
Figure 1. *Processor 0 subdivides patch 9 to refine the interaction between patches 5 and 9. Later, processor 0 asks processor 1 for the radiosity of the elements of patch 9. Thus, processor 1 must refine patch 9 before sending the radiosity of its elements to processor 0.*

implemented a protocol based on making requests about the state of the patch that causes the conflict.

A processor, before taking a free patch, sends a request message to the owner of that patch, that is, the processor that has the patch by means of the static cyclic assignment (which is known by all the processors). If the owner of the patch is not still processing it, the ownership of the patch is transferred to the requesting processor (ACK) provided that the patch had not been still given to any other processor. Otherwise, the patch request is refused (NACK). Note that in this case explicit messages are not used because the processor that is taking charge of that patch communicates this situation to the rest of processors.

Using this protocol, any kind of incoherence arising from the multiple assignment of one patch to two or more processors is avoided. For example, in Fig. 2 it can be observed that, once processors 1 and 3 have finished the computations associated with their assigned patches, they search for free patches in the ordered list, beginning from the last patch. Both processors try to get patch 41 (initially assigned to processor 2), but only processor 1 will finally get it; processor 3 carries on with the search of free patches in the list.

During this scheduling stage, nonblocking communications (both send and receive primitives) are used to overlap communication and computation. Besides,



Static allocation : 0 1 2 3 0 1 2 3 0 1 2 3
Ordered list 25 13 17 20 ... 12 8 21 35 29 23 41 11
Proc 0
Proc 1
Proc 2
Proc 3
Processing patch 12
Processing patch 29
Processing patch 23
Searching for free patches
patch 41
Processing patch 41
Request
ACK
Processing patch 21
NACK
Request
Processing patch 11
Searching for free patches
patch 41
(Wait)
Message from Proc 1: processing patch 41
Searching for free patches
NO free patches
: Send message to the rest of processors to indicate the patch to be processed

Figure 2. *Example of the protocol: processors 1 and 3 compete for patch 41, but only processor 1 gets the patch.*

as the messages to be sent in this stage have the same format and size, as well as the same destinations, we have used MPI persistent communications. Therefore, the tasks involved in setting up the communication are accomplished only once.

4 Experimental Results

We have tested our parallel algorithm on the Fujitsu AP3000 [8] and Cray T3E [13] distributed-memory computers, using a graphical workstation as front-end to display the illuminated scene. The nodes of our AP3000 are UltraSparc-II at 300 Mhz connected via a network called AP-Net (with a bandwidth of 200 Mbytes/s) in a 2D torus topology. The T3E we have used consists of Alpha 21164 processors at 300 Mhz with a 3D torus interconnection topology and a network bandwidth of 480 Mbytes/s. The test scene is composed of 1292 input triangles and is depicted in Fig. 3. The resultant illuminated scene is shown in Fig. 4.

The results in terms of speedups for the AP3000 (up to 12 processors) and the T3E (up to 30 nodes) are shown in Figures 5 and 6, respectively, both for a static cyclic patch assignment and for the approach that includes a dynamic scheduling. The execution time of the sequential algorithm is 2309 seconds on the AP3000 and 1293 on the T3E; and it is 222 seconds using the static parallel algorithm and 198 seconds with the dynamic scheduling on 12 processors of the AP3000 (57 and 48 seconds, respectively, for 30 processors on the T3E). It is clear that the execution

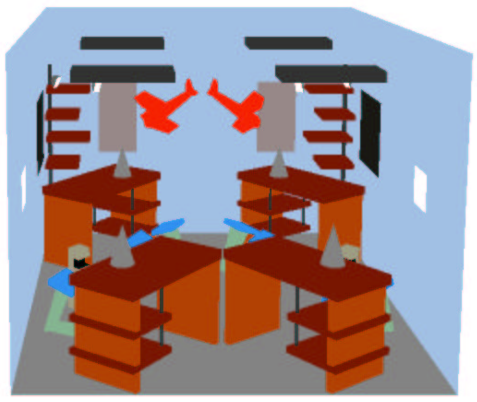


Figure 3. *Original test scene.*

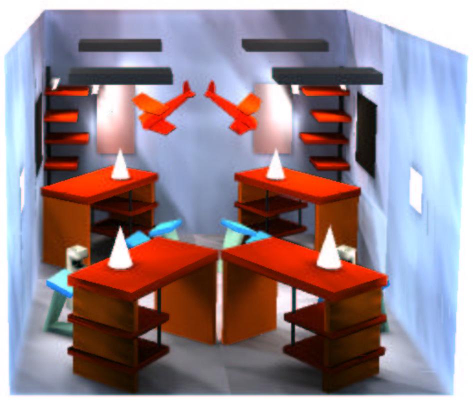


Figure 4. *Illuminated test scene.*

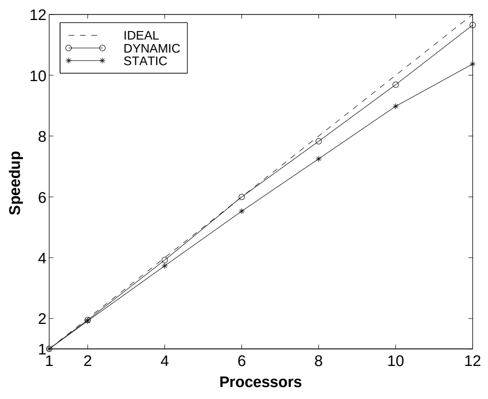


Figure 5. *Speedups on the AP3000.*

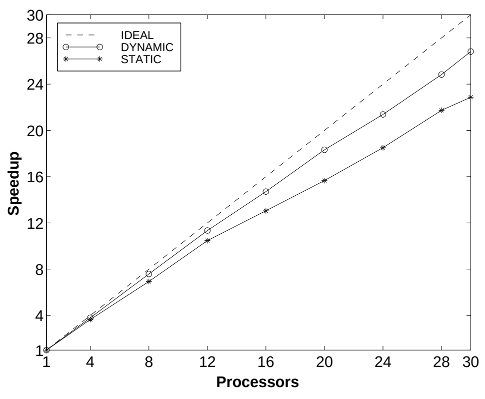


Figure 6. *Speedups on the T3E.*

times are much better on the T3E due to the faster processors and network it has.

As can be observed, the speedups are better using the dynamic scheduling, and the difference between the static and the dynamic strategies is greater as the number of processors increases, due to the effect of load imbalance. The speedups are greatly improved using the dynamic scheduling that balances the load, and are very close to the ideal curve. Nevertheless, speedups fall slightly in the T3E from the 16-processor configuration because the local radiosity computation assigned to each processor is less and less significant in relation to the communication overhead.

As an example, load balancing was experimentally checked by measuring in each processor (for a 12-processor configuration on the AP3000) the execution time of the first iteration of the algorithm, both for the dynamic and static approaches. The results are depicted in Fig. 7. In the static case, processor 3 is a bottleneck because the iteration does not finish until all processors finish; so, the rest of processors are idle waiting for processor 3. The dynamic scheduling, according to Fig. 7,

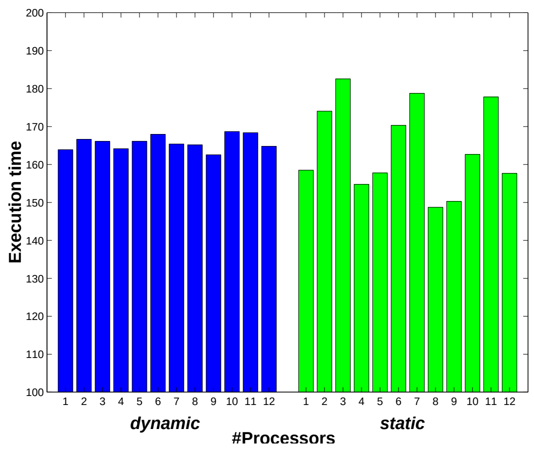


Figure 7. *Execution time (in seconds) of the first iteration for each processor on the AP3000.*

balances the load and, thus, minimizes idle times.

In further experiments we tried to reduce the communication overhead at the expense of a coarser load balance. That is, we varied the scheduling (see Subsection 3.2) so that each processor took a set of say k free patches of the list instead of only one. The results showed that, for small values of k, the execution times were very close to $k = 1$, which meant that the overhead of the scheduling was adequately overlapped with the computations of the algorithm. As k increased, the execution times were coming close to the static algorithm because load balance was not so accurate. Therefore, $k = 1$ was the best choice and we maintained it in the algorithm.

4.1 Numerical errors

Regarding the correctness of the algorithm results (see the illuminated scene in Fig. 4), we have used the residual error of the radiosity as error metric:

$$r_i = \widehat{B}_i^l - E_i - \rho_i \sum_{j=1}^{N} \widehat{B}_j^l F_{ij} \tag{6}$$

where $\widehat{B}_i^l / \widehat{B}_j^l$ is the radiosity at patch i/j in iteration l (the rest of parameters correspond to those of (3)).

Table 1 presents several statistical parameters of the residual errors for the sequential algorithm and for the parallel algorithm with dynamic scheduling on the AP3000: mean, maximum relative error ($max(r_i/\widehat{B}_i^l)$), standard deviation and maximum deviation.

As can be observed, errors are small, which shows the accuracy of the algorithm. Nevertheless, the sequential algorithm presents smaller errors because it uses

Table 1. *Statistical parameters of the residual errors for the test scene.*

	sequential	2 Proc	4 Proc	6 Proc	8 Proc
mean	8.72E-4	9.92E-4	18.41E-4	21.69E-4	22.20E-4
m.r.e.	1.94E-2	2.27E-2	4.62E-2	3.18E-2	4.29E-2
s. d.	1.35E-3	0.96E-3	1.54E-3	1.82E-3	1.78E-3
max. d.	1.73E-2	0.97E-2	1.74E-2	2.11E-2	1.60E-2

the Gauss-Seidel method, that is, the values calculated in the current iteration are used to compute the rest of values of the iteration ($\widehat{B}_i^l$ is computed using the $\widehat{B}_j^l$ values). That is not the case for the parallel approach, because during the current iteration, one processor cannot have access to the values calculated by other processor in the same iteration, as the values are provided between iterations. Therefore, a combination of Gauss-Seidel and Jacobi methods is used in the parallel algorithm ($\widehat{B}_i^l$ is computed using the $\widehat{B}_j^l$ value if patches i and j belong to the same processor; otherwise, the $\widehat{B}_j^{l-1}$ value is used), which requires more iterations to converge and results in higher residual values.

5 Conclusions

In this paper we have described a parallel implementation of the hierarchical radiosity method on distributed-memory multiprocessors. The irregular and dynamic behavior of the method causes load imbalance. In order to improve the performance, we tried to assign the same number of radiosity computations to each processor by means of two steps. First, the patches are ordered by decreasing area and cyclically distributed. Second, a distributed scheduling performs a finer tuning to balance the load dynamically, by reassigning the smallest non-processed patch to any processor as soon as it finishes its work. As a result, load balance and good speedups have been achieved through this approach, as we have experimentally shown on the AP3000 and T3E multicomputers. Finally, the parallel algorithm presents a small loss of accuracy with respect to the sequential algorithm.

As future work, we intend to reduce memory overhead by detecting sub-environments in the scene that involve local interactions. Thus, the scene could be completely distributed among the processors so that only the visibility zones between sub-environments should be replicated.

Acknowledgments

We gratefully thank CESGA (Centro de Supercomputación de Galicia, Santiago de Compostela, Spain) and CIEMAT (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas, Madrid, Spain) for providing access to the Fujitsu AP3000 and Cray T3E, respectively.

Bibliography

[1] B. Arnaldi, T. Priol, L. Renambot and X. Pueyo, *Visibility masks for solving complex radiosity computations on multiprocessors*, in Proc. First Eurographics Workshop on Parallel Graphics and Visualisation, 1996, pp. 219–232.

[2] X. Cavin, L. Alonso and J.-C. Paul, *Parallel wavelet radiosity*, in Proc. Second Eurographics Workshop on Parallel Graphics and Visualisation, 1998, pp. 61–75.

[3] M. Cohen, S.E. Chen, J.R. Wallace and D.P. Greenberg, *A progressive refinement approach to fast radiosity image generation*, Computer Graphics, 22 (1988), pp. 75–84.

[4] C.-C. Feng and C. Yang, *A parallel hierarchical radiosity algorithm for complex scenes*, in Proc. IEEE Parallel Rendering Symposium, 1997, pp. 71–78.

[5] T.A. Funkhouser, *Coarse-grained parallelism for hierarchical radiosity using group iterative methods*, in Proc. ACM SIGGRAPH, 1996, pp. 343–352.

[6] S.J. Gortler, P. Schröder, M.F. Cohen and P. Hanrahan, *Wavelet radiosity*, in Proc. ACM SIGGRAPH, 1993, pp. 221–230.

[7] P. Hanrahan, D. Salzman and L. Aupperle, *A rapid hierarchical radiosity algorithm*, Computer Graphics, 25 (1991), pp. 197–206.

[8] H. Ishihata, M. Takahashi and H. Sato, *Hardware of AP3000 scalar parallel server*, Fujitsu Sci. Tech. J., 33 (1997), pp. 24–30.

[9] J. T. Kajiya, *The rendering equation*, Computer Graphics, 20 (1986), pp. 143–150.

[10] D. Meneveaux and K. Bouatouch, *Synchronization and load balancing for parallel hierarchical radiosity of complex scenes on a heterogeneous computer network*, Computer Graphics Forum, 18 (1999), pp. 201–212.

[11] L. Renambot, B. Arnaldi, T. Priol and X. Pueyo, *Towards efficient parallel radiosity for DSM-based parallel computers using virtual interfaces*, in Proc. IEEE Parallel Rendering Symposium, 1997, pp. 79–86.

[12] P. Schröder, S.J. Gortler, M.F. Cohen and P. Hanrahan, *Wavelet projections for radiosity*, Computer Graphics Forum, 13 (1994), pp. 141–151.

[13] S.L. Scott, *Synchronization and communication in the T3E multiprocessor*, in Proc. ASPLOS-VII, 1996, pp. 26–37.

[14] J.P. Singh, A. Gupta and M. Levoy, *Parallel visualization algorithms: performance and architectural implications*, IEEE Computer, 27 (1994), pp. 45–55.

[15] D. Zareski, B. Wade, P. Hubbard and P. Shirley, *Efficient parallel global illumination using density estimation*, in Proc. IEEE Parallel Rendering Symposium, 1995, pp. 47–54.