Parallel Wavelet Radiosity

Xavier Cavin, Laurent Alonso, Jean-Claude Paul

Parallel Wavelet Radiosity

Xavier Cavin, Laurent Alonso and Jean-Claude Paul
{Xavier.Cavin,Laurent.Alonso,Jean-Claude.Paul}@loria.fr

LORIA - INRIA Lorraine,
Campus scientifique, BP 239,
54506 Vandœuvre-les-Nancy Cedex, France

Abstract: This paper presents parallel versions of a wavelet radiosity algorithm. Wavelet radiosity is based on a general framework of projection methods and wavelet theory. The resulting algorithm has a cost proportional to $O(n)$ versus the $O(n^2)$ complexity of the classical radiosity algorithms. However, designing a parallel wavelet radiosity is challenging because of its irregular and dynamic nature. Since explicit message passing approaches fail to deal with such applications, we have experimented various parallel implementations on a hardware ccNUMA architecture, the SGI Origin2000. Our experiments show that load balancing is a crucial performance issue to handle the dynamic distribution of work and communication, while we do make all reasonable efforts to exploit data locality efficiently. Our best results yield a speed-up of 24 with 36 processors, even when dealing with extremely complex models.

1 Introduction

Radiosity methods have been proven to be an efficient mean to simulate the inter-reflections of light in Lambertian (diffuse) environments. The radiosity - power per unit area $[W/r^2]$ - on a given surface, is governed by an integral equation which can be solved by projecting the unknown radiosity function onto a set of basis functions with limited supports, resulting in a set of $O(n^2)$ interactions. The wavelet theory introduced by [5] and [9], has been shown to reduce the required interactions to $O(n)$. Unfortunately, wavelet radiosity still requires too much computation time when dealing with extremely complex models (several millions of polygons with physical properties), even on modern workstations [3].

Designing a parallel wavelet radiosity algorithm involves dealing with complex issues, such as:

- the physical domain being simulated is typically non-uniform, which has implications for both load balancing and communication;
- work and communication change dynamically across the computation of the solution because of the hierarchical nature of wavelets.

Most previous works on parallel radiosity are related to classical radiosity algorithms. Although the most recent of them [8] obtains quite good results on a SGI Origin2000 and yields radiosity computations for very large models, the implemented algorithm suffers from having a $O(n^2)$ complexity. Only a few papers address the design problem of parallel hierarchical algorithm. Zareski implemented in [12] a parallel version of the hierarchical radiosity algorithm on a network of workstations using a master-slave architecture, in which each slave performed surface-elements interactions for a separate subset of elements in the scene. Speed-up with this fine-grained approach was limited by both master processing bottleneck and the overhead of inter-process communication, resulting in longer execution times as the number of processors increases. Aiming to use a coarse-grained parallelism in a similar master-slave approach, [4] designed partitioning and scheduling algorithms that

allowed multiple hierarchical radiosity solvers to work on the same radiosity solution in parallel. In this way, the author achieved significant speed-ups, 65% to 75%, with a very large model, but the scalability of the parallelism had to stay moderate (less than 8 slave workstations). A more general study of hierarchical algorithms has been addressed in [11]. Experiments on the 48 processors Stanford DASH machine (a Cache Coherent Shared Address Space Multiprocessor) have yielded very good performance, although it was on a very small model (94 input polygons). When dealing with much larger environments (typically in the order of hundreds of thousands of polygons), the amount of communication changes the dimension of the problem.

In summary, all related works failed to provide a parallel response to the following goals combined:

- for an asymptotic radiosity algorithm,
- dealing with extremely large models,
- and effectively scalable to a large number of processors.

The parallel solution presented here addresses these three problems. Load balancing algorithms implemented on the SGI Origin2000 achieve significant speed-ups for the wavelet radiosity algorithm, running on 32 processors, for models composed of more than $100\ 000$ polygons of arbitrary geometry.

The organization of the paper is as follows. In Section 2, we present the mathematical framework of the wavelet radiosity method and a very efficient sequential implementation of this algorithm. Then, we present our parallel implementation strategy on a SGI Origin2000 in Section 3. We then describe our experiments in Section 4, results and discussion in Section 5. Finally, we conclude and present future works in Section 6.

2 Wavelet Radiosity

2.1 The Radiosity Equation

Given some physical assumptions, the radiosity equation can be formulated as follows. Let $\mathcal{M}$ denote the collection of all surfaces in an environment, which we assume to form an enclosure for simplicity. Let χ be a space of real-valued functions defined on $\mathcal{M} \times S^2$; that is, over all surface points and angular directions in the unit sphere S^2. Given the surface emission function $g \in \chi$, which specifies the origin and directional distribution of emitted light, we wish to determine the surface radiosity function $f \in \chi$ that satisfies:

$$f(\lambda, x) = g(\lambda, x) + k(\lambda, x) \int_{\mathcal{M}} G(x', x) f(\lambda, x')\, d\, x', \tag{1}$$

where:

- λ is the wavelength at which functions are computed,
- $k(\lambda, x)$ is the local reflectance function of the surface (*i.e.*, we suppose that the surfaces are ideally diffuse),
- $G(x', x) = \frac{1}{\pi} \cos\theta_{x'} \cos\theta_x\, r_{x'x}^{-2}\, v(x', x)$ is a geometry term consisting of the cosines made by the local surface normals with a vector connecting the two surface points x and x', the distance r between theses two points, and a visibility function v whose value is in $\{0, 1\}$ according to whether the line between the two points x and x' is obstructed or un-obstructed respectively.

Using the operator notation we can express the equation to be solved as $f = g + KGf$, which is a linear operator of the second kind, or more compactly as $Mf = g$.

Projection methods, whose role is to recast infinite-dimensional problems in finite dimensions, can be used to solve the radiosity equation. The idea is to construct an approximate solution from a subspace $\chi_n \subset \chi$, where the parameter n typically denotes the dimension of the subspace. In any case, each element of the function space χ_n is a linear combination of a finite number of basis functions $\{u_1, \ldots, u_n\}$.

Given this space of basis functions $\chi_n = \mathrm{span}(\mathrm{u}_1, \ldots, \mathrm{u}_\mathrm{n})$, we seek an approximation f_n from the space χ_n that is close to f. This is equivalent to determining n unknown coefficients $\{\alpha_1, \ldots, \alpha_n\}$ such that $f = \sum_{j=1}^{n} \alpha_j u_j$.

Projection methods select such approximations from χ_n by imposing a finite number of conditions on the residual error, which is defined by $r_n \equiv M f_n - f$.

Specifically, we attempt to find f_n such that r_n simultaneously satisfies n linear constraints, that is $\psi_i(r_n) = 0$, for $i = 1, \ldots, n$, where the ψ_i are linear functionals. Any collection of n linearly independent functionals defines an approximation f_n by "pinning down" the residual error with sufficiently many constraints to uniquely determine the coefficients. Note that functionals and basis functions together define a projection operator [1]. This way, we have:

$$\psi_i \left(M \sum_{j=1}^{n} \alpha_j u_j - g \right) = 0, \tag{2}$$

which is a system of n equations for the unknown coefficients $\{\alpha_1, \ldots, \alpha_n\}$. The quality of the approximation as well as the computations required to obtain it, depend on the approximation properties of the space χ_n, the choice of functionals, and finally the numerical techniques which are used to compute the matrix coefficients and solve the linear system. There are other sources of error due to imprecise geometry or boundary conditions [2].

2.2 Radiosity and Wavelet Theory

The wavelet radiosity algorithm is a special case of the formulation in which the matrix form of (2) is:

$$\begin{bmatrix} a(\phi_1,\phi_1) & a(\phi_1,\phi_2) & \cdots & a(\phi_1,\phi_n) \\ a(\phi_2,\phi_1) & a(\phi_2,\phi_2) & \cdots & a(\phi_2,\phi_n) \\ \vdots & \vdots & \ddots & \vdots \\ a(\phi_n,\phi_1) & a(\phi_n,\phi_2) & \cdots & a(\phi_n,\phi_n) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} < g,\phi_1 > \\ < g,\phi_2 > \\ \vdots \\ < g,\phi_n > \end{bmatrix}, \tag{3}$$

where (ϕ_i, ϕ_j) are wavelets.

As shown in [9], since wavelets can be used as bases for function spaces, a linear operator can be expressed in them. If this operator satisfies certain smoothness conditions - as the radiosity operator does - the resulting matrix is approximately sparse and the system can be solved asymptotically faster if only finite precision is required of the answer.

More precisely, the linear system in (3) has entries which are the coefficients of the kernel function with respect to some basis. In classical radiosity, the Galerkin method gives rise to a system relating all of these basis functions with each other resulting in a system of size $O(n^2)$. However, such bases possess properties which derive directly from the kernel itself. Using wavelets as basis functions the resulting matrix system is approximately sparse if the kernel is smooth. By ignoring some entries whose value is below some threshold the resulting linear system has only $O(n)$ remaining entries leading to fast solution algorithms. To realize an algorithm of $O(n)$ complexity, a function *oracle* is needed to help enumerate the important entries in the matrix system.

2.3 Sequential Implementation: The Candela Project

The Candela project was designed to provide a flexible architecture for testing and implementing new radiosity and radiance algorithms [7]. It was at the same time intended to be able to deal with real data (complex geometrical surfaces, accurate light sources models, real spectrum modeling) and to compute physically correct results. Candela is based on the Open Inventor library in order to get the required flexibility, both on inputs and algorithmical combinations.

Several sequential wavelet algorithms and accelerating techniques have already been implemented. This part is detailed in [3]. For the sake of completeness, we give here the main characteristics of the existing algorithms:

- the modeling is described using the Open Inventor file format:
  - polygonal surfaces ($\mathcal{M}$) can be of arbitrary geometry;
  - the spatial part of an emitter g is described using the C_γ representation;
  - the spectral distribution of a light source g or of a diffuse coefficient k of a surface can be coded in several function bases.
- the available bases of functions ϕ_i are the Haar basis, $\mathcal{M}_2$ and $\mathcal{M}_3$,
- the spectra λ are computed exactly and projected when the result is stored on a surface in a basis of functions which can be defined by the user.
- the visibility $v(x', x)$ can be accelerated using the hardware or a BSP (*i.e.*, a Binary Space Partition),
- the kernel coefficients $a(\phi(i), \phi(j))$ are computed by Gaussian quadrature,
- both geometry based and energy based *oracles* exist,
- two solvers are implemented, one implements a gathering iterative scheme and the other one a progressive shooting scheme. The links can be stored or recomputed each time there are needed.

Our sequential experiments have shown that the wavelet coding of the radiosity function is very important in order to obtain correct results, but that the links storage can quickly become a great bottleneck because of its huge memory requirements. The effect is even more annoying with the progressive shooting solver: in the first steps of this algorithm, a lot of energy is exchanged between surfaces, which leads to many links stored. Most of the time, these links are over refined, with respect to the energy exchanged in the next steps.

This over-cost disappears if the links are not stored, of course at the cost of longer computation times. However, this is currently the only way to deal with large scenes. In this case, the progressive shooting algorithm converges faster than the gathering algorithm, at least at the beginning of the computations, allowing to get results earlier. In conclusion, progressive shooting wavelet radiosity without links storage appears to be a very efficient sequential algorithm for complex scenes. It will be described in detail in Section 4.

Nevertheless, despite these choices of algorithms and accelerating techniques, computation times and memory requirements still remain too important for effective use on a single workstation, especially when simulated scenes are very large (the common case for architectural simulations). The alternate way to bypass the limitations of single workstations, is to turn towards parallelism. This will be the subject of the reminder of the paper.

3 Parallel Issues

3.1 General Considerations

Implementing some parallel algorithm can be done on two main kinds of architectures: cluster of workstations or shared memory supercomputer. Using the first approach, the parallel program has to be based on the message passing paradigm. This can be very attractive, because it can be applied both on a supercomputer or on a network of workstations (a very

common resource in today's research centers). In the other approach, communication is implicitely managed through shared variables, making algorithms implementation much easier. Shared memory supercomputers mostly use the Distributed Shared Memory (DSM) architecture (like the SGI Origin2000): the shared memory is distributed among processes and can be addressed transparently; its efficiency heavily depends on the caching of data accessed in a remote memory. Singh *et al.* show in [10] that shared address space is best suited for dynamic and irregular algorithms like the hierarchical radiosity.

As pointed out in [11], effective speed-up over the best sequential algorithm can be constrained by six kinds of overhead:

- inherently sequential code: this includes initialisation and termination phases;
- redundant work: this is a part of the work that have to de done by each process, while it would only be done once in the sequential algorithm;
- overhead of parallelism management: the distribution of work to processes can introduce extra costs;
- overhead of synchronization: this includes synchronization barriers and mutual exclusion locks;
- imbalanced distribution of work among processes: the *load balancing* problem;
- inter-processors communication and overhead of communication: this both concerns *data locality* and *false sharing*.

As far as scientific applications are concerned, the two key, but conflicting, aspects to address are load balancing [11] and data locality [8].

3.2 Parallelism and Radiosity

When designing a parallel program, an important aspect to determine is the level of parallelism (*i.e.*, which parts of the sequential program might be parallelized). Let us now focus on specific issues involved in the parallelization of radiosity algorithms by re-examining equation (1) terms:

- the first two terms $g(\lambda, x)$ and $k(\lambda, x)$ represent the inputs of the algorithm: they are composed of a spatial part and of a spectral distribution. They do not lead to any particular problem for parallelism;
- the third term $f(\lambda, x)$ is the radiosity function, that is to say the solution we aim to compute. We have chosen to code this function using wavelet bases: this gives very efficient sequential algorithms but has two drawbacks in parallelism. Contrarily to classical radiosity implementations, we are not able to allocate all the required memory at the beginning of the algorithm, but we need to make dynamic allocations. Moreover, it is very difficult to forecast how long an interaction between two surfaces will take to compute;
- the fourth term $G(x', x)$ which is mainly composed of the visibility function $v(x', x)$ is the first global term of the equation. In sequential implementations, many accelerating techniques may be used to compute the visibility, which represents a huge part of the overall computation [3]. We have chosen the well-known Binary Space Partitioning (BSP) structure for its efficiency. It is important to use the same technique in parallelism if we want to obtain efficient algorithms, though the storage of the BSP could become problematic as scene size grows. We have expected that even if this structure is too large to fit in the processor memory cache, the natural data locality of the visibility requests will not generate too many memory problems. Indeed when an energy transfer is done between two surfaces, we only have to test the visibility between points of these surfaces and we can hope that the same parts of the BSP will be traversed. It is the same if we can have each processe dealing with neighboring surfaces;

– the last part is the integral equation in itself: it codes the interactions between the different surfaces of the scene. Many sequential and parallel algorithms have been devoted to solving this complex integral equation. As we mentioned in Section 2, we have chosen to focus on a very efficient sequential algorithm allowing to deal with large real world scenes: the progressive shooting wavelet radiosity [3]. More precisely, we consider the version which does not store links between surfaces, both because of memory problems and because it would lead to really tricky problems in parallel algorithms. Furthermore, the visibility computations are done with the BSP structure, due to the "limitation" of our Origin2000 which has no graphic card.

The parallelization work is done on the solving part of the algorithm and will be the subject of the next paragraph.

3.3 Parallelization of the Progressive Shooting Wavelet Radiosity

The progressive shooting wavelet radiosity algorithm sequentially handles the most energetic emitter (either a direct light source or a reflecting surface) and propagates its energy to all scene receivers (surfaces), decomposing them both into smaller surface elements when needed. Unfortunately, the amount of work that will be required for a given emitter-receiver interaction is highly unpredictable and even depends on the resolution process in progress. These dynamic and unpredictable characteristics of the algorithm make it challenging to be parallelized in a well load balanced way.

Several granularities can be considered in order to get an efficient tasks decomposition of the problem:

– the finest granularity can be found inside an emitter-receiver interaction. In [11], a task can either be a surface-element or even an element-element interaction. Implementation with distributed task queues and clever task stealing allows a good control over load balancing while preserving enough data locality when dealing with a small number of surfaces;
– at the opposite, a very coarse granularity would be to define a task as the set of interactions between one (or several) emitters and all its associated receivers. This allows several emitters (one per process) to propagate energy to all the scene receivers in parallel. A given receiving surface can so be shot by two emitters at the same time and mutual exclusion is needed for accessing it;
– an intermediate (rather small) granularity is to consider a task as a standard surface-surface interaction. The way these tasks are distributed to processes leads to different constraints and algorithms. If emitters are processed one after the other, as in the sequential algorithm, energy propagation from an emitter to all its receiving surfaces is parallelized among the processes without restrictive access but is bounded by a synchronization barrier before dealing with the next emitter. If we see the problem as a global pool of tasks that processes have to execute, mutual exclusion is needed for accessing surfaces and a load balanced distribution algorithm has to be found.

We have chosen to test the intermediate surface-surface granularity which seemed to be best suited to our problem. We hoped it would be fine enough in the case of large scenes.

4 Experimentation

We focus in this section on the implemented parallelizations of the progressive shooting wavelet radiosity (with the different considerations of the previous sections) and on the different tests which we have performed.

We have parallelized the solver phase of this algorithm. Due to the differences between the emission of a light source on a surface and the redistribution of the energy between

two surfaces, this phase is partitioned for efficiency in two phases. We present below the pseudo-code of these two sequential algorithms for clarity in the following subsections.

```
1  Direct Illumination Algorithm:
2  begin
3     /* Get all the light sources. */
5     /* Get all the scene surfaces. */
7     foreach light do
8        foreach surface do
9           call illuminate(light, surface);
10       od od
11    foreach surface do
12       call pushPull(surface); od
13 where
14 proc illuminate(light, surface) ≡
15    call illuminateRecursive(light, surface); .
16 end
```

```
1  Intereflections Algorithm:
2  begin
3     /* Get all the scene surfaces. */
5     sortedList := ∅; /* A sorted list of energetic surfaces. */
6     foreach surface do
7        if hasEnergyToShoot(surface)
8           then insertSorted(surface, sortedList); fi
9     od
10    while (¬converged) do
11          shooter := removeFirst(sortedList);
13          foreach surface do
14             call shoot(shooter, surface);
15          od od
16    foreach surface do
17       call pushPull(surface); od
18 where
19 proc shoot(shooter, surface) ≡
20    call pushPull(shooter);
22    call remove(surface, sortedList);
24    call shootRecursive(shooter, surface);
26    call setEnergyToShoot(shooter, 0);
28    call insertSorted(surface, sortedList); .
29 end
```

4.1 Parallelization of Direct Illumination

The *direct illumination* algorithm is the easiest to be parallelized. The number of basic emitter-receiver interactions [1] is known in advance and does not change with time.

We choose to implement it with the emitter-surface granularity for its better compromise between ease of programming and expected efficacity. The coarser granularity is not interesting in this case, because the relatively small number of light sources compared to the number of processes would lead to severe load imbalance. On the other hand, exploiting the finest granularity does not seem necessary in the case of large scenes. We anticipated that the intermediate granularity would be enough to get good load balancing and implemented two algorithms with it.

[1] between a light source and a surface

Inner Loop The first of these algorithms is the parallelization of the inner of the two loops of the sequential algorithm: emitters are shot one after the other, and each shoot is done by all the processes in parallel. A given receiver can not be shot by two emitters and so requires no locking access. However, synchronization barriers are needed at the end of each shoot before processing the next one.

Single Queued To remove the potentially costly synchronization barriers, we can have the processes starting processing the next emitter instead of waiting for others to complete their job. This can be done in a dynamic tasking approach where the algorithm consists of a pool of tasks (here emitter-receiver interactions) processes have to execute. In that case, restrictive access to receivers is needed to prevent two emitters from shooting the same receiver.

The way tasks are distributed to processes can lead to different algorithms. We have chosen a simple approach with a single centralized tasks queue in which interactions are sorted by emitters and then by receivers. This gives us an extension of the inner loop algorithm.

4.2 Parallelization of Inter-reflections algorithm

This algorithm differs from the *direct illumination* algorithm in that it has to manage a sorted list of the emitters. That makes dynamic tasking algorithms like the single queued very hard to design. Another difference is that we have to make a "copy" of the emitting surface before processing it, either to have it treated by several processes or to allow it to receive energy from another emitter, while it sends energy itself.

The problem is that copying a huge surface may become very time consuming, in the case where this surface is already strongly subdivided; moreover the copy operation has to be done in a critical section, because it uses the Open Inventor graph. We so had to implement a "lazy copy" mechanism that allows to use a "light" copy of an emitter, and rebuild its children only when needed.

After taking into account these considerations, we implemented the two loop level parallel *inter-reflections* algorithms.

Inner Loop Here also, the inner loop is the most direct way to get a parallel algorithm. Its main advantage is that the sorted list can be dealt with as in the sequential version. The drawbacks are the same as in the *direct illumination* algorithm.

Outer Loop The outer loop is in fact the implementation of the coarser granularity consideration. It seemed to be interesting to consider it, because we have here a larger number of tasks (emitter-all receivers interactions) in comparison with the number of processes.

The convergence speed of the algorithm heavily depends on the order by which emitters propagate their energy. This can be problematic if emitters are shot in parallel, because after N shots, much less energy will have been exchanged than with N shots done sequentially.

If parallel execution of the inner loop proves to be well enough load-balanced, on the average, the first algorithm is much more efficient (no locking on the receiver is required), despite the synchronization barriers, because of its faster convergence speed.

4.3 Technical Constraints

The Candela libraries are written in C++ and consist of approximatively 358 classes. Even if it can seem challenging to parallelize C++ algorithms inside such a large platform, it is really important not to move apart from the sequential part in order to be able to make comparisons and to take advantage of its last enhancements.

Furthermore, Candela is built over the Silicon Graphics Open Inventor class library and intensively uses the graph structure and its associated nodes. Hence we have absolutely no control over the storage and manipulation of the data structures, and even less in parallel, because the Open Inventor library is not provided thread-safe: that is to say it can be very problematic if two or more processes manipulate the graph at the same time.

This has led to several implementation problems, for instance:

- we had to overload the memory allocation C functions ("malloc", "free", "realloc", "calloc") to avoid thread-safe memory manipulation locks: keep in mind that the wavelet radiosity algorithm implies many memory allocations in its parallel part and that Open Inventor provides no control over the storage of its nodes;
- another problem was to determine memory variables which could potentially be modified by several process at the same time (typically static class variables) and to build a set of preprocessor macros which allow to transparently transform them into an array of variables (one per process);

this gave us a general purpose C/C++ framework for "assisted" parallelization which could be reused for other projects.

4.4 Protocol Considerations

Test Scenes We performed our experiments on three test scenes coming from real world applications, but with different characteristics, in order to analyze our implemented algorithms:

- *Stanislas Square Opera* in Nancy. This test scene comes from an evaluation project of potential new lighting design. The geometrical model was created from architectural drawings. The direct illumination is computed using accurate light and reflectance models. In this paper, we only consider the first floor in order to be able to compute a solution on a single processor in a reasonable time (less than one day).
- *Cloister* in Quito. It is also a lighting design project, but it was chosen because the effects of indirect illumination (inter-reflections) are more visible. It serves as a life-size test.
- *Soda Hall*. The Soda Hall building has become a reference test scene. It is suitable for virtual reality environments and interactive walk-through. We both consider a single room of this building (with high precision parameters) for speed-up measures and one complete floor with furnitures as another life-size test.

Table 1 gives their numerical characteristics and associated images can be found in Figure 2 of the color plate.

Test scene	Stanislas Square	Quito Cloister	Soda Hall Room	Soda Hall Floor
Number of initial surfaces	17 307	54 789	9 189	144 255
Number of light sources	36	83	3	163
Number of final meshes	217 307	597 135	232 349	1 721 354
Time [Processors]	11 495 s [1]	35 883 s [4]	11 001 s [1]	–
Time [Processors]	593 s [36]	9 663 s [24]	613 s [36]	54 627 s [16]

Table 1. The test scenes

Measures The experiments were ran on a ccNUMA Silicon Graphics Origin2000 [6] with 64 processors organized in 32 nodes. Each node consists of two $R10000$ processors with 32 KBytes of first level cache (L1) of data on the chip, 4 MBytes of external second level cache (L2) of data and instructions and 256 MBytes of local memory, for a total of 8 GBytes of physical memory. Its hardware performance counters combined with the software tool *perfex* allow performance measures of the behavior of the parallel program. We have chosen to study:

- *Speed-up*. This is defined by the fraction between the best sequential time over the parallel time obtained with n processors.
- *Memory overhead*. This is the fraction of time spent in memory over the total execution time.
- *L1 cache hit rate*. This is the fraction of data accesses which are satisfied from a cache line already resident in the primary data cache.
- *L2 cache hit rate*. This is the fraction of data accesses which are satisfied from a cache line already resident in the secondary data cache.

5 Results and Discussion

5.1 Performance Evaluation

Table 1 gives the surfaces complexity of each scene and the computational time needed to simulate each scene first with a small number of processors and then with a large number of processors. Each result shows good speed-up. More precise results appear in Figure 1 for the Stanislas Square Opera and the room of the Soda Hall and are examined in the next subsection.

Let us recall for correctness of these figures that we work with polygons which can be concave or convex for greater efficiency. The number of surfaces which will appear if we decided to triangulate them to work only with triangles and/or parallelograms would be much greater. For instance, for the Stanislas Square, a straightforward triangulation gives two times more surfaces, and a triangulation which provides good shaped triangles gives ten times more original surfaces.

Moreover we want to remind that due to the absence of graphics card on the Origin2000, we have parallelized a version of the algorithm which only uses the "BSP" decomposition in order to accelerate the visibility and does not use any *hardware graphic* acceleration. When using the sequential version [3], this choice decreases the overall performance of the general simulation by a factor of 2. This degradation is important enough to relativize the parallel speed-up.

However, even with such accelerating techniques, huge scenes like the Cloister of Quito (see Figure 2.d) can hardly be simulated on a sequential computer. In parallelism, our test shows that this scene can be simulated with 24 processors in 2 hours 41 minutes. These sort of results are very promising for the lighting design and other real-world applications.

5.2 Discussion

Load balancing and data locality are two key but conflicting goals to reach in order to obtain good parallel performance. We show here their impact on our different algorithms.

Load balancing Figure 1.a shows that all the methods achieve a speed-up in CPU times when used in parallel. The speed-up is very high (actually supra-linear) when we use no visibility for the direct illumination. In the other cases, it is approximately equal to $0.5n$, where n is the number of processes.

In the case of a direct illumination without visibility, a supra-linear speed-up is not so surprising. Indeed, in this case, when we want to illuminate a surface we need only to use the data of that surface, this leads to excellent data locality and very similar computation times for the illumination of each surface. In this case, the Origin2000 is very efficient.

If we look at the results for the two different methods used for the Place Stanislas illumination, problems begin to appear. Of course, the algorithm that does not stop after each light gives better speed-up than the other one as expected, but the speed-up results are not very good. In fact, what seems to happen is the following:

- in this simulation, each light illuminates only a part of the building and only a few interactions are very time consuming. Therefore, when some process finishes the illumination of the last surface seen by the first light, some processes are still stacked processing one of the few interactions which are very time consuming. The synchronization of the processors, after a light source has handled all the surfaces it sees, is therefore very costly in the "inner loop" method;
- but the situation is even worse because in this simulation two neighboring lights often share a group of very time consuming interactions. Indeed the lighting designers have positioned the lights very near the facade, so that most of the energy received by a single surface came with high probability from those two light sources.
  This has a large impact on the "single queued" method. When some processors begin to compute the interaction between the second light and the building, three or four time consuming interactions between the first light and the sets of surfaces are still processed. The first processors which will attempt to shoot on one of these surfaces (the surfaces which have not yet been totally illuminated by the previous illumination) will be locked. This is problematic because these interactions are with high probability also very time consuming interactions for the second light.

We think that this situation appears because we have used a coarse [2] granularity for this simulation. This is slightly annoying because the same problem appears in the general shoot phase when the residual energy to shoot becomes too small.

Decreasing the granularity of the algorithm to tackle this problem is feasible for the direct illumination but becomes much trickier for interactions between two surfaces. Another solution seems more reasonable: when a processor attempts to shoot on a surface which is actually the receiver handled by another processor, a locking situation appears. We can remove this lock, if the processor starts to shoot on a lure: a disjoint copy of this surface; of course we will need after to retrieve the energy received by the lure and to re-project it on the real surface.

As in the simulation of the Soda Hall room, Figure 1.b exhibits the same behavior we just noted for the Place Stanislas illumination in the shooting phase. The same problem appears during the first shoots: the shoot of an emitter on the walls or on the floor takes much more time than the other computations. Also, after some shoots, the amount of energy left to emit becomes very small. In this case, the time needed for distributing the work among processes is larger than the time needed to do the shoot. The granularity of the parallelization must then be increased.

Data locality When we designed our parallel algorithms, we did not focus on data locality problems. First, we had expected that:

- the used granularity would naturally increase the data locality. Indeed when one processor deals with an emitter-receiver interaction, it computes this interaction at each needed level. Moreover, in order to compute the form factor of the kernel, it needs to

[2] A trick to artificially reduce the granularity problem would be to add a preprocessing step that would subdivide large surfaces into smaller ones.

compute the visibility between lists of points of the two surfaces. We expected this would lead to the same portions of the BSP structure being traversed;
- the type of architecture used by the Origin2000, a cc-NUMA, is well known to decrease the importance of this problem compared to explicit message passing.

Our tests on a room of the Soda Hall and on the Stanislas Square place show that:

- the ratio of time spent accessing the memory decreases slightly with the number of processors used (see Figures 1.c and 1.d);
- the hit rates are approximatively of 97% for the $L1$ cache and of 93% for the $L2$ cache (see Figures 1.e and 1.f).

The first behavior of the algorithm can be explained by the fact that the simulations of these two scenes use a lot of memory. It is not so surprising, in this case, that it can be better to distribute the data memory to all processors, which will only access a part of the total memory.

The cache hit rates have surprisingly high values, which seems in total contradiction with results presented by [8]. This seems to prove that our hypothesis is valid and that data locality is not the main problem with wavelet radiosity.

However, we want to moderate this conclusion since two other factors may have influenced this behavior:

- first, in order to do meaningful tests, we have limited ourselves to simulations which could be completed on a single processor in less than one day. Thus, though some of our scenes are somewhat big, their complexity is not tremendous;
- second, we use scenes coming from real-world applications. This means that surfaces appearing in the scene description are not listed in a random order. On the contrary, surfaces that are neighbors (as geometrical entities) often appear close to one another in the scene description. This locality of the scene description clearly impacts on the algorithm data locality.

The two preceding paragraphs show that the progressive wavelet radiosity algorithm can "efficiently" be parallelized even if reaching a good load balancing between process remains difficult. Some problems are annoying for small scenes; fortunately when the scene and the amount of work to be done become larger, the importance of these problems seems to decrease. This remark is important because these scenes can not reasonably be simulated on a single sequential computer: in those cases, parallel computing appears to be a worthwhile solution.

6 Conclusion

We have experimented in this paper various parallel implementations of the wavelet radiosity method on a hardware ccNUMA architecture, the SGI Origin2000. Experiments showed that straightforward parallelization provides good speed-up - even if this application is highly irregular and dynamic - and reasonable means to solve extremely large models with a large number of processors.

Even if the application is written in "C++" and the granularity of the parallelization remains a little coarse in this implementation, we obtain an excellent data locality - which confirms the good properties of the ccNUMA architecture - and a reasonable load balancing. However, on some scenes, load balancing can likely be enhanced using some specific improvements (shoot on a lure if needed, ...).

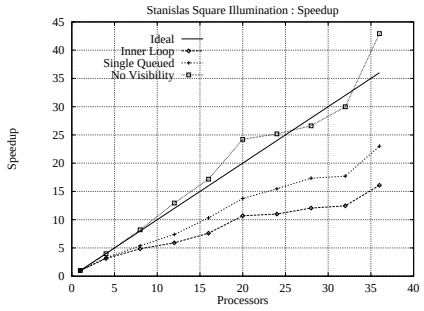
These results are interesting, first for the lighting designers who can expect to simulate some real world projects. Moreover we anticipate that these results are probably generalizable to numerical applications with linear projection operators, in the case where the space χ is some space χ_n in a multi-resolution analysis.

Acknowledgments

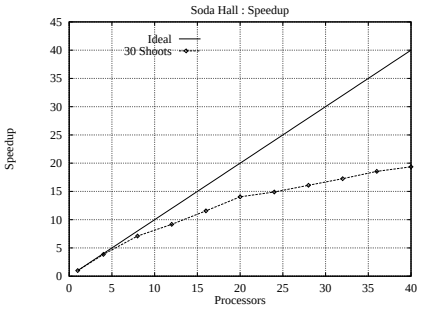
The authors would like to thank François Cuny, Slimane Merzouk and Christophe Winkler from the ISA team of the LORIA (`http://www.loria.fr`) for their work on Candela, Didier Bur and Bertrand Courtois from the school of Architecture of Nancy (`http://www.crai.archi.fr`) for their work on the models, Marc Albouy, from Electricité de France which provided the Stanislas Square and the Quito models, Carlo Sequin who provided the Soda Hall. We also thank Silicon Graphics for taking the Centre Charles Hermite (`http://cch.loria.fr`) as a beta-test site of the Origin2000, Luc Renambot for his help on performance measures, and last but not least, Alain Filbois for his great technical support on the computer.

References

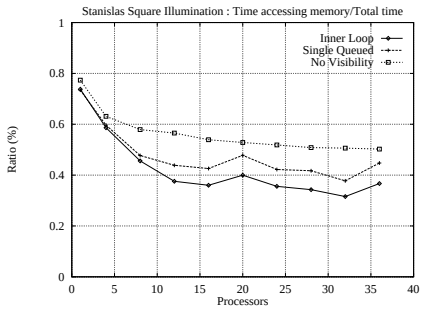
1. James Arvo. The Role of Functional Analysis in Global Illumination. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 115–126, New York, NY, 1995. Springer-Verlag.
2. James Arvo, Kenneth Torrance, and Brian Smits. A Framework for the Analysis of Error in Global Illumination Algorithms. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 75–84, 1994.
3. François Cuny, Christophe Winkler, and Laurent Alonso. Wavelet Algorithms for Complex Models. Technical Report, LORIA, INRIA Lorraine, 1998.
4. 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.
5. Steven J. Gortler, Peter Schroder, Michael F. Cohen, and Pat Hanrahan. Wavelet Radiosity. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pages 221–230, 1993.
6. James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241–251, Denver, June 1997. ACM Press.
7. Slimane Merzouk. *Architecture logicielle et algorithmes pour la résolution de l'équation de radiance*. PhD thesis, Institut National Polytechnique de Lorraine, 1997.
8. 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)*, Phoenix, AZ, October 1997. IEEE Computer Society. To be published.
9. Peter Schroder, Steven J. Gortler, Michael F. Cohen, and Pat Hanrahan. Wavelet Projections for Radiosity. *Computer Graphics Forum*, 13(2):141–151, June 1994.
10. Jaswinder P. Singh, Annop Gupta, and Marc Levoy. Parallel Visualization Algorithms: Performance and Architectural Implications. *IEEE Computer*, 27(7):45–55, July 1994.
11. Jaswinder Pal Singh, Chris Holt, Takashi Totsuka, Anoop Gupta, and John Hennessy. Load Balancing and Data Locality in Adaptive Hierarchical N-body Methods: Barnes-Hut, Fast Multipole, and Radiosity. *Journal of Parallel and Distributed Computing*, 27(2):118, June 1995.
12. David Zareski. Parallel Decomposition of View-Independent Global Illumination Algorithms. M.Sc. thesis, Ithaca, NY, 1995.



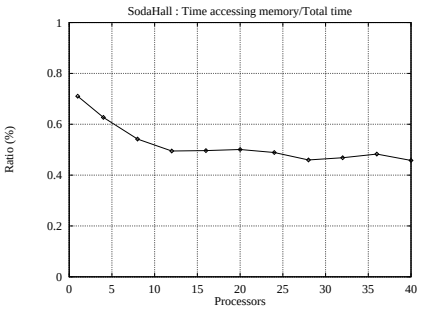
(a) Stanislas Speedup



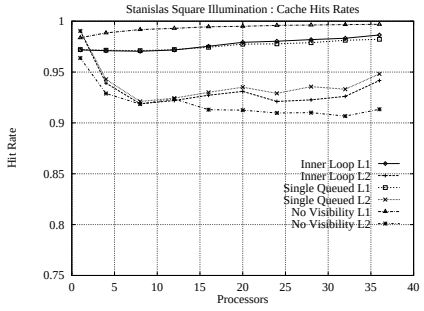
(b) Room Speedup



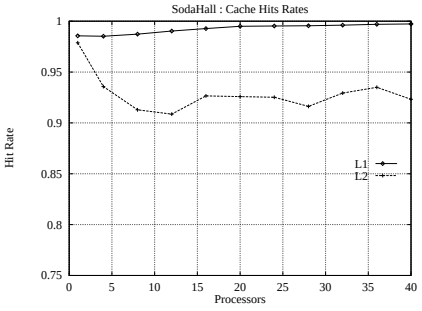
(c) Stanislas Memory



(d) Room Memory

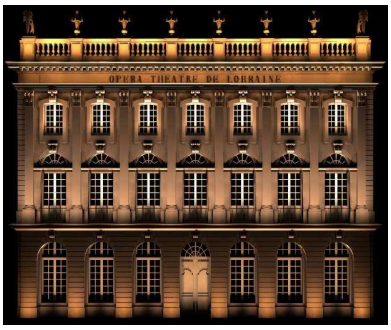


(e) Stanislas Caches



(f) Room Caches

Fig. 1. Experimentation results



(a) Stanislas Square Opera

(b) Soda Hall Room

(c) Soda Hall Floor

(d) Cloister in Quito

Fig. 2. Images generated with our parallel algorithms