Experimentation of Data Locality Performance for a Parallel Hierarchical Algorithm on the Origin2000

Xavier Cavin, Laurent Alonso and Jean-Claude Paul
{Xavier.Cavin,Laurent.Alonso,Jean-Claude.Paul}@loria.fr
LORIA - INRIA Lorraine
615 rue du Jardin Botanique, BP 101,
F-54600 Villers-lès-Nancy, France

August 20, 1998

Abstract

Hierarchical algorithms form a class of applications widely being used in high-performance scientific computing, due to their capability to solve very large physical problems. They are based on the physical property that the further two points are, the less they influence each other. However, their irregular and dynamic characteristics make them challenging to parallelize efficiently. Indeed, two conflicting objectives have to be taken into account: load balancing and data locality.

It has been shown that the message passing paradigm was not well suited for this kind of applications, because of the intensive communication they introduce. Implicit communication through a shared address space appears to be better adapted. Particularly, the ccNUMA architecture of the Origin2000 can help us getting the desired data locality through its memory hierarchy.

We have experimented a parallel implementation of a well known computer graphics hierarchical algorithm: the wavelet radiosity. This algorithm is a very efficient approach to compute global illumination in diffuse environments but still remains too much time and memory consuming when dealing with extremely complex models.

Our parallel algorithm focuses on load balancing optimization and heavily relies on the ccNUMA architecture efficiency for data locality. Load balancing is handled with a general dynamic tasking mechanism with specific improvements. Minimal efforts are made towards memory management (like the writing of thread-safe non-blocking malloc/free C functionalities) and the Origin2000 proves all its capabilities to efficiently handle the natural data locality of our application.

The implemented algorithm allows to compute the illumination of a complex scene (a cloister in Quito, composed of 54789 initial surfaces and leading to 600000 final meshes) in 2 hours 41 minutes with 24 processors. To the knowledge of the authors, this is the most complex "real world" scene ever computed.

1 Introduction

As scientific and engineering computing requires more and more computational power, parallelism appears to be one, if not the only, efficient response. Among scientific applications, a class of solutions, namely *hierarchical algorithms*, is widely being used in order to solve very large physical problems. These include fluid dynamics, chemistry, structural mechanisms, semiconductor and circuit simulation, oil reservoir simulation [4], or more recently computer graphics [8], and so is representative of a large class of problems. That is the reason why they greatly

prompted the development of supercomputers and are now likely to get their benefits.

The design of an efficient parallel hierarchical algorithm, however, is challenging, because it involves dealing with complex issues. In particular, simultaneously management of *load balancing* and *data locality* quickly becomes a headache, both because of the non-uniformity of the physical domain being simulated and because of the dynamic and unpredictable changes of workload and communication across the computation of the solution.

Chan, in [4], has pointed out that parallel (hierarchical) architectures should be designed especially to support the hierarchical communication and synchronization needs of these algorithms. Singh *et al.* focused in [12] on the implications of the hierarchical N-body methods and concluded to the necessity of an efficiently supported shared address space versus the message passing paradigm for this kind of applications.

In another paper [13], they studied load balancing and data locality for the hierarchical radiosity application, on the 48 processors *Stanford DASH Multiprocessor*, and confirmed the appropriateness of cache-coherent, distributed shared memory (*i.e.* DSM) supercomputers for this kind of algorithms. However, their experiments were limited to a very small model (94 input polygons) and they had to "extrapolate" their conclusions. Indeed, dealing with much larger environments (typically hundreds of thousands of polygons) changes the amount of communication, and so the dimension of the problem. That is why it seems interesting to us to run these experiments, with a more recent algorithm (the wavelet radiosity), on "real world" scenes and on a commercial shared-memory supercomputer, the SGI Origin2000.

Section 2 of this paper discusses hierarchical methods in general, detailing hierarchical N-body methods and hierarchical radiosity, and the issues involved in their parallelization. Then, Section 3 presents our parallel algorithm and some interesting implementation details. Experiments are described in Section 4 and results are commented in Section 5. Finally, Section 6 concludes and presents future works.

2 Hierarchical Methods

Many physical processes, described by the mathematical models of the physical laws, are hierarchical by nature. That is, they exhibit a large range of scales, both in space and time. A given point in the physical domain is progressively less influenced less frequently by parts of the domain that are further away from it. Hierarchical algorithms efficiently use the range of length scale to capture the global features of the solution. Here are some examples of such algorithms: multi-grid methods, domain decomposition methods, adaptative mesh refinement algorithms, wavelet basis, N-body methods, hierarchical radiosity. All these algorithms have common features: they result in near optimal sequential complexities and possess high potential of parallelism.

The remaining of this Section focuses on classical hierarchical N-body methods and on hierarchical radiosity, and presents the issues of their effective parallelization.

2.1 Hierarchical N-body Methods

The N-body problem studies the evolution of a system of n discrete particles (or bodies) under the influence exerted on each particle by the whole ensemble. The naive algorithm which computes all interactions between each pair of particles suffers from a $O(n^2)$ complexity. Fortunately, this complexity can be reduced to $O(n\ log\ n)$ by hierarchical algorithms [1, 2].

Hierarchical N-body methods are based upon a physical property that dates back to Isaac Newton (1687): "If the magnitude of interaction between particles falls off rapidly with distance (as it does in most physical interactions, such as gravitation or electrostatics with their $1/r^2$ force laws), then the effect of a large group of particles may be approximated by a single equivalent particle, if the group is far enough away from the point at which the effect is being evaluated.", as shown by Figure 1 [12].

Actually, hierarchical N-body methods can be applied to many physical domains, including astrophysics, plasma physics, molecular and fluid dynamics or computer graphics. They so represent a large class of supercomputers potential users.

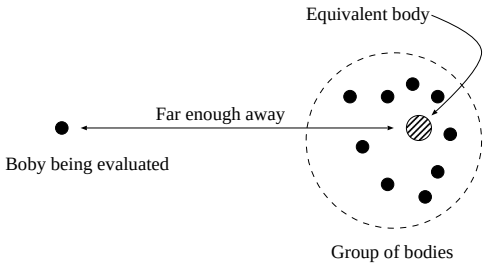

Figure 1: Principle of N-body methods.

2.2 Hierarchical Radiosity

Radiosity methods are an efficient approach to compute global illumination in Lambertian (*i.e.* diffuse) environments. They are based on the physics of light transport and capture both direct illumination (by light sources) and indirect illumination (through multiple inter-reflections), resulting in a view-independent solution.

The radiosity - power per unit area $[W/r^2]$ - on a given surface is defined as the light energy leaving the surface per unit area. Let $\mathcal{M}$ denote the collection of all surfaces in an environment. Let χ be a space of real-valued functions defined on $\mathcal{M}$, that is, over all surface points. 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[1]:

$$f(x) = g(x) + k(x) \int_{\mathcal{M}} G(x', x) f(x')\, dx' \qquad (1)$$

where $k(x)$ is the local reflectance function of the surface (*i.e.*, we suppose that the surfaces are ideally diffuse), and $G(x', x)$ is a geometry term defined by:

$$G(x', x) = \frac{\cos\theta_{x'} \cos\theta_x}{\pi\, r^2_{x'x}}\, v(x', x) \qquad (2)$$

and consisting of the cosines made by the local surface normals with the vector connecting the two surface points x and x', of the distance r between theses two points (see Figure 2), and of the visibility function v whose value is in $\{0, 1\}$ according to whether the line between the two points x and x' is respectively obstructed or un-obstructed.

[1] In the case of monochromatic radiosity, that is to say for a given wavelength.

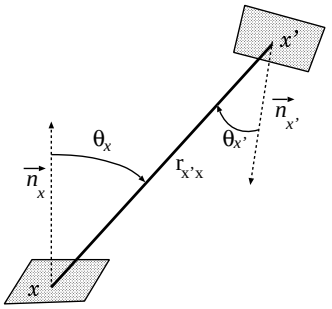

Figure 2: Components of geometry term G.

In the original radiosity method, the input surfaces ($\mathcal{M}$) are subdivided into small elements over which the radiosity is approximated as a constant function. This results in a set of $O(n^2)$ interactions to be computed, where n is the number of final elements.

Fortunately, the geometry term $G(x', x)$ involved in the integral equation (1) decays with the inverse square of the distance $r_{x'x}$ between elements x and x'. As in the hierarchical N-body methods, this property allows to build a $O(n\ log\ n)$ hierarchical algorithm, which was done by Hanrahan *et al.* in [8].

It is important, however, to note that hierarchical radiosity algorithms differ from hierarchical N-body methods in many points. Indeed, the physical law used to compute the interactions between surfaces not only involves the interacting entities x and x' and the distance $r_{x'x}$ between them, but also a cosines product $\cos\theta_{x'} \cos\theta_x$ that takes into account their respective orientations and a visibility function $v(x', x)$ that detects possible occluding surfaces. The orientation consideration prevents the "clustering" of surfaces without loss of informations; on the contrary, the starting point of the algorithm is the input surfaces of $\mathcal{M}$, which are hierarchically subdivided as the algorithm proceeds, as shown by Figure 3. The visibility computation is a global term that may involve other surfaces, from the entire scene, than the two interacting ones; however, it can be accelerated using a binary space partition (*i.e.* BSP) of the scene. More-

over, the algorithm in itself does not proceed over hundreds of time-steps, but either by a few *gathering* passes (each surface gathers energy from all others) or many *shooting* iterations (the most energetic surface propagates its energy to all others).

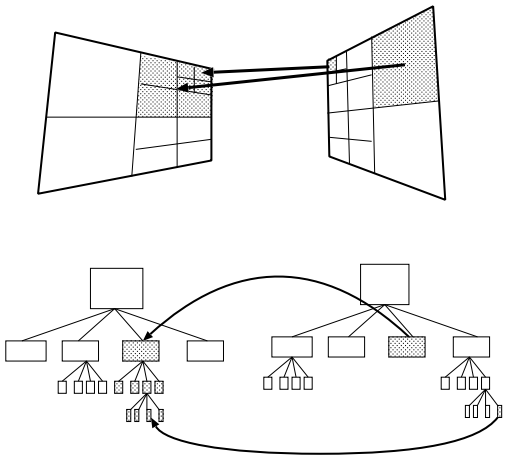

Figure 3: Principle of hierarchical radiosity.

The wavelet radiosity algorithm, introduced by [7] and [11] is a generalization of the hierarchical radiosity: the radiosity function is no longer approximated as being uniform on a surface element but is now projected on wavelet bases, thus allowing a better subdivision and error control.

2.3 Parallel Issues

As pointed out in [4], hierarchical algorithms possess relatively high degrees of parallelism. If the parallelization of hierarchical N-body methods has been well studied [12, 13], the design problem of parallel hierarchical radiosity has not yet been completely addressed [3, 6, 13, 14]. However, all works seem to prove that the message-passing paradigm is not well suited for the dynamic, irregular and unpredictable behavior of hierarchical algorithms. On the contrary, cache-coherent DSM supercomputers seem to be able to efficiently support the hierarchical constraints through their hierarchy of memories.

Unfortunately, choosing a shared memory supercomputer (randomly, the SGI Origin2000) is not sufficient to get an efficient parallel application! Indeed, as in many scientific algorithms, designers have to deal with two key but conflicting problems: load balancing and data locality.

Related to these problems is the choice of the granularity used for the decomposition of the algorithm into elementary tasks. For the hierarchical radiosity method, the granularity can go from very coarse (a task is the set of interactions between a surface and all other surfaces) to very fine (surface-surface element interaction or even surface element-surface element interaction) through intermediate (surface-surface interaction). Obviously, the smaller the granularity, the easier the load balancing but the worse the data locality (and *vice versa*).

3 Parallel Wavelet Radiosity

We focus in this Section on our work for the parallelization of a wavelet radiosity algorithm. We start with the description of the sequential version we use, then we present our parallel algorithm. Finally, we explain some interesting implementation details.

3.1 Sequential Algorithm

Our work takes place inside a large project, named *Candela*, which has been designed to provide a flexible architecture for testing and implementing new radiosity and radiance algorithms [10]. It was at the same time intended to be able to deal with real input data and to compute physically correct results.

Several sequential wavelet algorithms have already been implemented and experimented in [5]. Our sequential experiments have shown that the wavelet coding of the radiosity function is very important in order to obtain correct results. Moreover, one particular instance of our wavelet algorithms came out as the most effective one to deal with large architectural scenes : the *progressive shooting wavelet radiosity without link storage*.

This 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). Each energy propagation (*i.e.*

interaction) is done "hierarchically": an *oracle* function decides (on energical or geometrical considerations), which of the two surfaces (maybe none of them) should be subdivided to enhance the precision of the solution; then the energy propagation is recursively done between the non-subdivided surface and each of the children[2] of the subdivided one (see Figure 3). In traditional hierarchical radiosity methods, links are created between surface elements not involving subdivision; because of the huge amount of memory this requires, our algorithm does not store the links.

3.2 Parallel Algorithm

Despite all the qualities of our sequential algorithm, time and memory requirements still remain too high for effective use on a single workstation, especially when the scenes to simulate are very large (the common case for architectural simulations). Thus, parallelism appears to be the alternate way to bypass the single workstation limitations.

A complete study of our choices and algorithms can be found in [3]. We just give here a short description of the important points.

Basically, we have chosen to decompose the sequential algorithm with the intermediate granularity (*cf.* Paragraph 2.3), that is to say, we have defined a task as a standard surface-surface interaction. We hoped this would be fine enough to enable an efficient load balancing (at least in the case of large scenes), but not too much to get some data locality (keep in mind that we heavily rely on the Origin2000 architecture to help us on this quest).

The parallel algorithm thus consists in distributing these tasks to processors, in the best load balanced way. Our experiments showed that a simple dynamic tasking algorithm with a single centralized tasks queue can give good results, at least at moderate processors scale (up to 40 processors). When scaling to a much larger number of processors, a distributed tasks queues algorithm, associated with a *task stealing* mechanism, may become necessary to avoid the single queue bottleneck.

[2]Each child is referred as a *surface element.*

3.3 Implementation

The *Candela* libraries consist of approximatively 360 *C++* 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 SGI *Open Inventor* library and intensively uses the scene graph structure and its associated nodes. Hence, we absolutely have no control over the storage and manipulation of the data structures. It is even worse in parallel, because of the non *thread-safe* behavior of the *Open Inventor* library: a catastrophic error (*i.e.* a *core dump*) may occur if several processes manipulate the scene graph at the same time.

Finally, the many dynamic memory allocations implied by the hierarchical wavelet algorithm have to be done in parallel in a non-blocking way, in order to avoid memory allocations congestion. Unfortunately, this is not the case with the standard *IRIX* memory allocation package, which has been made thread-safe by serializing the parallel allocations.

Consequently, we have developed an independent, general purpose *C/C++* framework, which aims to facilitate the design of parallel programs, at least on the SGI Origin2000. Here are some of the available functionalities. First, it provides encapsulated calls to specific *MP* routines, such as processes management (we use `m_fork`'ed processes) and synchronization methods (locks, barriers). This allows us to automatically monitor the synchronization times and to consider porting the application under other operating systems and/or architectures.

Then, a set of preprocessor macros allows to transparently transform a variable which could potentially be modified by several processes at the same time (typically `static` class variables) into an array of variables (one per process).

Finally, a new *C* memory allocation package (*i.e.* overloaded `malloc`, `free`, `realloc`, `calloc` functions) has been written. It allows fully parallel, contention free, memory operations **without any modifications** to the source code.

4 Protocol Considerations

This Section is intended to give a full description, as complete as possible, of our experiments protocol. We first describe the hardware/software configuration. Then we mention the measures we perform. Finally, we present our architectural test scenes.

4.1 Hardware and Software

We performed our experiments on the Silicon Graphics Origin2000 installed at the *Centre Charles Hermite*[3], in France.

The machine is equipped 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 128 MBytes of local memory, for a total of 8 GBytes of physical memory. The operating system running on it is the *IRIX 6.5SE* OS.

For sake of completeness, our application has been compiled with the *MIPSpro 7.2.1 C++* compiler, with the following options: `CC -n32 -Ofast=IP27 -mips4 -r10000`.

4.2 Measures

For our data locality performance analysis, we intensively used the $R10000$ hardware performance counters, combined with the software tool `perfex`. In particular, 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.

[3]See the web page: `http://cch.loria.fr`

The curves we present further in this paper are automatically generated, with the help of *shell scripts*, from our application and `perfex` output traces. We are currently working on a graphical interface for instrumenting, running and exploiting the parallel experiments. Such a tool would be a great help for parallel program tuning and optimization phases.

4.3 Test Scenes

Our experiments were performed on three test scenes, all coming from real world applications, but with different characteristics:

- *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.
- *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 (number of initial surfaces, number of light sources and number of final meshes) and reference computation times.

Scene	Initial	Lights	Final	Time [P]
Opera	38 258	98	513 310	40 982 s [1]
Cloister	54 789	83	597 135	9 663 s [24]
Room	9 189	3	232 349	11 001 s [1]
Floor	144 255	163	1 721 354	54 627 s [16]

Table 1: The test scenes.

5 Results and Discussion

In this Section, we analyze the performance of our parallel algorithm on the test scenes of Paragraph 4.3. Table 1 gives the complexity of each scene and the computational time needed to simulate them. We used the Opera and room models for a complete analysis and the cloister and floor models as punctual life-size tests.

We start by presenting a performance evaluation, in terms of speed-up and synchronization times. Then we focus in detail on the data locality analysis, and we complete the Section with an open question about the scalability.

5.1 Speed-up Results

Figure 4 shows the speed-up results for the Stanislas Square Opera illumination and for the Soda Hall room simulation. The speed-ups, although not linear, are quite good for both scenes.

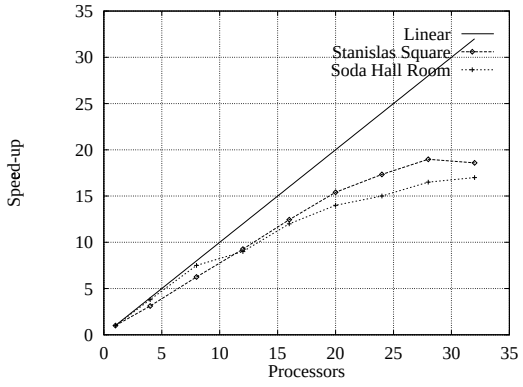

Figure 4: Speed-up measures.

Our application was instrumented to compute the time lost at synchronization points (*i.e.* locks and barriers); unfortunately, a bug in `time()` and `gettimeofday()`[4] prevents us from exploiting the results. However, when it did not occur, we could notice that synchronization times increase slowly with the number of processors and only represent, in the worst cases, about 10% of the total execution time.

[4]Making sometimes time go backward of 1 second!

The intermediate granularity we have chosen allows a fair load balancing with an algorithm of low complexity, at least in the case of large scenes and at a moderate scale. Indeed, reducing the amount of work (*i.e.* the scenes size) or increasing the number of processors might have processors either waiting on a lock, or remaining idle at the end of the computation, especially when only a few interactions, sharing common receivers, are time consuming compare to the others.

Let us now see the impact of this granularity choice on the data locality afforded by the algorithm.

5.2 Data Locality

The aim of our study was to evaluate the capabilities of the SGI Origin2000 architecture to handle the hierarchical behavior and inherent data locality of the parallel hierarchical wavelet radiosity algorithm.

Actually, the design of our parallel algorithm did not take into account the data locality problems. Indeed, we expected that the granularity we had chosen would naturally increase the data locality. First, when a given processor handles a surface-surface interaction, it computes the energy transfers at each needed level of the hierarchy, thus ensuring a good reuse of the data structures. Then the visibility computations for this interaction involve points of the two surfaces and thus lead the same parts of the BSP structure to be traversed. Finally, the ccNUMA architecture of the SGI Origin2000 [9] seems to be very promising to handle the dynamic behavior of our hierarchical algorithm.

The first point of our experiments concerns the memory overhead. Figure 5 shows the ratio of time spent in memory to total execution time for the Opera and room models. We can first notice that our application intensively uses the memory (about 70% for a single processor and about 50% with more than 8 processors). This fact is due to the many memory allocations involved by the hierarchical algorithm and confirms the necessity of an efficient non-blocking memory management.

Moreover, the communication to computation ratio decreases slightly with the number of processors. This can be explained by the fact that a large amount

of memory (more than 128 MBytes[5]) is needed for the simulation of both scenes: the memory becomes better distributed as the number of processors increase, and the latest have only access to a part of the total memory.

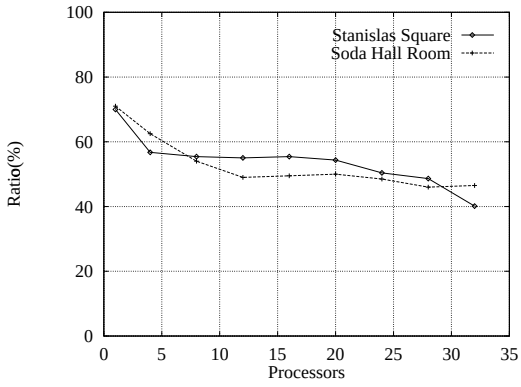

Figure 5: Time accessing memory/Total time.

At the same time, the cache miss rates are very low: less than 5% for the $L1$ cache and less than 10% for the $L2$ cache (see Figure 6).

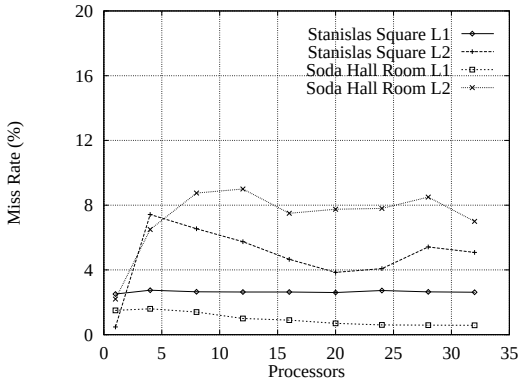

Figure 6: Cache Miss Rates.

This rates have surprisingly low values, if we consider that we do not focus our algorithm on data locality. However, this seems to prove that the working set (*i.e.* hierarchical data structures + parts of the BSP) of an elementary task has an adequate size for the memory caches of the Origin2000 and that its ccNUMA architecture is very efficient to handle the dynamic communications of the algorithm.

To be complete, our two punctual life-size tests, with the Quito cloister and the Soda Hall floor models, gave similar results. Naturally, we plan to confirm these experiments with a complete study of these two models.

5.3 Scalability Problems?

The careful reader will have noticed that the results we have shown only concern 1 to 32 processors, while our Origin2000 has 64 processors. Actually, Figure 7 shows the complete speed-up curve for the Stanislas Square Opera model (the same problem appears with the room scene): with more than 28 processors, the speed-up is limited about 20. There is clearly a scalability problem.

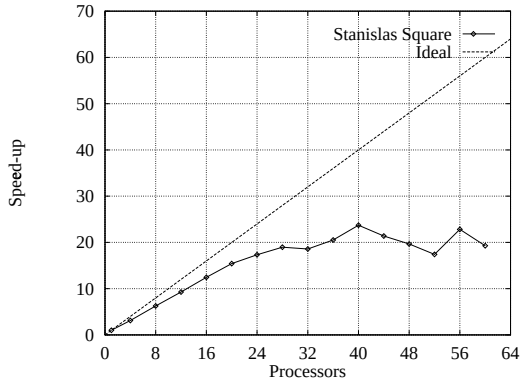

Figure 7: Speed-up for the Opera model.

What seems very surprising to us is that all `perfex` informations (TLB, $L1$, $L2$, ...) and synchronization times are coherent from 1 to 60 processors.

One common cause to speed-up decrease is known under the term of *false sharing*: when a process modifies a memory page owned by one or several other processes, this page is automatically invalidated in their memory caches, thus causing a cache miss and a page transfer for the next access. The more processes there are, the more it is likely to happen. It may be more critical within our application, because

[5] The size of a processor's local memory.

we did absolutely nothing (for instance, data alignment) to prevent this phenomenom.

Fortunately, a specific counter of the $R10000$ processor (counter 31) allows to study false sharing, by counting *stores or prefetches with store hints to shared blocks in secondary memory cache*. Figure 8 shows the evolution of this counter with the number of processors for the same scene. Surprisingly, our application does not suffer from false sharing problems: another good point for the Origin2000.

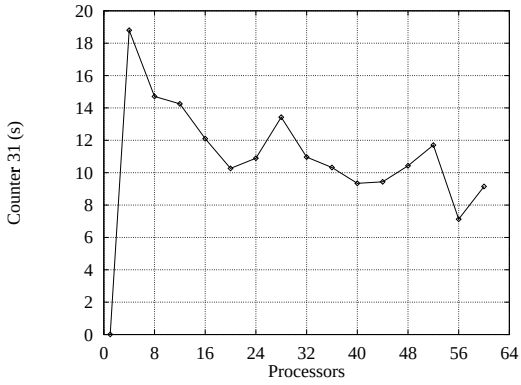

Figure 8: Evolution of $R10000$ counter 31.

However, "the truth is out there": for an unknown reason, the number of cycles increases with the number of processors, thus decreasing the speed-up, as shown by Figure 9. This curve is interesting: it presents a (non dramatic) first grow from 1 to 4, due to the parallelism overhead. Then, it remains constant (the normal case) to 20, giving a quite linear speed-up. It then begins to slowly increase from 24 to 40 before abnormally growing until 60, with a strange decrease at 56. Looking at Figure 7, we can notice that the speed-up curve exactly follows these variations.

It just remains to locate the problem, that is to say, the part of the code which takes more time as the number of processors increase. We first tried the `ssrun` tool, but the results it gave were not coherent enough to help us. We would have liked to try the `pixie` tool, but, unfortunately, it seems to crash the computer under the *IRIX 6.5SE* OS, and so we cannot use it. The scalability problem remains open.

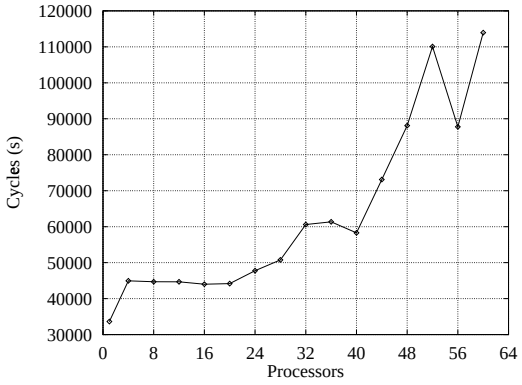

Figure 9: Scalability problem.

6 Conclusion

We intended in this paper to make a data locality performance analysis of the SGI Origin2000 for a modern computer graphics hierarchical application: the wavelet radiosity algorithm. The machine proves to be, as it was designed for, very efficient to handle the dynamic and unpredictable communication and synchronization needs of this algorithm. The memory management performances (memory caches, false sharing) of its ccNUMA architecture, from 1 to 60 processors, allow us to focus on load balancing problems, leaving the data locality to the computer.

However, the speed-up of our parallel application is currently bounded to 20, from 28 to 60 processors. Indeed, for an unexplainable reason, the number of cycles suddenly increases after 20 processors. Currently, the state of the available tools and of our knowledges do not allow us to solve this scalability problem: we are still working on it.

During this study, we have felt the need to work on the development environment. Indeed, the SGI Origin2000 provides some efficient, but not convivial, tools for applications development and tuning. We so have developed a framework for *C/C++* applications development, synchronization times monitoring and `perfex` outputs automatic graphical analysis. We plan to apply it to a completely different application to be sure of its reusability.

Acknowledgments

The authors would like to thank François Cuny, Slimane Merzouk and Christophe Winkler for their work on the *Candela* platform, 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 the Centre Charles Hermite, which owns the Origin2000, and Alain Filbois for his great technical support on the computer.

References

[1] Andrew W. Appel. An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6(1):85–103, January 1985.

[2] Josh Barnes and Piet Hut. A hierarchical *O(N* log*N)* force-calculation algorithm. *Nature*, 324(4):446–449, December 1986.

[3] Xavier Cavin, Laurent Alonso, and Jean-Claude Paul. Parallel Wavelet Radiosity. In *Proceedings of the Second Eurographics Workshop on Parallel Graphics and Visualisation*, Rennes, France, September 1998. Eurographics. To appear.

[4] Tony F. Chan. Hierarchical Algorithms and Architectures for Parallel Scientific Computing. In *Proceedings 1990 International Conference on Supercomputing, ACM SIGARCH Computer Architecture News*, volume 18, pages 318–329, Amsterdam, Netherlands, September 1990.

[5] François Cuny, Christophe Winkler, and Laurent Alonso. Wavelet Algorithms for Complex Models. Technical Report, LORIA, INRIA Lorraine, 1998.

[6] 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.

[7] 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.

[8] Pat Hanrahan, David Salzman, and Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991.

[9] 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.

[10] Slimane Merzouk. *Architecture logicielle et algorithmes pour la résolution de l'équation de radiance.* PhD thesis, Institut National Polytechnique de Lorraine, 1997.

[11] Peter Schroder, Steven J. Gortler, Michael F. Cohen, and Pat Hanrahan. Wavelet Projections for Radiosity. In *Fourth Eurographics Workshop on Rendering*, number Series EG 93 RW, pages 105–114, Paris, France, June 1993.

[12] Jaswinder Pal Singh, John L. Hennessy, and Annoop Gupta. Implications of Hierarchical *N*-body Methods for Multiprocessor Architectures. *ACM Transactions on Computer Systems*, 13(2):141–202, May 1995.

[13] 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.

[14] David Zareski. Parallel Decomposition of View-Independent Global Illumination Algorithms. M.Sc. thesis, Ithaca, NY, 1995.