Progressive Refinement Radiosity on Ring-Connected Multicomputers *

Tolga K. Çapın, Cevdet Aykanat, Bülent Özgüç
Department of Computer Engineering and Information Science
Bilkent University
06533 Bilkent, Ankara, Turkey

Abstract

The progressive refinement method is investigated for parallelization on ring-connected multicomputers. A synchronous scheme, based on static task assignment, is proposed, in order to achieve better coherence during the parallel light distribution computations. An efficient global circulation scheme is proposed for the parallel light distribution computations, which reduces the total volume of concurrent communication by an asymptotical factor. The proposed parallel algorithm is implemented on a ring-embedded Intel's iPSC/2 hypercube multicomputer. Load balance quality of the proposed static assignment schemes are evaluated experimentally. The effect of coherence in the parallel light distribution computations on the shooting patch selection sequence is also investigated.

Keywords : Progressive refinement radiosity, parallel computing, multicomputers, ring interconnection topology.

1 Introduction

Radiosity [7] is an increasingly popular method for generating realistic images of nonexisting environments. The recently proposed *progressive refinement radiosity* [4] allows to view the approximated partial radiosity solutions initially and approaches to the correct solution iteratively. However, the operations still require excessive computational power and limit the usage of the method for complex scenes with a large number of patches. Therefore, one can exploit *parallelism* in progressive refinement radiosity to achieve near-interactive image generation speeds.

In this work, we investigate the parallelization of the progressive refinement method for ring-connected multicomputers. In a multicomputer, processors have only local memories and there is no shared memory. In these architectures, synchronization and coordination among processors are achieved through explicit message passing. Multicomputers have been popular due to their nice scalability feature. Various interconnection topologies have been proposed and implemented for connecting the processors of multicomputers. Among them, ring topology is the simplest topology which requires only two links per processor. Ring topology can easily be embedded onto almost all other interconnection topologies (e.g. hypercube, 2D mesh, 3D mesh, etc). Hence, parallel algorithms developed for ring topologies can easily be adapted to other topologies.

The parallel progressive refinement implementations proposed in the literature [1, 2, 6, 8, 9] utilize asynchronous schemes based on demand-driven task assignment. The parallel progressive refinement algorithm proposed in this work utilizes a synchronous scheme based on static task assignment. The synchronous scheme is proposed in order to achieve better coherence during parallel light distribution computations. The proposed algorithm is implemented on a ring-embedded Intel iPSC/2 hypercube multicomputer. The organization of the paper is as follows. Section 2 summarizes the progressive refinement radiosity. Section 3 dicusses the parallelization of progressive refinement method. Finally, experimental results are presented and discussed in Section 4.

*This work is partially supported by Intel Supercomputer Systems Division grant no. SSD100791-2 and Turkish Scientific and Technical Research Council (TÜBİTAK) grant no. EEEAG-5

0-8186-4920-8/93 $3.00

2 Progressive Refinement Radiosity

The progressive refinement radiosity gives an initial approximation to the illumination of the environment and approaches to the correct light distribution iteratively. Each iteration can be considered as a four phase process:

1. Shooting patch selection,
2. Production of hemicube item-buffers,
3. Conversion of item-buffers to a form-factor vector,
4. Light distribution using the form-factor vector.

In the first phase, the patch with maximum energy is selected for faster convergence. In the second phase, a hemicube [3] is placed onto this patch and all other patches are projected onto the item-buffers of the hemicube using the z-buffer for hidden patch removal. The patches are passed through a projection pipeline consisting of: visibility test, clipping, perspective projection and scan-conversion. In the third phase, the form-factor vector corresponding to the selected shooting patch is constructed from the hemicube item-buffers by scanning the hemicube and adding the delta form-factors of the pixels that belong to the same patch.

In the last phase, light energy of the shooting patch is distributed to the environment, by adding the light contributions from the shooting patch to the other patches. Distribution of light energy necessitates the use of the form-factor vector computed in Phase 3. The contribution from the shooting patch i to patch j is given by [4]:

$$\Delta R(r,g,b) = r_j(r,g,b)\Delta B_i(r,g,b)F_{ij}A_i/A_j \quad (1)$$

$$B_j(r,g,b) = B_j(r,g,b) + \Delta R(r,g,b) \quad (2)$$

$$\Delta B_j(r,g,b) = \Delta B_j(r,g,b) + \Delta R(r,g,b) \quad (3)$$

In Eq.(1), $\Delta B_i(r,g,b)$ denotes the delta radiosity of patch i, $r_j(r,g,b)$ is the reflectivity value of the patch j for 3 color-bands, A_j denotes the area of the patch j, F_{ij} denotes the j^{th} element of the form-factor vector constructed in Phase 3 for the shooting patch i. During the execution

of the algorithm, a patch may be selected as the shooting patch more than once, therefore a delta radiosity value (ΔB) is stored in addition to the radiosity (B) of the patch, which gives the difference between the current energy and the last estimate distributed from the patch (i.e. the amount of light the patch has gathered since the last shooting from the patch). This iterative process is halted when $\Delta B_i A_i$ values for all the patches reduce below a user-specified tolerance value.

3 Parallelization

The ring topology is selected because of its *simplicity* requiring only two links per processor and because the ring can be *embedded* onto a wide range of popular topologies such as the hypercube, 2D mesh, 3D mesh. The processors in the ring perform the radiosity computations and send the computed radiosity values of the patches to the host, and the host runs the rendering program using these values. In this way, the processors can compute further iterations in parallel with display of previous iteration results on the host.

As is mentioned earlier, progressive refinement radiosity is an iterative algorithm. Hence, computations involved in an individual iteration should be investigated for parallelization while considering a proper interface between successive iterations. In this algorithm, strong computational and data dependencies exist between successive phases such that each phase requires the computational results of the previous phase in an iteration. Hence, parallelism at each phase should be investigated individually while considering the dependencies between successive phases. Furthermore, strong computational and data dependencies also exist within each computational phase. These *intra-phase* dependencies necessitate global interaction which may result in global interprocessor communication at each phase on a distributed-memory architecture. Considering the crucial granularity issue in parallel algorithm development for coarse-grain multicomputers we have investigated a parallelization scheme which slightly modifies the original sequential algorithm. In the modified algorithm, instead of choosing a single patch, P shooting patches are selected at a time on a multicomputer with P processors. The modified algorithm is still an iterative algorithm where each iteration involves the following:

1. Selection of P shooting patches,
2. Production of P hemicube item-buffers,
3. Conversion of P hemicubes to P form-factor vectors,
4. Distribution of light energy from P shooting patches using these P form-factor vectors.

Note that, the structure of the modified algorithm is very similar to that of the original algorithm. However, the computations involved in P successive iterations of the original algorithm are performed simultaneously in a single iteration of the modified algorithm. This modification increases the granularity of the computational phases since the amount of computation involved in each phase is duplicated P times. Furthermore, it simplifies the parallelization since production of P hemicube buffers (Phase 2) and production of P form-factor vectors (Phase 3) can be performed simultaneously and independently. Hence, processors can concurrently construct P form-factor vectors corresponding to P different shooting patches without any communication.

The modified algorithm is an approximation to the original progressive refinement method. The coherence of the shooting patch selection sequence is disturbed in the modified algorithm. The selection of P shooting patches at a time ignores the effect of the mutual light distributions between these patches and the light distributions of these patches onto other patches during this selection. Thus, the sequence of shooting patches selected in the modified algorithm may deviate from the sequence to be selected in the original algorithm. This deviation may result in a greater number of shooting patch selections for convergence. Hence, the modification introduced for the sake of parallelization may degrade the performance of the original algorithm. This performance degradation is likely to increase with the increasing number of processors. Section 4 presents an experimental investigation of this issue.

There are various parallel radiosity implementations in the recent literature [1, 2, 5, 6, 8, 9, 10]. The algorithmic modification mentioned here is similar to the parallel implementations discussed in [2, 6, 9]. However, these parallel implementations utilize an *asynchronous* scheme. These asynchronous schemes have the advantage of minimizing the processors' idle time since form-factor and light distribution computations proceed concurrently in an asynchronous manner. However in these schemes a processor, upon completing a form-factor vector computation for a shooting patch, selects a new shooting patch for a new form-factor computation. Hence, this shooting patch selection by an individual processor does not consider the light contributions of the form-factor computations concurrently performed by other processors. In this work, we propose a synchronous scheme which is expected to achieve better coherence in the distributed shooting patch selections. The parallelization of the proposed scheme is discussed in the following subsections.

3.1 Phase 1: Shooting Patch Selection

There are two alternative schemes for performing this phase: local shooting patch selection and global shooting patch selection. In the local selection scheme, each processor selects the patch with maximum $\Delta B_i A_i$ value among its local patches. In the global selection scheme, each processor selects the first P patches with the greatest $\Delta B_i A_i$ value among its local patches and puts these patches (together with their geometry and color data) into a local buffer in decreasing order according to their $\Delta B_i A_i$ values. Then, these buffers of sizes P are circulated in P concurrent communication steps as follows. In each concurrent step, each processor merges its sorted buffer of size P with the sorted buffer received of size P, discarding P patches with smaller $\Delta B_i A_i$ values. Then, each processor sends the resulting buffer to the next processor in the ring. Note that, each processor keeps its original local buffer intact during the circulation. At the end of P communication steps, each processor holds a copy of the same sequence of P patches with maximum $\Delta B_i A_i$ values in decreasing order. Then, each processor k selects the k^{th} patch in the local sorted patch list.

The number of shooting patch selections required for convergence of the parallel algorithm to the user-specified tolerance depends on the shooting patch selection scheme. Global scheme is expected to converge more quickly because the patches with *globally* maximum energy are selected. However, in the local scheme, the shooting patches that are selected may deviate largely, if maximum energy holding patches are gathered in some of the processors, while the other processors hold less energy holding patches. Hence, the global scheme is expected to achieve better coherence in distributed shooting patch selection. However, the global scheme requires circulation and comparison of P buffers,

hence necessitating global communication overhead.

3.2 Phase 2: Hemicube Production

In this phase, each processor needs to maintain a hemicube for constructing the form-factor vector corresponding to its local shooting patch. Furthermore, each processor needs to access the whole scene description in order to fill its local hemicube item-buffers corresponding to its local shooting patch. One approach is to replicate the whole patch geometry data in all the processors, hence avoiding interprocessor communication. However, this approach is not suitable for complex scenes with large numbers of patches because of the excessive memory requirement per processor. Hence, a more valid approach is to evenly decompose whole scene description into P patch data subsets and map each data subset to a distinct processor of the multicomputer. However, the decomposition of the scene data necessitates global interprocessor communication in this phase since each processor owns only a portion of the patch database and needs to access the whole database. The local patch data of each processor should visit all other processors at each iteration.

Patch circulation needed in this phase can be achieved in P concurrent communication steps as follows. In each concurrent step, the current subset of the patch data in the local memory of the processor is projected onto the local hemicube; then this subset is sent to the next processor in the ring, and the new subset is received in a single communication phase. Note that, only geometry data of the patches (the patch vertex coordinates in 3D, patch normals, patch *id*'s) are needed for projecting the patches in this phase. As the messages can only be sent and received from/into contiguous memory blocks, patch data are divided into geometry and color parts in different arrays.

At the end of P concurrent communication steps, each processor completes the projection of all patches onto its local hemicube. Although P-1 communications would be enough for this operation, one more communication is required in order to have the geometry data of local patches in the processors' local memory for maintaining consistency of geometry and color data for rendering and further iterations. It follows that parallel complexity of Phase 2 is:

$$T_{P2} = Pt_{SU} + P(N/P)T_{TR} + P(N/P)T_{PRO} \quad (4)$$

$$= Pt_{SU} + NT_{TR} + NT_{PRO} \quad (5)$$

Here, t_{SU} represents the message start-up overhead or the message latency, T_{TR} is the time taken for the transmission of a single patch geometry, T_{PRO} is the average time taken to project and scan-convert one patch onto a hemicube and N is the total number of patches in the scene.

There are two crucial factors that affect the efficiency of the parallelization in this phase: load imbalance and communication overhead. Note that, the parallel complexity given in Eq. (5) assumes a perfect load balance among processors. Mapping equal number of patches to each processor achieves balanced communication volume between successive processors in the ring. Furthermore, as will be discussed later, it achieves perfect load balance among processors in the parallel light distribution phase (Phase 4). However, this mapping may not achieve computational balance in the parallel hemicube production phase (Phase 2).

The complexity of the projection of an individual patch onto a hemicube depends on several geometric factors. Recall that, each patch passes through a projection pipeline consisting of visibility test, clipping, perspective projection and scan-conversion. A patch which is not visible by the shooting patch requires much less computation compared to a visible patch since it leaves the projection pipeline in a very early stage. The complexity of the scan-conversion stage for a particular patch depends strongly on the distance and the orientation of that patch with respect to the shooting patch. That is, a patch with larger projection area on a hemicube requires more scan-conversion computation than a patch with a smaller projection area. As is mentioned earlier, each iteration of the proposed algorithm consists of P concurrent steps. At each step, different processors concurrently perform the projection of different sets of patches onto different hemicubes. Hence, the decomposition scheme should be carefully selected in order to maintain the computational load balance in this phase of the algorithm.

Two possible decomposition schemes are *tiled* and *scattered* decompositions. In *tiled* decomposition, the neighbouring patches are stored in the local memory of the same processor. This type of decomposition can be achieved in the following way: assuming that the patches that belong to the same object are supplied consecutively, the first N/P patches are stored in processor 0, the next N/P patches are allocated to processor 1, etc. At the end of the decomposition, each processor stores almost equal number of patches in its local memory. In *scattered* decomposition, the neighbouring patches are stored in different processors, therefore the patches that belong to an object are shared by different processors. Scattered decomposition can be achieved in the following way: again assuming that the neighbouring patches that belong to the same object are supplied consecutively, the incoming patches are allocated to the processors in a round-robin fashion. That is, the first patch is allocated to processor 0, the next to processor 1, etc. When P patches are allocated, the next incoming patch is allocated to processor 0, and this process continues. When the decomposition is completed, (N *mod* P) processors store $\lceil N/P \rceil$ patches, while the remaining processors store $\lfloor N/P \rfloor$ patches in their local memories. Figure 1 illustrates the scattered and tiled decomposition of a simple scene consisting of four faces of a room. The numbers shown inside the patches indicate *ids* of the processors that store them in their local memory.

Assuming that neighbour patches require almost equal amount of computation for projection on different hemicubes, the scattered decomposition is expected to produce patch partitions requiring almost equal amount of computations in Phase 2. So, it can be expected that the scattered decomposition achieves much better load balance in Phase 2 than the tiled decomposition.

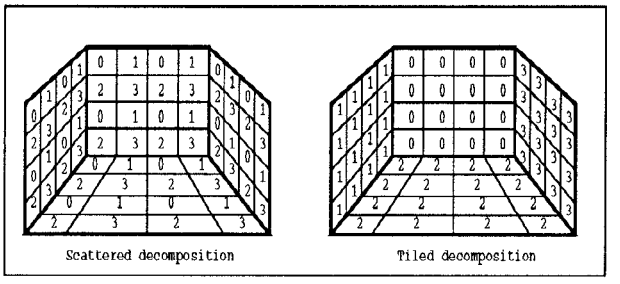

Figure 1: Scattered and tiled decomposition schemes

Communication overhead in this phase consists of two components: number of communications and volume of communications. Each concurrent communication step adds a fixed message set-up time overhead t_{SU} to the parallel algorithm. In medium grain multicomputers (e.g. Intel's iPSC/2 hypercube) t_{SU} is substantially greater than the transmis-

sion time t_{TR} where t_{TR} denotes the time taken for the transmission of a single word. For example, $t_{SU} \approx 550\mu sec$ whereas $t_{TR} \approx 1.44\mu sec$ per word in iPSC/2. Note that, communication of an individual patch geometry involves the transmission of 3 floating point words for the vertices of the triangular patches, 3 words for their normal and one word for the patch id, adding to 52 bytes (i.e. $T_{TR} = 13\ t_{TR}$ in Eq. (5)). However, as seen in Eq. (5), the total number of concurrent communications at each iteration is equal to the number of processors P, whereas the total volume of communication is equal to the number of patches N. Hence, the set-up time overhead can be considered as negligible for complex scenes ($N \gg P$). Then, assuming a perfect load balance, efficiency of Phase 2 can be expressed as:

$$E_{P2} = \frac{1}{P}\frac{PNT_{PRO}}{NT_{PRO} + Pt_{SU} + NT_{TR}} \approx \frac{NT_{PRO}}{NT_{PRO} + NT_{TR}} \quad (6)$$

$$= \frac{T_{PRO}}{T_{PRO} + T_{TR}} = \frac{1}{1 + T_{TR}/T_{PRO}} \quad (7)$$

since one iteration of the parallel algorithm is computationally equivalent to P iterations of the sequential algorithm. Eq. (7) means that projection of an individual patch onto a hemicube involves the communication of its geometry data as an overhead. As is seen in Eq. (7), the overall efficiency of this phase only depends on the ratio T_{TR}/T_{PRO} for sufficiently large N/P. For example, efficiency is expected to increase with increasing patch areas and increasing hemicube resolution, since the granularity of a projection computation increases with these factors.

3.3 Phase 3: Form-Factor Computation

In this phase, each processor can concurrently compute the form-factor vector corresponding to its shooting patch using its local hemicube item-buffers constructed in the previous phase. This phase requires no interprocessor communication. Local form-factor vector computations involved in this phase require scanning all hemicube item-buffer entries. Hence, perfect load balance is easily achieved since each processor maintains a hemicube of equal resolution.

3.4 Phase 4: Contribution Computation

At the end of Phase 3, each processor holds a form-factor vector corresponding to its shooting patch. In this phase, each processor should compute the light contributions from all P shooting patches to its local patches. Hence, each processor needs all form-factor vectors. Thus, this phase necessitates global interprocessor communication since each processor owns only a single form-factor vector.

We introduce a vector notation for the sake of clarity of the presentation in this section. Let $\mathbf{X_k}$ denote the k^{th} slice of a global vector $\mathbf{X}$ assigned to processor k. For example, each processor k can be considered as storing the k^{th} slice of the global array of records representing the whole patch geometry. Each processor k is responsible for computing the k^{th} slice $\mathbf{\Delta R_k}$ of the global contribution vector $\mathbf{\Delta R}$ for updating the k^{th} slices $\mathbf{B_k}$ and $\mathbf{\Delta B_k}$ of the global radiosity and delta radiosity vectors $\mathbf{B}$ and $\mathbf{\Delta B}$, respectively. The notation used to label the P distinct form-factor vectors maintained by P processors is slightly different. In this case, $\mathbf{F}^\ell$ denotes the form-factor vector computed by processor ℓ and $\mathbf{F^\ell_k}$ denotes the k^{th} slice of the local form-factor vector of processor ℓ.

As is seen in Eq. (1), red, green and blue reflectivity values $r_i(r,g,b)$ and the patch area A_i of each patch i are needed as three ratios $r_i(r,g,b)/A_i$. Hence, each processor computes three constants $r_i(r,g,b)/A_i$ for each local patch i during the preprocessing. In vector notation, each processor k can be considered as holding the k^{th} slice $\mathbf{r_k}(r,g,b)$ of the global vector $\mathbf{r}(r,g,b)$ consisting of $r_i(r,g,b)/A_i$ values. Thus, in vector notation, each processor k, for $k = 0, 1, ..., P-1$, is responsible for computing

$$\mathbf{U^\ell_k}(r,g,b) = \Delta B^\ell_s(r,g,b) A^\ell_s \mathbf{F^\ell_k} \quad (8)$$

$$\mathbf{U_k}(r,g,b) = \sum_{\ell=0}^{P-1} \mathbf{U^\ell_k}(r,g,b) \quad (9)$$

$$\mathbf{\Delta R_k}(r,g,b) = \mathbf{r_k}(r,g,b) \times \mathbf{U_k}(r,g,b) \quad (10)$$

where $\Delta B^\ell_s(r,g,b)$ and A^ℓ_s denote the delta radiosity values and the area of the shooting patch of processor ℓ. In Eq. (10), " $\times$ " denotes the element-by-element multiplication of two column vectors. Each processor k can concurrently update its local $\mathbf{B_k}$ and $\mathbf{\Delta B_k}$ vectors by simply performing local vector additions $\mathbf{B_k} = \mathbf{B_k} + \mathbf{\Delta R_k}$ and $\mathbf{\Delta B_k} = \mathbf{\Delta B_k} + \mathbf{\Delta R_k}$ for each color-band. These concurrent update operations do not necessitate any interprocessor communication. It is the parallel computation of the contribution vector $\mathbf{\Delta R_k}$ which requires global interaction.

Note that, the notation used to label the $\mathbf{U}$ vectors is similar to that of the $\mathbf{F}$ vectors since the P $\mathbf{U}$ vectors, of sizes N/P, are concurrently computed by P processors. That is, $\mathbf{U^\ell_k}(r,g,b)$ represents the contribution vector of the shooting patch of processor ℓ to the local patches of processor k omitting the multiplications with the $r_i(r,g,b)/A_i$ coefficients. Hence, $\mathbf{U_k}(r,g,b)$ represents the total contribution vector of all P shooting patches to the local patches of processor k.

The first approach discussed in this work is very similar to the implementation proposed by Chalmers and Paddon [2]. In their implementation, each processor ℓ *broadcasts* a packet consisting of the delta radiosities, area and the form-factor vector of its shooting patch. Each processor k, upon receiving a packet $\{ \Delta B^\ell_s, A^\ell_s, \mathbf{F}^\ell \}$, computes a local contribution vector $\mathbf{U^\ell_k}(\mathrm{r,g,b})$ by performing a local scalar vector product for each color (Eq. (8)) and *accumulates* this vector to its local $\mathbf{U_k}(r,g,b)$ vector by performing a local vector addition operation (Eq. (9)). However, multiple broadcast operations are expensive and may cause excessive congestion in ring interconnection topologies. In this work, indicated packets are circulated in a synchronous manner, similar to the patch circulation discussed for Phase 2. Between each successive communication steps of this form-factor vector circulation scheme, each processor concurrently performs the contribution vector accumulation computations (Eqs. (8) and (9)) corresponding to its current packet. At the end of P-1 concurrent communication steps, each processor k accumulates its total contribution vector $\mathbf{U_k}(r,g,b)$. Then, each processor k can concurrently compute its local $\mathbf{\Delta R_k}(r,g,b)$ vector by performing local element-by-element vector multiplications (Eq. (10)).

It is obvious that perfect load balance in this phase can easily be achieved by mapping equal number of patches to each processor. Hence, the parallel complexity of Phase 2 using the form-factor vector circulation scheme, is:

$$T_{P4} = (P-1)t_{SU} + (P-1)Nt_{tr} + \\ + NT_{CONTR} + (N/P)T_{UPD} \quad (11)$$

Here, t_{tr} is the time taken to transmit a single floating point word, T_{CONTR} is the time taken to compute and accumulate a single contribution value, and T_{UPD} is the time taken to

update a single radiosity and delta radiosity value using the corresponding entry of a local $\mathbf{U_k}$ vector.

Note that, in this scheme, processors accumulate the contributions for their local patches during the circulation of form-factor vectors. Hence, as is also seen in Eq. (11), this scheme necessitates high volume of communication ($(P-1)N$ words) since whole form-factor vectors of sizes N are concurrently communicated in each communication step. However, as is also seen in Eq. (8), each processor k needs only the k^{th} slices (of sizes N/P) of the form-factor vectors it receives during the circulation. That is, form-factor circulation scheme involves the circulation of redundant information. In this work, we propose an efficient scheme which avoids this redundancy in the interprocessor communication. In the proposed scheme, partial contribution computation results ($\mathbf{U}'_{\mathbf{k}}(r,g,b)$ vectors of sizes N/P) are circulated instead of the form-factor vectors (of sizes N). Hence, each processor effectively accumulates the contributions of its local shooting patch to all other processors' local patches during the circulation of the partial contribution computation results.

Figure 2 illustrates the pseudocode for the node program for the proposed contribution vector circulation scheme. This scheme also preserves the perfect load balance, if exactly equal number of patches is mapped to each processor. Hence, the proposed circulation scheme reduces the overall parallel complexity of Phase 4 to

$$T_{P4} = (P-1)T_{SU} + 3(P-1)(N/P)t_{tr} + P(N/P)T_{CONTR} + (N/P)T_{UPD} \quad (12)$$

The constant 3 appears as a coefficient in "t_{tr}" term since each entry of individual $\mathbf{U}'_{\mathbf{k}}$ vectors consists of 3 contribution values for 3 color-bands. Hence, the proposed circulation scheme reduces the total concurrent communication volume in Phase 4 by an asymptotical factor of $P/3$ for $P > 3$.

```
/* ΔB_s(r,g,b) : delta radiosity of local shooting patch;
   A_s : area of local shooting patch;
   F : local form-factor vector (of size N);
   U, ΔR, B, ΔB are local vectors (of size N/P); */
   nextnode = (mynode + 1) mod P;
   prevnode = (mynode - 1) mod P;
   k = mynode;        U(r,g,b) = ΔB_s(r,g,b)A_s F_prevnode;

   for i=1 to P-1 do
       send U(r,g,b) to processor nextnode;
       receive into U(r,g,b) from processor prevnode;
       U(r,g,b) = U(r,g,b) + ΔB_s(r,g,b)A_s F_((k-i-1) mod P);
   endfor
   ΔR(r,g,b) = r(r,g,b) * U(r,g,b);
   B(r,g,b) = B(r,g,b) + ΔR(r,g,b);
   ΔB(r,g,b) = ΔB(r,g,b) + ΔR(r,g,b);
```

Figure 2: The contribution vector circulation scheme

4 Experimental Results

The proposed schemes are implemented on a ring-embedded Intel's iPSC/2 hypercube multicomputer. The form factors are computed using hemicubes of constant resolution $50 \times 100 \times 100$. The proposed parallel algorithms are experimented for six different scenes with 522, 856, 1412, 3424, 5648 and 8352 patches. The test scenes are selected as house interiors consisting of objects such as chairs, tables, windows, lights in order to represent a realistic 3D environment.

Table 1: Effect of local and global shooting patch selection (in Phase 1) on convergence.

N	P	Number of patch selections			Total Execution Time (secs)		
		Loc	Glo	Perc. Dec	Loc	Glob	Perc. Dec
522	4	280	256	8.57	313	288	7.99
	8	320	280	12.50	184	167	9.23
	16	400	304	24.00	120	96	20.00
1412	4	412	376	8.74	790	725	8.22
	8	488	376	22.95	477	374	21.59
	16	592	464	21.62	295	241	18.30
5648	4	464	432	6.90	2618	2450	6.41
	8	592	448	24.32	1692	1289	23.81
	16	688	496	27.91	990	721	27.17

Table 1 illustrates the effect of the local and global shooting patch selection (in Phase 1) on the convergence of the parallel algorithm. As is seen in Table 1, the global selection scheme decreases both the total number of shooting patch selections and the total parallel execution time significantly.

Table 2: Effect of the decomposition scheme on the performance of the parallel hemicube production phase (Phase 2).

N	seq prod time (secs)	P	Tiled Dec. Hemicube		Scattered Dec. Hemicube	
			Prod. Time	Eff.	Prod. Time	Eff.
856	5.020	4	1.866	0.672	1.413	0.888
		8	1.007	0.623	0.720	0.872
		16	0.582	0.539	0.382	0.821
1412	6.460	4	2.435	0.663	1.868	0.865
		8	1.411	0.527	0.946	0.854
		16	0.827	0.488	0.481	0.839
3424	11.799	4	4.660	0.633	3.423	0.862
		8	2.487	0.593	1.731	0.852
		16	1.438	0.513	0.885	0.833
5648	17.335	4	6.646	0.652	4.896	0.885
		8	3.680	0.589	2.496	0.868
		16	2.068	0.524	1.277	0.848
8352	25.901	4	9.092	0.712	7.397	0.875
		8	4.975	0.651	3.824	0.847
		16	2.837	0.571	1.993	0.812

Table 2 shows the effect of the decomposition scheme on the performance of the hemicube production phase. Parallel timings (T_{PAR}) in Table 2 denote the parallel hemicube production time per shooting patch. These timings are computed as the execution time of P concurrent hemicube productions divided by P since P hemicubes are concurrently produced for P shooting patches in a single iteration of Phase 2. Sequential timings (T_{SEQ}) in Table 2 denote the sequential execution time of a single hemicube production. Efficiency values in Table 2 are computed using $Eff = T_{SEQ}/(PT_{PAR})$. Efficiency values are considered as qualitative measures for comparison of the decomposition schemes. As is seen in Table 2, scattered decomposition always achieves better load balance than tiled decomposition.

Table 3 illustrates the execution times of the distributed contribution vector computation during a single iteration of the parallel algorithm. The last column of Table 3 illustrates the percent decrease in the execution times obtained by using the contribution vector circulation instead of form-factor vector circulation. Note that, the advantage of the contribution vector circulation over the form-factor circulation increases with increasing P as is expected.

Table 3: Effect of the circulation scheme on the performance of the parallel light contribution computation phase (Phase 4).

N	P	Contribution Computation Time (secs) form factor vector circulation	Contribution vector circulation	percent decrease
522	4	0.032	0.032	0.00
	8	0.047	0.041	12.77
	16	0.067	0.051	23.88
1412	4	0.097	0.092	5.15
	8	0.120	0.108	10.00
	16	0.162	0.117	27.78
5648	4	0.405	0.370	8.64
	8	0.466	0.414	11.16
	16	0.614	0.428	30.29

Figure 3 illustrates the overall efficiency curves of the parallel progressive radiosity algorithm. Note that, global shooting patch selection, scattered decomposition and contribution vector circulation schemes are used in Phases 1, 2 and 4, respectively, in order to obtain utmost parallel performance. As is seen in Fig. 3, efficiency decreases with increasing P for a fixed N. There are two main reasons for this decrease in the efficiency. The first one is the slight increase in the load imbalance of the parallel hemicube production phase with increasing P. The second, and the more crucial reason is the modification introduced to the original sequential algorithm for the sake of parallelization. As is discussed in Section 3, this modification increases the total number of shooting patch selections required for convergence in comparison with the sequential algorithm.

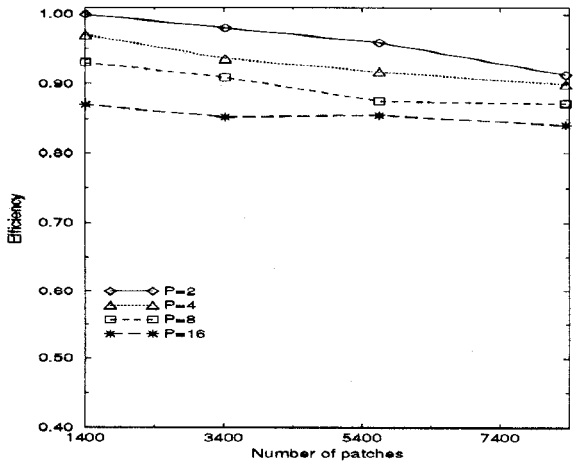

Figure 3: Overall efficiency of the parallel solution

5 Conclusion and Future Work

In this paper, a parallel progressive radiosity algorithm is proposed for ring-connected multicomputers and implemented on a ring-embedded Intel's iPSC/2 hypercube computer. The proposed parallel algorithm utilizes a synchronous scheme based on static task assignment. Experimental results show that scattered decomposition of the scene geometry yields adequate load balance during parallel hemicube production computations. Circulation of partial contribution results instead of the form-factor vectors is proved to decrease the total volume of concurrent communication by an asymptotical factor. Experimental results show that global shooting patch selection yields much better performance than local shooting patch selection as is expected.

Modification of the original progressive radiosity for the sake of efficient parallelization is experimentally found to yield good results. The performance of this modification is expected to increase with decreasing tolerance values which necessitate larger number of iterations for convergence.

6 Acknowledgement

We would like to acknowledge Guy Moreillon for his house interior model. The house data is available through anonymous ftp at site gondwana.ecr.mu.oz.au.

References

[1] Baum, Daniel R., James M. Winget, "Real Time Radiosity Through Parallel Processing and Hardware Accelaration", Proceedings of the 1990 Symposium on Interactive 3D Computer Graphics, In Computer Graphics, Vol.24, No.2, 1990, pp 67-75.

[2] Chalmers, Alan G., Derek J. Paddon, "Parallel Processing of Progressive Refinement Radiosity Methods", Proceedings of the Second Eurographics Workshop on Rendering, Barcelona, Spain, May 1991.

[3] Cohen, Micheal F., Donald P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments", Proceedings of SIGGRAPH '85 (San Fransisco, California, July 1985). In Computer Graphics, Vol.19, No.3, 1985, pp 31-40.

[4] Cohen, Michael F., Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation", Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1988). In Computer Graphics, Vol.22, No.4, 1988, pp 75-84.

[5] Drucker, Steven M. and Peter Schröder, "Fast Radiosity Using a Data Parallel Architecture", Proceedings of the Third Eurographics Workshop on Rendering, Bristol, England, May 1992, pp 247-258.

[6] Feda, Martin, Werner Purgathofer, "Progressive Refinement Radiosity on a Transputer Network", Proceedings of the Second Eurographics Workshop on Rendering, Barcelona, Spain, May 1991.

[7] Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, "Modelling the Interaction of Light Between Diffuse Surfaces", Proceedings of SIGGRAPH '84 (Boston, Massachusetts). In Computer Graphics, Vol.18, No.3, July 1984, pp 213-222.

[8] Puech, Claude, Francois Sillion, Cristophe Vedel, "Improving Interaction with Radiosity-based Lighting Simulation Programs", Proceedings of the 1990 Symposium on Interactive 3D Computer Graphics, In Computer Graphics, Vol.24, No.2, 1990, pp 51-57.

[9] Recker, Rodney J, David W. George, Donald P. Greenberg, "Acceleration Techniques for Progressive Refinement Radiosity", Proceedings of the 1990 Symposium on Interactive 3D Computer Graphics, In Computer Graphics, Vol.24, No.2, 1990, pp 59-66.

[10] Varshney, Amitabh and Jan F. Prins, "An Environment Projection Approach to Radiosity for Mesh-Connected Computers", Proceedings of the Third Eurographics Workshop on Rendering, Bristol, England, May 1992, pp 271-281.