Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer

François X. Sillion, Jean-Marc Hasenfratz

iMAGIS *- GRAVIR/IMAG
INRIA Rhône-Alpes
ZIRST, 655, avenue de l'Europe
38330 Montbonnot Saint Martin
France

Abstract

We introduce a simple, yet efficient extension to the hierarchical radiosity algorithm for the simulation of global illumination, taking advantage of a distributed shared memory (DSM) parallel architecture. Our task definition is based on a very fine grain decomposition of the refinement process at the level of individual pairs of hierarchical elements, therefore allowing a very simple implementation from an existing code with minimal modifications. We describe a generic refinement scheme based on a *scheduler*, allowing both easy parallelization and reordering of refinement tasks, which is useful for interactive and user-driven applications. We show that a very simple task grouping mechanism suffices to avoid excessive time waste in synchronization. Results obtained on an SGI Origin computer with 64 processors validate the approach, with excellent speedups using the full capacity of the machine.

1 Introduction

The radiosity method is a numerical simulation technique capable of determining the distribution of global illumination in a three-dimensional scene composed of diffuse reflectors [SP94]. Recent advances, in the form of hierarchical (wavelet) formulations and the introduction of clustering techniques, have made it possible to compute radiosity solutions in large scenes. Still these hierarchical algorithms are relatively slow on complex industrial scenes: usable solutions can be computed in tens of minutes, but very high-quality solutions typically require hours of calculation [HDSD99].

It is therefore quite natural to use parallel computers to reduce computation times – a state of the art is proposed by [RCJ98]. Two major approaches have been studied in previous work, corresponding to different parallel architectures and to different granularities: clusters of independent processors, or massively parallel machines with distributed shared memory. We review below the most significant approaches, and note that the high price of large parallel computers can only be accepted if very good speed-ups are obtained.

*iMAGIS is a joint research project of CNRS, INRIA, INPG and UJF.

Different authors describe parallel implementations of hierarchical radiosity on clusters of workstations. Funkhouser describes an algorithm where multiple hierarchical radiosity solvers work in parallel [Fun96]. A master process distributes sets of polygons over the set of workstations to determine an approximate and partial radiosity solution. The master then collects and merges solutions. This mechanism is iterated until convergence. This approach is well adapted to very complex scenes which could not be duplicated on all processors, provided visibility heavily restricts the potential interactions between subsets. This is particularly true of architectural scenes, with appropriate visibility preprocessing. The validation is done by tests over eight SGI workstations on the Soda Hall model with a speed-up of 5.5. Feng *et al.* also exploited the spatial coherence of the scene with 3D cells visible sets [FY97]. Tests on 8 DEC 3000 workstations with the PVM (Parallel Virtual Machine) programming environment show that the speed-up degrades with the number of processors due mostly to the Ethernet connection.

There is not as much work on hierarchical radiosity based on distributed shared memory. Bohn *et al.* proposed a parallel hierarchical radiosity approach on a *Connection Machine 5*, with a relatively modest speedup of 8.4 on 64 processors [BG95].

Renambot *et al.* proposed a parallel hierarchical radiosity based on a geometrical splitting of the scene. The radiosity is computed within each sub-scene. Exchange of energy between sub-scenes is performed by means of virtual interfaces and visibility masks. The size of sub-scenes can be adapted in order to fit into cache or local memory [RAPP97]. This algorithm was tested on a SGI Origin2000 with 32 processors. An accurate and interesting analysis of the hardware counters of the R10000 processor is presented. This seems to be a good approach, but requires a good knowledge of the different parameters to set and a very specific implementation. Cavin *et al.* proposed a parallel shooting wavelet radiosity algorithm on a large number of processors [CAP98]. A precise study of load balancing is proposed and a well adapted version for the Origin 2000 is presented in [Cav99]. Results show a good behavior for up to 32 processors (speedup of 24), with an unexplained but severe degradation afterwards. Singh *et al.* [SGL94] place pairs of patches which could interact into queues. Every processor has its own queue. When the treatement of a pair produces sub-patches, there are enqueued on the same processor. When a queue is empty, the associated processor steals tasks from other processors. Speed-up seems to be very good but tests are made only on one scene with a small number of polygons (174). A precise memory cache study is proposed, but the used of a (very) small scene makes conclusions difficult to generalize.

With the development of off-site computation facilities and companies, more and more opportunities exist to perform heavy simulations at specialized sites. Therefore the issue of acquisition cost for large parallel computers such as the SGI Origin is largely solved, and the development of an efficient parallel algorithm for hierarchical radiosity becomes of interest to a large user community. We placed our work in the context of such a machine, with tests on a 64-processor Origin2000, and aim to provide a simple and efficient algorithm, allowing the easy adaptation of existing codes.

The paper is organized as follows. The next section briefly recalls important mechanisms of hierarchical radiosity. Section 3 describes the extension of hierarchical radiosity into a parallel refinement algorithm. Results and discussion are in section 4 and finally, section 5 concludes and presents future work.

2 Hierarchical Radiosity

To understand our parallel implementation of hierarchical radiosity, we briefly describe the sequential version of the algorithm. The goal of the method is to compute a suitable approximation of the lighting distribution in the scene, by projecting it onto a set of basis functions. Basis functions, and the surface elements that form their support, are arranged in a hierarchy.

The core of the algorithm consists of refining the set of interactions between surface elements (although formally the interactions are between basis functions, we will refer to surface elements –equivalent to constant basis functions–for a more intuitive discussion). Refinement proceeds by pushing interactions down the hierarchy, until the radiosity exchange is "sufficiently well" approximated. The quality of an implementation largely rests on the choice of refinement criteria, but this issue is orthogonal to the parallelization method described here.

Interactions can be represented explicitly using "links", where a link embodies the impact of a surface element on another. The set of links attached to an element therefore represents the various sources of illumination for this element. The refinement process operates on a given set of links (initially and when we use clustering, a single link represents all the energy transfered in the scene [Sil95]), evaluates their quality using estimates of visibility, geometrical and energetic considerations, and decides for each link to either keep it or replace it with links at lower hierarchical levels.

Light energy can be transfered across the resulting set of links, iteratively until a global solution is obtained. Note however that since contributions are "gathered" at all levels of the hierarchy, a consolidation pass ("push-pull") can be needed at each iteration to ensure each element has a consistent view of all energy received higher and lower in the hierarchy. A typical execution of the program consists of several refinement iterations, each of which executing a refinement stage, followed by an energy propagation stage. Such iterations are useful because the refinement criterion uses the current estimate of radiosity in its decisions, and this estimate improves with each iteration. Note however that very few refinement iterations usually suffice to reach a high-quality solution (our results were computed for 4 iterations): they should not be confused with shooting iterations (e.g. in [Cav99]) which only compute the contribution of a single object, therefore requiring thousands of iterations for convergence.

Links therefore collectively represent the interactions in the scene, and are typically very numerous, placing heavy load on memory resources. As an alternative, shooting methods do not store links but instead propagate energy immediately upon calculation of transfer coefficients (form factors) [SSSS98]. The disadvantage is that links must then be recomputed in subsequent iterations, therefore it can be beneficial to keep some of them [GD99], using heuristic criteria.

Since the energy gathering and pushpull stages are easily parallelized (they basically consist of a simple array traversal), we focus in this paper on the hierarchical refinement stage.

3 Extension of Hierarchical Radiosity into a Parallel Refinement Algorithm

3.1 General considerations

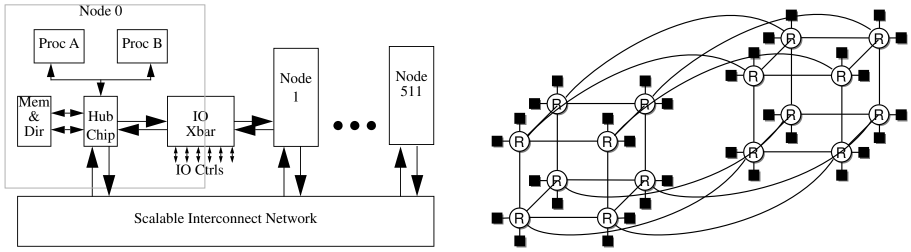
From the survey of the various approaches to the parallelization of radiosity algorithms, we observe the classical distinction between algorithms that explicitly manage data exchange (message-passing algorithms), and shared memory algorithms operating on a single data space. In practice, the latter class actually resorts to the operating system to implement virtual, or distributed, shared memory across the processors.

Algorithms assuming shared memory are obviously easier to implement, since data access is only an issue in terms of concurrency, therefore only requiring some synchronization or locking mechanisms. This is in contrast to distributed algorithms where low-level data management must be integrated in the algorithm. In the context of radiosity calculations, the global nature of light propagation (where each object can potentially illuminate many other objects in different areas of the scene) makes it very difficult to organize and monitor data locality, unless a very strong spatial structure is present as in some architectural scenes [Fun96]. Distributed Shared Memory (DSM) systems therefore appear particularly suited to radiosity calculations.

Even for the simpler shared memory case, however, a principal difficulty remains the segmentation of the work into tasks that can be efficiently distributed to the processors. A major factor influencing the overall efficiency is the granularity of the chosen tasks. Ideally, we want to divide the work into tasks of uniform complexity, to avoid situations where one processor is still working on its task while all others have finished. In the case of hierarchical radiosity calculations, the complexity of the calculations is not known in advance, since we precisely aim to adjust the effort spent on each radiant interaction, keeping it to the minimum needed to achieve the desired accuracy. The computation is therefore very dynamic, generating more calculations in areas of high importance.

Cavin *et al.* described a successful parallel implementation of wavelet radiosity on a DSM computer [CAP98, Cav99], and actually report that some of their "iterations" take hours, while others take only a few seconds. Furthermore, because of the large granularity of their algorithm they had to resort to complex operations on their data structures in the form of temporary copies and "lures", to avoid wasting time in synchronization locks. Therefore, using a fine granularity with very lightweight tasks seems a more promising response to the load balancing issue, assuming tasks can be quickly assigned to idle processors. A finer task segmentation would be difficult in a shooting wavelet radiosity approach where input surfaces are used as top-level shooting objects. By contrast, our use of a *complete* hierarchy in the scene, made possible by the clustering algorithm, allows us to consider the entire radiosity solution phase as a sequence of atomic refinement operations of very small, and nearly uniform, complexity. In essence, energy transfers (modeled either as link refinements or shooting operations) can be performed at any level of the hierarchy.

In summary, we chose a very fine granularity at the level of individual interactions involving an emitter-receiver pair: a task consists of treating such an interaction, which means either establishing a link (transferring energy, in a shooting approach) or deciding to reconsider the interaction at a finer level. Note however that in the latter case, several new tasks are created,



a. Block diagram. b. Topology of the DSM Origin 2000.

Figure 1: Scalable DSM Origin 2000 (*figures extracted from [LL97]*)

and the corresponding effort is therefore not part of the original task.

3.2 The ccNUMA architecture

To validate our approach, we conducted tests on an SGI Origin 2000 computer. The Origin is a scalable multiprocessor computer (up to 1024 processors) with Distributed Shared Memory (DSM), based on the SN0 (Scalable Node 0) architecture [LL97]. Figure 1 a. describes this architecture. The basic building block is the node board, composed of two MIPS R10000 processors, each with separate 32 KB first level (L1) instruction and data caches on the chip with 32-byte cache line, and a unified (instruction and data), commonly 4 MB, two-way set associative second level (L2) off-chip cache with 128-byte cache line. Each node contains 64MB to 4Go of main memory, accessible through a custom circuit called the hub. Large SN0 systems are built by connecting the nodes together via a scalable interconnection network. Connecting two nodes is done by connecting their hub chips through a router, which can be itself connected up to six hubs or other routers. Figure 1 b. shows the topology of the 64 processor Origin 2000 used for this paper.

People with extensive experience in using this ccNUMA architecture report that it has a very good properties of data locality [CAP98, Cav99] and is well adapted for a hierarchical radiosity algorithm.

3.3 Work flow of the refinement stage

The hierarchical radiosity algorithm lends itself well to a recursive implementation, due to the hierarchical structure of mesh elements and wavelet basis functions. However as discussed above such recursive implementations are prone to severe imbalance, as some interactions will require much more effort than others.

We introduce a generic mechanism for radiosity calculations, which we call a *scheduler*. The scheduler is basically a repository of potential links, or transfer interactions, with appropriate methods to request one or several of these links, and to submit new links. The set of links in the scheduler at any time therefore represent the set of pending tasks. A minimal modification of a hierarchical radiosity code is sufficient to use the scheduler: we basically reorganize the recursive traversal of the set of interactions, with the recursive creation of lower-level interactions, into a loop that continually requests a task from the scheduler, makes a decision about its subdivision and potentially submits resulting new tasks to the scheduler.

It is easily seen that a scheduler based on a stack data structure simply mimics the recursive behavior of the original algorithm. However, and even for purely sequential implementations, there are several advantages to using the scheduler. First, the scheduler offers a global view of all pending tasks at any given time. This allows more global decisions to be made during the course of the solution. Second, task extraction can be made according to various selection criteria, with maximal flexibility. For instance, the scheduler can implement a priority list, with criteria such as error estimates ("refine links with most error first"), importance or user-defined priorities. This is especially useful for interactive design sessions where the user might shift focus from one area of the scene to another, or modify the scene [DS97]. In such cases the ordering of tasks can be dynamically modified.

In the DSM model, a unique scheduler exists in memory, and we shall see later how to synchronize accesses to its data from the multiple processes.

Architecture of the radiosity program

The architecture of our radiosity program is depicted in Figure 2. We employ the simple *sproc* system call to create threads sharing their entire address space. The main process deals with the graphical user interface (for interactive sessions) and high-level control of the computation (running multiple iterations, monitoring convergence). Each iteration is handled by the solver process, which itself controls a number of identical refinement processes (refiners). The solver process is responsible for the initial loading of the scheduler, with all existing links at the start of the iteration. All refiners continuously obtain tasks from the scheduler, perform the corresponding refinement decisions, and add new tasks to the scheduler as needed. Note that there is no scheduling process, but rather a data structure whose access is controlled by a synchronization mechanism as we will see below.

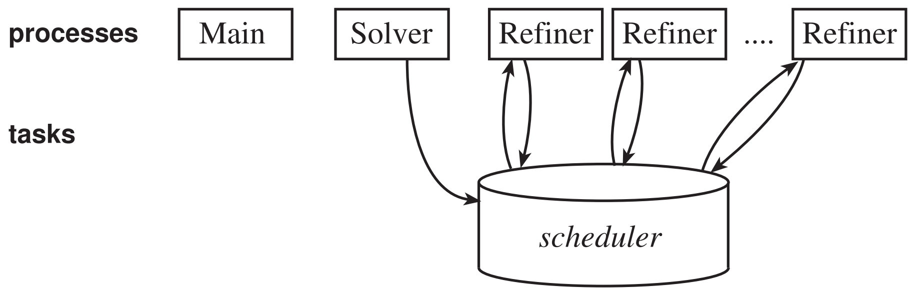


Figure 2: *Organization of our radiosity code. We use* $n+2$ *threads for n simultaneous refiners. The solver feeds the scheduler structure with an initial set of tasks (links), and each refiner requests tasks from the scheduler and returns new tasks as appropriate.*

3.4 Synchronization

A mentioned above, our architecture accommodates a number of refinement processes operating on a shared address space. Appropriate synchronization and locking mechanisms are needed to ensure complete data consistency. We found it sufficient to implement the following three locking mechanisms, isolating the three corresponding code fragments.

1. scheduler: since all refiners access the same pool of tasks through the scheduler interface, both the extraction and the incorporation of tasks should be protected against concurrent access. A semaphore with a single access right is used to control the operation of the scheduler.

2. hierarchical data structure. The hierarchy of elements representing the objects in the scene must be guarded against concurrent modifications of any given element. However, simultaneous modifications of different elements are allowed. Therefore we can use a pool of semaphores indexed by a hashing function to avoid congestion. The critical code section is the subdivision of a surface element.

3. Interactions. When using links to represent transfer interactions, each element possesses a list of links, which must be guarded against concurrent modification (links can either be added as a result of refinement, or deleted when they are selected for refinement and "pushed down" in the hierarchy). For the shooting version of the algorithm, which does not use links, the equivalent operation is the update of radiosity values. Similar to the previous case, we can use a pool of semaphores to avoid congestion.

3.5 Performance

Our scheduling mechanism, based on the acquisition of tasks from the scheduler, leaves only a small number of sources for performance degradation. These are mainly (a) the time spent waiting for semaphore acquisition in the various locks described above, and (b) performance issues at the system level, in particular concerning memory access across the different caches and nodes of the Origin computer. We do not consider the second problem at this time, and focus on synchronization performance.

Clearly the code implementing task extraction and incorporation in the scheduler is critical, as it can be performed for a single refiner at once. Therefore it is important to minimize the corresponding effort. Since the order in which links are refined is not critical in general, we optimized our code to operate on blocks of tasks, therefore in essence grouping a set of links to be refined. It should be noted that this conserves the desired property that all tasks be of similar complexity, and that the grouping is arbitrary and does not necessarily correspond to high-order hierarchy elements. As expected, the overall performance is improved by using blocks of links, since the time needed to operate the scheduler is not negligible with respect to the time for refining a single link. Quite naturally, the size of the block is not important as long as it provides "enough work" to overshadow the scheduler operation: Figure 7 shows that blocks of 10 links already provide a significant improvement, whereas increasing to 100 links make no difference.

Using blocks of links takes care of the acquisition of tasks by each refiner: for the submission of new tasks (as a result of refinement), we use a buffering mechanism. New tasks are accumulated in a table that is unique to each process, and only after all links in the block have been refined is this table integrated in the scheduler (with appropriate synchronization).

For the other critical code portions, namely the modification of the hierarchy through surface element splits, and the representation of energy exchanges (links or radiosity updates), we found that using a pool of semaphores indexed by a hashing function works very well and basically removes all contention. We used a table of about twice the number of refiners, and an indexing function based on the memory address of the concerned elements.

4 Results and Discussion

4.1 Test Scenes Chosen

We performed our tests on three different scenes, shown in Figure 3. Two of them are rather large industrial-type models while the third one is more artificial.

- **AIRCRAFT** (184,456 original polygons)
  Model of an aircraft cabin (courtesy of LightWork Design Ltd). All objects have been tessellated into (rather small) triangles to account for the rounded shapes.
- **VRLAB** (30,449 original polygons)
  A virtual reality lab with two floors and mostly overhead lighting (courtesy of Fraunhofer Institut für Graphische Datenverarbeitung). This scene has a mixture of large polygons, likely to be subdivided, and very small patches (on chairs and desktop computers).
- **OFFICE** (5,260 original polygons)
  A model of a simple office scene. This model is much smaller than the other three and is provided to show that clustering performs well on this kind of small scene usually found in the literature.

4.2 Measurements

We tested our algorithm on a ccNUMA Silicon Graphics Origin 2000 computer [LL97] with 64 processors. In fact, we limit our tests to 40 processors to avoid locking the Origin for our own usage! Each processor is a 195 MHz R10000 with L1 data cache of 32 Kbytes, 4 Mbytes of secondary unified data cache L2 and 384 MBytes of local memory.

Refinement time was measured with the *times* system call, which returns clock ticks. Memory access time, cache usage etc. were measured through the 31 hardware counters of the R10000 using the *perfex* software tool.

Note that we have replaced the standard SGI memory allocation functions by the corresponding functions in the *GNU library* (*GNU C Library, Wolfram Gloger and Doug Lea*) to avoid thread-safe memory manipulation locks.

VRLab.

Office.

Aircraft.

Figure 3: The three test scenes used.

For all scenes we chose execution parameters to require about 5 hours of total CPU time, which corresponded to different levels of refinement for different scenes (with for instance very many links in the Office scene which is comparatively much simpler).

4.3 CPU time and speed-up

Figure 4 shows the total time (summed over all refiners) used for each refinement iteration, for different numbers of processors[1]. We clearly see that the time is constant for the VRLab and quite constant for the Office. The Aircraft has a "strange" behavior: time increases with the number of processors, at different rates for different iterations (and with a perfect constant time for the first iteration).

In Figure 5a. we show the speed-up for each scene. Logically, VRLab has a quite perfect speed-up of 39.4 on 40 processors (in fact, we have obtained a speed-up of 49.2 on 50 processors) and the Office has a speed-up of 25.7 on 30 processors. The aircraft speedup is not as good, reaching 24 for 40 processors. We can see in Figure 5b. that the speed-up of Aircraft decreases with the number of processors for the last three iterations: respectively, we obtain speed-ups of 35.1, 23.4 and 17.3 on 40 processors for iterations 2, 3 and 4.

To explain this behavior, we have investigated memory usage, thinking the problem may be due to cache performance or data locality issues. However as shown in Figure 6 this is not a

[1] A technical problem prevented us to obtain results for the Office scene with 40 processors. However it is clearly not related to our algorithm.

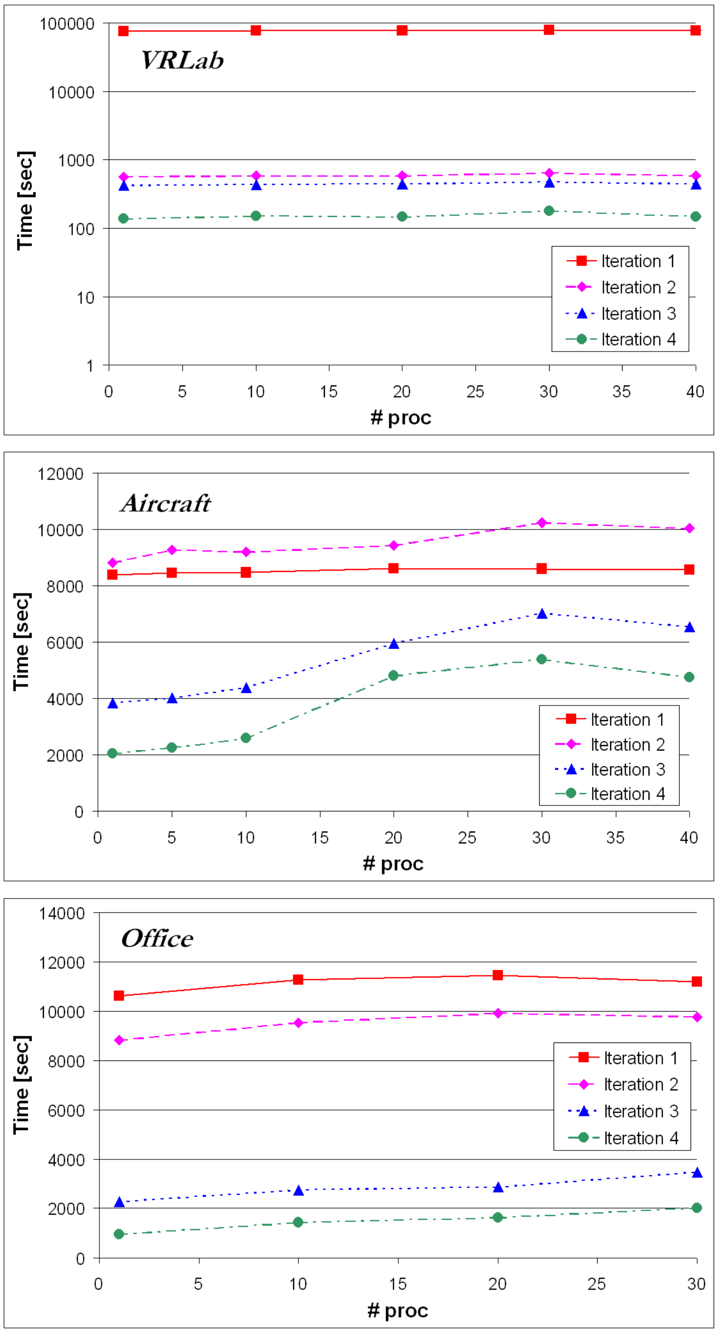
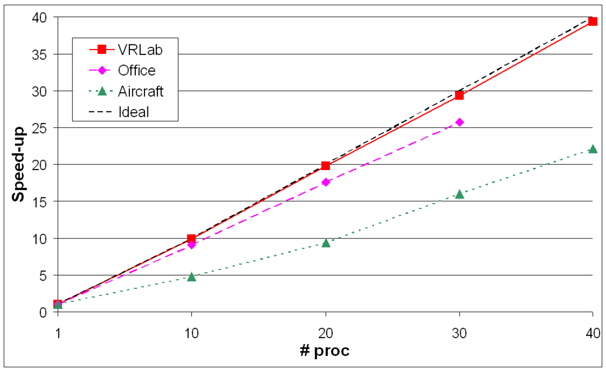
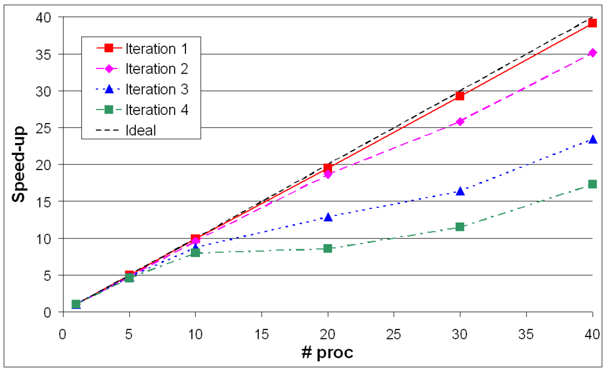


Figure 4: CPU Refinement time for the three test scenes.



a. Speed-up for all test scenes
(four iterations).



b. Aircraft speed-up for each iterations.

Figure 5: Speed-up.

good explanation: we have plotted both the number of links created (and stored, in our implementation), as well as the total memory usage, for each scene and at the end of each iteration (cumulative). We clearly see that memory consumption is directly correlated to the number of links (not surprisingly, following analyses of [WH97, SSSS98]). However the aircraft scene has the smallest number of links and memory size, and fits easily in the local memory of a single processor board. Conversely, Office uses the most memory but exhibits a perfect speedup even for subsequent iterations. At this point we are looking for an explanation to this behavior. We also looked at the percentage of total time spent accessing memory, as provided by *perfex*, but for the aircraft scene this percentage *decreases* (from about 65% to about 45 %) as we increase the number of processors... In fact, we have analyzed all 31 hardware counters without discovering a satisfactory explaination of this behaviour.

We note however that if a memory locality issue can be identified, the *scheduler* could easily incorporate other criteria to better balance the data over the processors by ordering links differently. This does not appear to be necessary given the good results already obtained, unless maximal performance is absolutely necessary.

Finally we present in Figure 7b results for different block sizes in the tasks distributed by

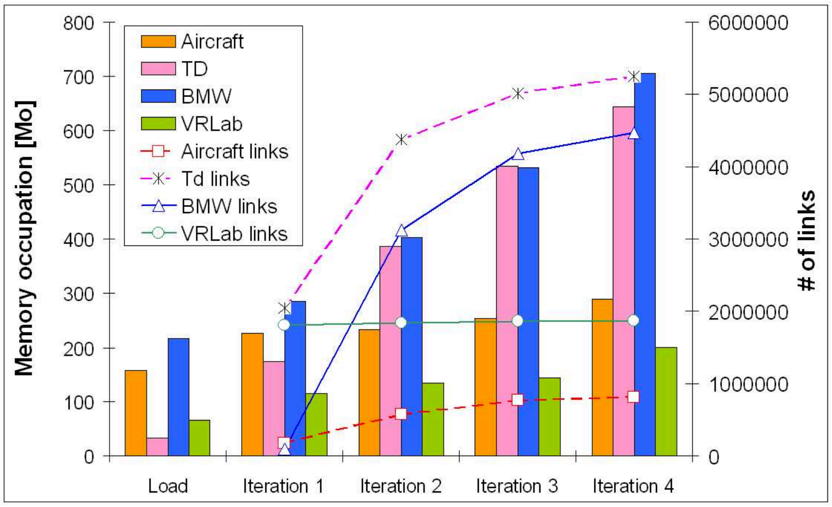


Figure 6: *Memory used before and during the iterations.*

the scheduler. As we increase this size from 1 to 10, we observe a natural improvement, but increasing it to 100 offers no more improvement.

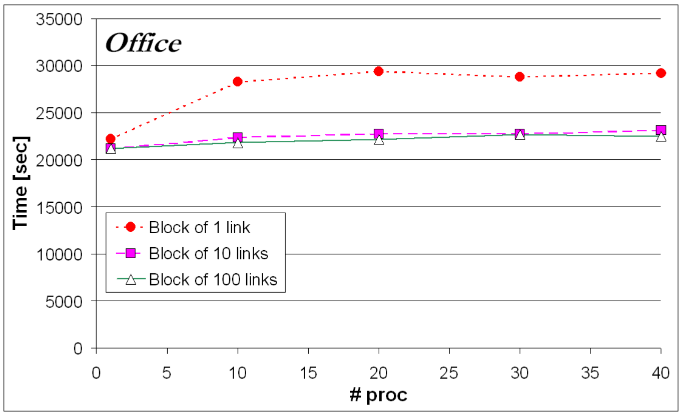


Figure 7: *Influence of the size of link blocks on overall CPU time.*

5 Conclusions and future work

We have presented a parallel algorithm for hierarchical radiosity calculations on a computer with distributed shared memory. The work is divided into very simple atomic tasks, consisting of the refinement decision on a single interaction (or a small group of such interactions). The distribution is easily managed with a single *scheduler* structure, with appropriate synchronization. Incidentally the use of a scheduler is beneficial in many radiosity applications including interactive steering by the user. The resulting organization encapsulates the parallel behavior of the algorithm into the scheduler and thread management portions of the code, and allows full flexibility in the rest of the simulation, including choice of wavelet bases, shooting or link-based algorithms, visibility determination techniques and refinement oracles. Therefore our proposed algorithm can be very easily implemented on top of an existing radiosity simulation code.

Our results indicate good to excellent speedups depending on the scenes, for up to 40 processors. This is particularly encouraging considering that no effort was made to localize tasks and thereby improve memory access time. Future work includes the understanding of the peculiar behavior observed for the aircraft scene, in which speedups are consistently decreasing for successive iterations of the refinement algorithm. While our analyses so far have not yet identified a data locality problem, we note that should such an issue be identified, the scheduler mechanism can incorporate any ordering based for instance on data locality. Therefore if we can predict a grouping of link refinement tasks that improves performance it can readily be implemented in the scheduler.

6 Acknowledgments

Peter Kipfer contributed to the design and early implementation of this work. We would like to thank the Centre Charles Hermite for providing us access to its computational resources, and Laurent Alonso for his advice on performance questions. This work was supported in part by the European Union's ESPRIT project #24944, ARCADE ("Making Radiosity Usable").

References

[BG95] Christian-A. Bohn and Robert Garmann. A Parallel Approach to Hierarchical Radiosity. In V. Skala, editor, *Proceedings of the Winter School of Computer Graphics and CAD Systems '95*, pages 26–35, Plzen, Czech Republic, February 1995. University of West Bohemia.

[CAP98] Xavier Cavin, Laurent Alonso, and Jean-Claude Paul. Parallel wavelet radiosity. In *Proceedings of the Second Eurographics Workshop on Parallel Graphics and Visualisation*, pages 61–75, Rennes, France, September 1998. Eurographics.

[Cav99] Xavier Cavin. Load balancing analysis of a parallel hierarchical algorithm on the origin2000. In *Fifth European SGI/Cray MPP Workshop*, Bologna, Italy, September 1999.

[DS97] George Drettakis and François Sillion. Interactive update of global illumination using a line-space hierarchy. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '97 (Los Angeles, CA), pages 57–64. ACM SIGGRAPH, New York, August 1997.

[Fun96] Thomas A. Funkhouser. Coarse-Grained Parallelism for Hierarchical Radiosity Using Group Iterative Methods. In *Computer Graphics Proceedings, Annual Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings)*, pages 343–352, 1996.

[FY97] Chen-Chin Feng and Shi-Nine Yang. A parallel hierarchical radiosity algorithm for complex scenes. In *Proceedings of the Third Parallel Rendering Symposium (PRS '97)*, pages 71–78, Phoenix, AZ, October 1997. IEEE Computer Society.

[GD99] Xavier Granier and George Drettakis. Controlling memory consumption of hierarchical radiosity with clustering. In *Graphics Interface*, pages 58–65, June 1999.

[HDSD99] Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, and George Drettakis. A practical analysis of clustering strategies for hierarchical radiosity. In *Computer Graphics Forum (Proc. Eurographics '99)*, volume 18(3), September 1999.

[LL97] James Laudon and Daniel Lenoski. The sgi origin: a ccnuma highly scalable server. In *Proceedings of the 24th international symposium on Computer architecture*, pages 241–251, Denver, CO USA, June 1997. ACM.

[RAPP97] Luc Renambot, Bruno Arnaldi, Thierry Priol, and Xavier Pueyo. Towards efficient parallel radiosity for dsm-based parallel computers using virtual interfaces. In *Proceedings of the Third Parallel Rendering Symposium (PRS '97)*, pages 79–86, Phoenix, AZ, October 1997. IEEE Computer Society.

[RCJ98] E. Reinhard, A. Chalmers, and F. Jansen. Overview of parallel photo-realistic graphics. In *Computer Graphics Forum (Proc. Eurographics '98)*, pages 1–25, 1998.

[SGL94] Jaswinder P. Singh, Annop Gupta, and Marc Levoy. Parallel Visualization Algorithms: Performance and Architectural Implications. *IEEE Computer*, 27(7):45–55, July 1994.

[Sil95] François X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3), September 1995. (a preliminary version appeared in the fifth Eurographics workshop on rendering, Darmstadt, Germany, June 1994).

[SP94] François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann publishers, San Francisco, 1994.

[SSSS98] M. Stamminger, H. Schirmacher, P. Slusallek, and H.-P. Seidel. Getting rid of links in hierarchical radiosity. *Computer Graphics Forum (Proc. Eurographics '98)*, 17(3):C165–C174, September 1998.

[WH97] Andrew Willmott and Paul Heckbert. An empirical comparison of progressive and wavelet radiosity. In Julie Dorsey and Phillip Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 175–186. Springer Wien, 1997. ISBN 3-211-83001-4.