A mixed dataflow algorithm for ray tracing on the CRAY T3E

Jean-Christophe Nebel
E-mail : nebel@limeil.cea.fr

Commissariat à l'énergie atomique
Centre de Limeil-Valenton
DRIF/DCSA/MLS/GDD
94195 Villeneuve-St-Georges Cedex
FRANCE

Ecole Nationale Supérieure des Mines de Saint-Etienne
Centre SIMADE
158, cours Fauriel
42023 Saint-Etienne Cedex 2
FRANCE

Abstract

The ray tracing scheme is one of the most complete and efficient rendering methods. A major drawback of this model is its high computational cost which limits its practical use. Moreover, the quest for realistic rendering requires larger and larger databases to describe scenes. With the development of distributed memory parallel computers such as the CRAY T3E, the most promising way to improve ray traced pictures productions seems to be parallelization which offers both increased CPU power and memory facilities.

In the ray tracing algorithm, each pixel of the screen is processed independently. A natural way of parallelization is to distribute pixels over the machine nodes. However, since we want to deal with large scenes, objects also have to be distributed among processors, so a modified parallel algorithm is necessary.

Strategies based on object dataflow have been proposed, but their communication load is too high. More efficient algorithms have to reduce the number of messages. Therefore we propose a mixed dataflow approach : each message will contain several pieces of information on both objects and rays. By this way, we hope to limit the communication load and to ensure a dynamic load balancing.

A parallel ray tracing algorithm on the T3E using our mixed dataflow approach is implemented. The results are very encouraging since the computation time and the communication flows may be reduced by significant factors. Scalability is globally improved, mainly because saturation occurs for larger problem sizes.

Key-words

Computer graphics, rendering, ray tracing, parallelism, DMPC, MPP, load balancing.

1. Introduction

The ray tracing scheme, introduced by Whitted [16], is one of the most complete and efficient rendering methods. The basic idea is simple: when an observer sees a point on a surface, he actually looks at the result of interactions between this point's rays and other rays coming from the scene. These interacting rays may come from light sources, reflections from other surfaces or refractions through transparent objects.

A major drawback of this model is its high computational cost which limits its practical use, despite numerous acceleration schemes (see survey [1]). Moreover, the quest for realistic rendering requires larger and larger databases to describe scenes. With the development of distributed memory parallel computers (DMPC), the most promising way to improve ray traced pictures productions seems to be parallelization which offers both increased CPU power and memory facilities.

We start this paper by studying the different kinds of parallelization used for ray tracing. Next we describe a new scheme of dynamic load balancing for the object dataflow algorithm and show some experimental results. Then, we propose an original approach of parallelization, concurrently using object and ray dataflow. And, finally, we present results and discussions.

2. Ray tracing on DMPC

In the ray tracing algorithm, all the pixels of the screen are processed independently. A natural way of parallelization is to distribute pixels over the machine nodes. If the entire scene can be duplicated in the memory of each processor (since there is no global memory available), a scheme without dataflow is used; otherwise, objects composing the scene have to be distributed over nodes. Then two strategies are possible to perform the computation : object dataflow or ray dataflow.

2.1. Duplication of the scene

For this kind of algorithm, each processor independently computes its subset of the pixels thanks to the duplication of objects in its local memory. A strategy for picture division has to be defined for a good load balancing,. First, static distributions (compared in [11]) have been proposed, but dynamic schemes are more efficient. The most classical one is the master-slaves model [13], [15], where slaves ask the master pixels to compute. However, it is not very scalable. Others strategies have been proposed : clusters with their own master [5] or rings of workers exchanging pixels with their neighbours [2].

This approach may reach a very high level of efficiency, but it is limited by the size of the local memory.

2.2. Object dataflow

In this approach, all the objects of the database are distributed on nodes and each processor has to compute only the pixels which are in its own memory. When a ray has to intersect an object missing from the local memory, a request to the processor owning this object is executed. Then, after receiving this request, a copy of the object is sent to the caller. Finally, this data is recorded in its cache memory. This object search may be performed by using master-slaves models [5], tree based architectures [6] or shared virtual memories [4].

Static partitions of pixels have to be associated with schemes of dynamic load balancing. Most of them are based on pixel exchanges, so that strategies explained in the previous section (δ2.1) can be used.

The drawback of this algorithm is that its efficiency is strongly linked to the cache memory size.

2.3. Ray dataflow

In this parallel scheme, the scene is divided into subsets so that each processor deals with a particular part of the 3-D space. When a ray leaves a region to enter another one, its computation is continued on by the processor which is responsible for the new area.

If a ray-object intersection cannot be performed, the solution is to send the ray to the processor that owns the object instead of requesting a copy of the object, as in the object dataflow method. Here, the critical phase is the object partition because it induces the processors load. Numerous scene cutting methods have been studied in [9], for the uniform spatial subdivisions, and in [3], [8], for those based on pre-sampling.

These pre-processing policies have to be combined with dynamic load balancing strategies. They are mainly based on the competition between two processes on each processor [10], [14]: one of the processes computes ray-object intersections, while the other executes tasks which are not object-dependent. So overloaded processors simply execute the first process, whereas unloaded ones execute the second process.

This parallelization requires a sharp load balancing, because unbalance is very scene dependent.

2.4. Conclusion

Algorithms without dataflow are the fastest. However, if the scene has to be partitioned because of its size, which one of the other two strategies is the best? This comparison is not easy because scientists use different computers, communication networks and scenes or their algorithms are not based on the same sequential algorithm.

Now that we have seen different schemes of parallelization, we are going to compare the two algorithms with dataflow and then we propose another strategy.

3. Object dataflow algorithm with dynamic load balancing

First we compare parallel schemes under our needs and constraints. That allows us to choose the basic algorithm for ours researches : an object dataflow scheme with asynchronous communications. Next we present a new algorithm for dynamic load balancing. Finally we show our experimental results.

3.1. Choice of an algorithm

At first, we wish to deal with scenes of large sizes. Consequently the scene has to be distributed. That induces a choice between object and ray dataflow. Secondly our algorithm shall be used on various parallel architectures (networks of workstations, CRAY T3D, CRAY T3E...). Therefore we cannot profit by the topology of the network, and communications have to be insured by higher level standard libraries (PVM or MPI). Moreover, CRAY T3D and T3E prohibit the using of several processes on a processor and need a good behaviour with hundreds of processors.

Efficient load balancing for ray dataflow, requiring the use of concurrent processes on each processor, cannot be implemented. Then we have to turn to object dataflow algorithm. However we have to limit costs of communications, because message passing is very time consuming specifically on networks of workstations. That is why we decide to implement an asynchronous scheme. At last, using massively parallel computers lead us to design a decentralised dynamic load balancing.

3.2. Principles

We decide to develop an object dataflow algorithm with asynchronous communications.
In such a scheme, when a ray-object intersection cannot be processed by one processor because it does not own the object, the processor asks the object to the processor which owns it. Then it waits for the object and continues its processing only when the message containing a copy of the missing object is returned. Such kind of communication is very time consuming. That is why we propose asynchronous communications and bufferization of object requests.

In our scheme, when a ray cannot be computed, its processing is interrupted and the ray is placed in a special buffer. Then other rays are computed. The request to the processor which owns the missing object is only performed when a certain number of rays needs this same object. Communications are asynchronous, so computations may go on. Finally, when the missing object is received, rays needing this object are processed.

Thanks to bufferization and asynchronous, communication costs may be reduced by significant factors.

Load balancing is fundamental for parallel computation. Thus we propose two complementary strategies. At first load balancing is ensured by an homogenous distribution of pixels on processors. But considering we wish to work on MPP, we also have to implement a scheme of decentralised dynamic load balancing.

This algorithm needs a scheme which allows every processor to know the load of the others. Furthermore, we have to avoid the master-slaves scheme because it leads to bottlenecks on MPP. Thus, each message will include a header, which contains information about the load of the sending processor. This load is the number of rays still to be computed. After a few messages, each processor is able to know the load of every other processor in the parallel system (indeed, in our scheme, each processor communicates with all other processors).

At every instant a node has the knowledge of other processors load. Thus, as soon as one processor is going to lack work, it just has to directly ask to the processor having important load.

At last, thanks to object requests bufferization, it is possible to select rays, which are sent to the asker processor. At first we send rays that the asker can resolve with its local memory, secondly other rays needing objects, and finally the rays left.

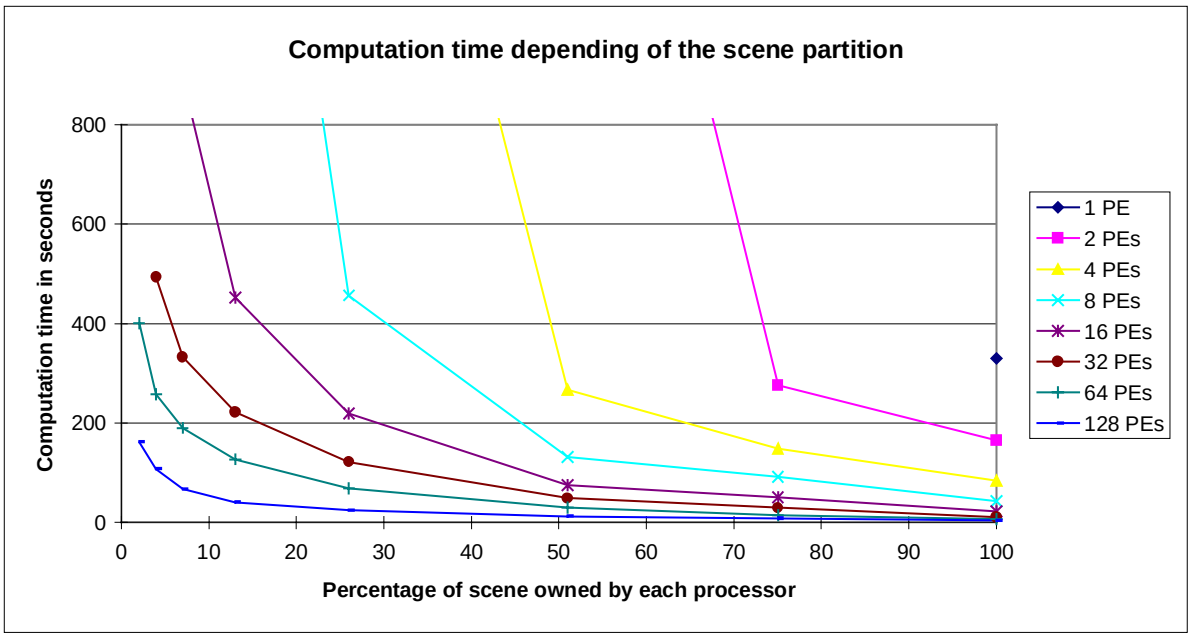
This dynamic load balancing scheme should be efficient on MPP and reduce the number of object requests.

3.3. Experimental results

Now that principles of our algorithm have been explained, we present our experimental results (entire results can be found in [12]).

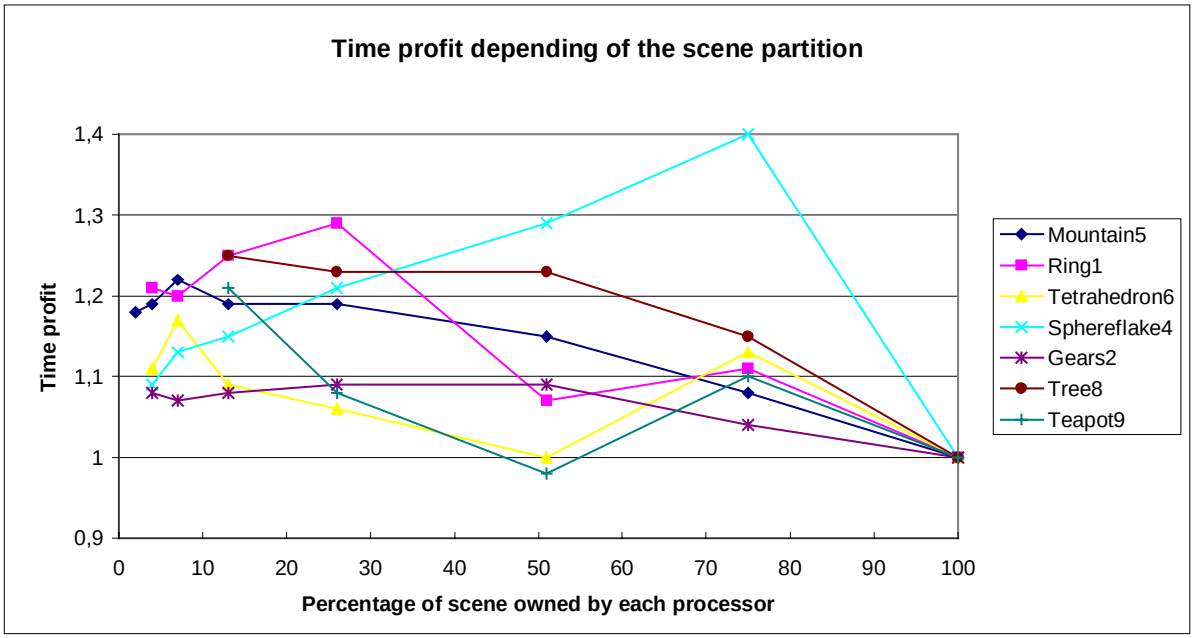
Our strategy has been implemented from the sequential ray tracer OORT [17]. The presented results have been simulated on a CRAY T3E with 128 processors interconnected by a 3-D toric grid. The message passing library used is PVM. The size of the pictures is 512x512 pixels and the maximal ray depth is 5. Scenes come from the SPD database [7] and contain between 341 and 8688 objects.

The following figure shows the computation time obtained with our object dataflow algorithm without dynamic load balancing, according to the percentage of scene owned by each processor, for various number of processors. Results are given for the database Mountain5 and are representative to those got with other pictures.



On each chart, two parts may be dissociated. When most of the database is owned by each processor, the computation cost of scene partition is low. But at a certain rate, depending of the number of processors, computation time increases very fast. In fact, under a certain partition rate, the number of communications is so high that asynchronism stops to be efficient, processors have to wait for messages in order to be able to compute rays.

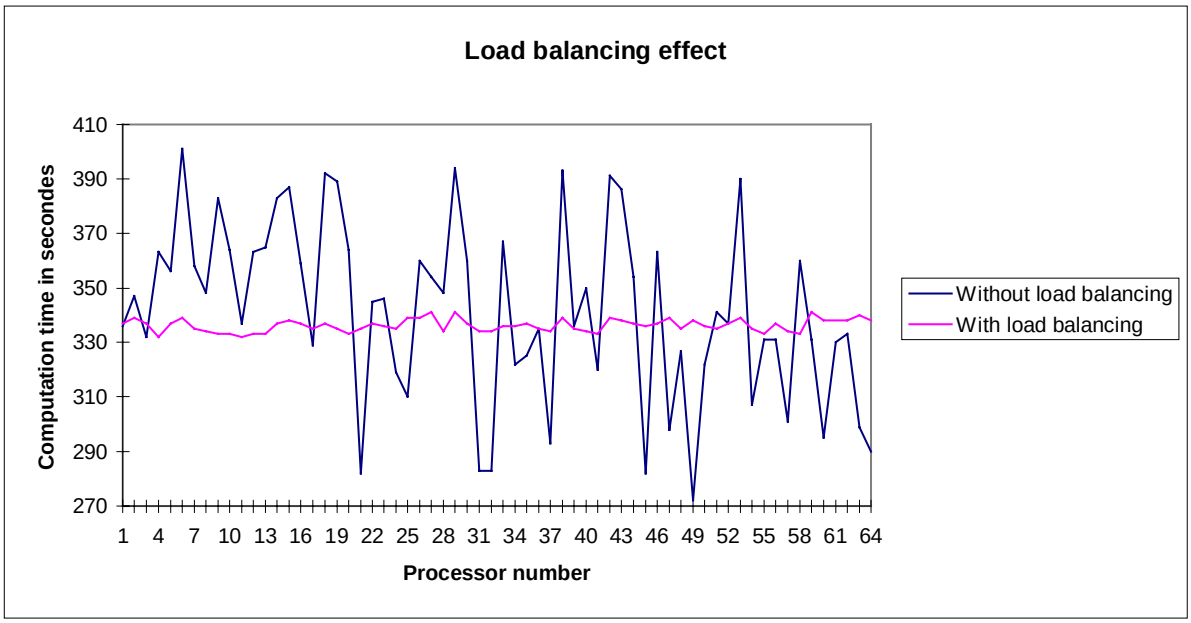
Now we study results obtained with our dynamic load balancing. In order to compare our new strategy with the classical scheme, we divide the processing time of the algorithm without dynamic load balancing by the processing time of the algorithm with dynamic load balancing. The obtained values, we call time profits, are presented on the next figure. We show results on 64 processors for various types of scenes.



Time profits of a load balancing scheme are strongly linked with the type of the tested scenes, but we can make some general comments. Profits are between 1 and 1.4 and tend to be optimal when a low percentage of the database is owned by each processor. It can be easily explained because our dynamic load balancing algorithm is based on the knowledge of the load of other processors. These load information are transmitted when communications are performed. So the more messages are sent, the more loads are known precisely.

The communication cost of this scheme is very low, because it induces only some messages asking rays and others sending rays.

On the next figure, we present the unbalancing of each processor for the algorithms with dynamic load balancing and without balancing.



On the figure, the balancing effect of our algorithm clearly appears. Our original scheme gives good results: computation time is divided by 1.2. And the expected profits for a better strategy are very low: the maximum unbalancing is 9 s (less than 3% of the computation time).

If we want to improve more efficiently the ray tracing algorithm, we have to propose a new way of parallelization instead of a new algorithm of load balancing. We present a new parallelization algorithm in the next chapter.

4. A mixed dataflow algorithm

In this chapter, we first propose an original parallelization scheme based on mixed dataflow. Then we present our experimental results.

4.1. Principles

A classical dynamic load balancing is only used at the end of computations when processors need new rays to compute. This kind of strategy does not changed the way of which the object dataflow algorithm is executed. Only the idle time of processors is reduced. So a classical dynamic load balancing can only provide a low improvement.

It appears that the cost of numerous communications, in spite of the asynchronism, is still important for the object dataflow algorithm. So more efficient algorithms have to reduce the number of messages by a significant factor. A way to perform this task is to build messages containing several pieces of information. Therefore from our object dataflow algorithm, we propose to benefit of each object sending by adding rays in the same message.

When a processor (A) has to send an object to an other processor (B), a mixed message may be composed. This message will contain the object and rays needing objects owned by the processor (B). So, future object requests, which should be generated by these rays, are avoided.

This ray insertion is done only if it does not risk to cause a load unbalancing and if the number of rays to send is sufficient.

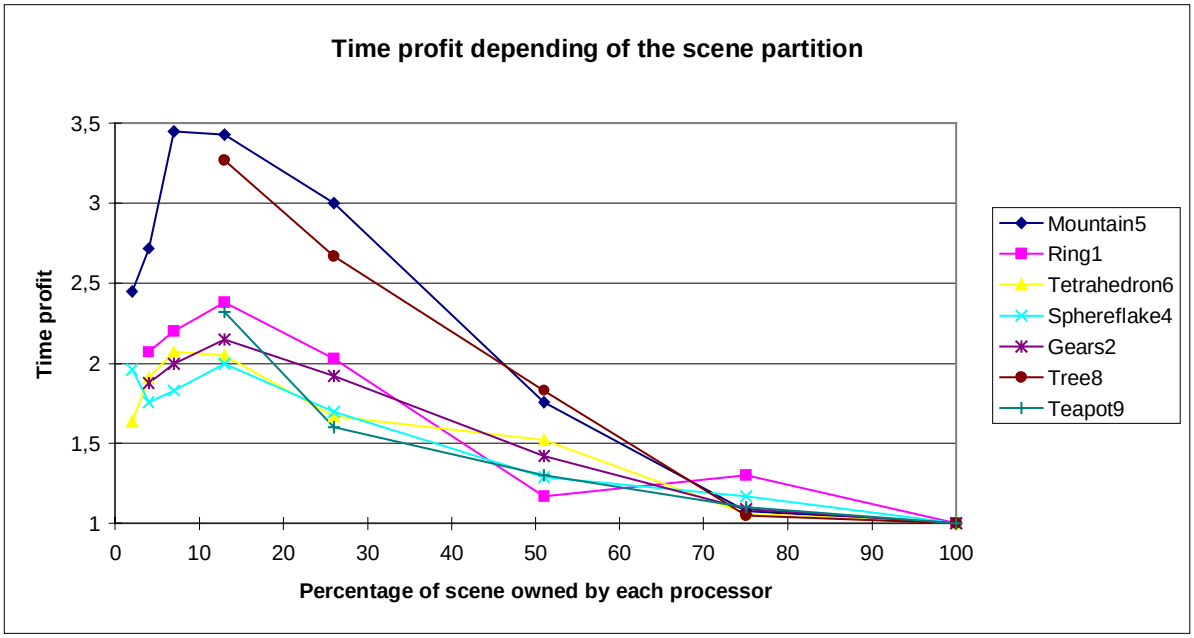
The following is an outline of our mixed dataflow algorithm :

```
IF I receive an object request from processor P
THEN
        A message is initialised
        IF my load is superior to a minimal load
        THEN
                IF I have a sufficient number of rays needing objects owned by processor P
                THEN
                        The rays are moved in the message
                ENDIF
        ENDIF
        The object is copied in the message
        The message is sent to processor P
ENDIF
```

Thanks to this new kind of dataflow, we hope to reduce the number of communications and ensure a dynamic load balancing.

4.2. Experimental results

We compare the results given by our mixed dataflow algorithm with these of the object dataflow algorithm (entire results can be found in [12]). Next figure shows the time profit obtained by the mixed dataflow algorithm for various scenes using 64 processors of the CRAY T3E:

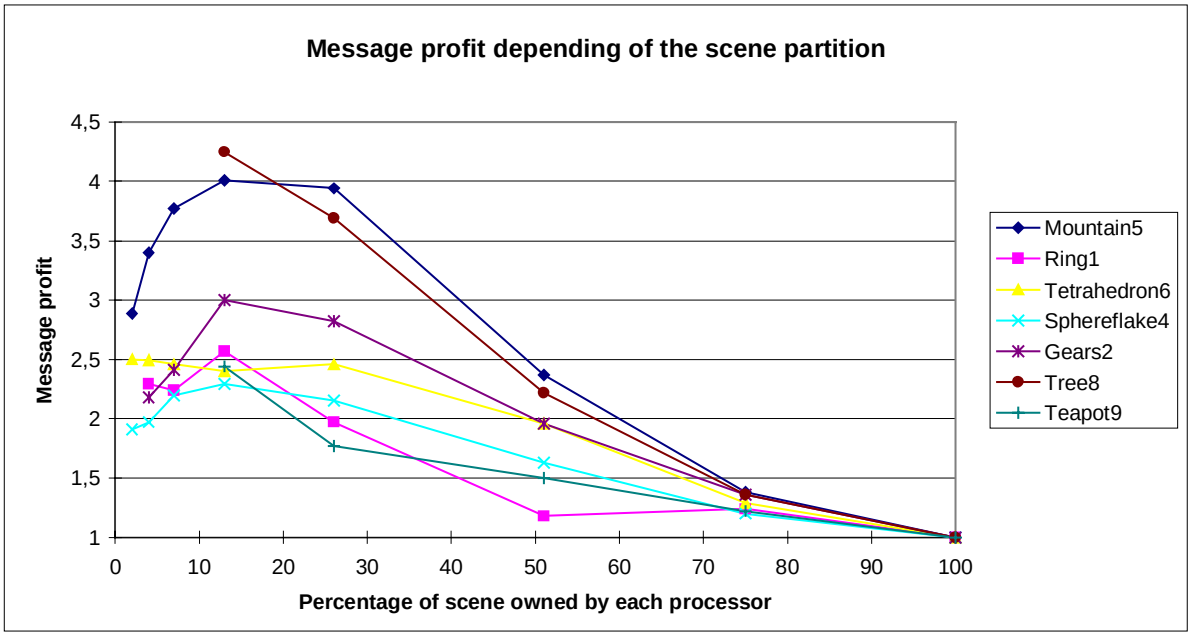


At first it appears that the time profits are not very linked with the type of the rendered scene. The charts have very similar shapes.

Next, time profits may reach very high values, the processing time of the Mountain5 scene has been divided by 3.4! Moreover when processors owned less than the half of the base, profits vary between 1.5 and 3.4. It is much better than the previous balancing scheme (δ3.3) for which profits vary between 1.05 and 1.3.

Our algorithm needs a convenient flow of communications to reach its best efficiency. But when there are too much communications - when less than 10 % of the scene is owned by each processor -, asynchronism stops to be efficient. The time profits provided by our algorithm decrease. Nevertheless they continue to be important (superior to 1.5).

Now we study the message profits induced by this new algorithm.



This figure shows that our mixed algorithm improves computation times thanks to the reduction of the message number. Message profits are between 1.8 and 4.3 when processors owned less than the half of the base. Again we see that waiting for messages decreases profits.

Our mixed algorithm succeeds in reducing the computation time and the number of messages sent during the computation. It fits to MPP architectures because communications are decentralised and it reduces the risk of communication bottlenecks.

5. Conclusion and future work

In this paper, we have presented a new dynamic load balancing scheme for the object dataflow algorithm which fits for MPP. The time profits provided for 64 processors reach 1.4 when only a small part of the database is owned by each processor.

By this way, better profits seem to be limited. So we have presented a new algorithm for parallel ray tracing using mixed dataflow. It gives very encouraging results, seeing that the computation time for a picture has been reduce by 3.4 compared with a classical object dataflow algorithm. Moreover communication flows are strongly reduced.

Thanks to this new type of dataflow used, saturation of networks can be delayed, results on massively parallel machine are improved and computation time is reduced by a significant factor.

We are currently studying behaviours of this scheme on networks of workstations.

Acknowledgements

We thanks Eric Haines for supplying the SPD package [7]. This work was supported by the CEA (Commisariat à l'Energie Atomique) and the EMSE (Ecole des Mines de Saint-Etienne).

References

[1] J. Arvo and D. Kirk. *An introduction to ray tracing*, chapter 5. A survey of ray tracing acceleration techniques, pages 201-262. Academic press, 1989.

[2] D. Badouel. *Schémas d'exécution pour les machines parallèles à mémoire distribuée. Une étude de cas: le lancer de rayon*. PhD thesis, Université de Rennes I - IFSIC, Rennes, October 1990.

[3] D. Badouel, K. Bouatouch and T. Priol. Distributed data and control for ray tracing in parallel. *IEEE computer graphics and applications*, 14(4), pages 69-76, July 1994.

[4] D. Badouel and T. Priol. An efficient parallel ray tracing scheme for highly parallel architectures. *Advances in computer hardware v. rendering, ray tracing audiovisualisation systems. Lausanne, CH, 2-3 September 1990*, pages 93-106, September 1990.

[5] S.A. Green and D.J. Paddon. A highly flexible multiprocessor solution for ray tracing. *The visual computer*, 6(2), pages 62-73, March 1990.

[6] S.A. Green, D.J. Paddon and E. Lewis. A parallel algorithm and tree-based computer architecture for ray traced computer graphics. *Parallel processing for computer vision and display. Leeds, UK, 1988*, January 1988.

[7] E. Haines. A proposal for standard graphics environments. *IEEE computer graphics and applications*, 7(11), pages 3-5, November 1987.

[8] V. Isler, C. Aykanat and B. Ozguc. Subdivision of 3-D space based on the graph partitioning for parallel ray tracing. *Proc. second eurographics workshop on rendering, univ. of Catalonia, Barcelona*, 1991.

[9] H. Kobayashi, S. Nishimura, H. Kubota, T. Nakamura and Y. Shigei. Load balancing strategies for a parallel ray-tracing system based on constant subdivision. *The visual computer*, 4(4), pages 197-209, October 1988.

[10] W. Lefer. An efficient parallel ray tracing scheme for distributed memory parallel computers. *Proc. of parallel rendering symposium (1993), San Jose, California, October 25-26*, pages 77-80, October 1993.

[11] K. Murakami, K. Hirota and M. Ishii. Fast ray tracing. *Fujitsu scientific and technical journal*, 24(2), pages 150-159, June 1988.

[12] J.-C. Nebel. *Développement de techniques de lancer de rayon dans des géométries 3-D adaptées aux machines massivement parallèles*. PhD thesis, Université de Saint-Etienne, Saint-Etienne, October 1997.

[13] I. Pandzic, N. Magnetat and M. Roethlisberger. Parallel ray tracing on the IBM SP2 and CRAY T3D. EPFL - *Supercomputing review*, 7, November 1995.

[14] T. Priol. *Lancer de rayon sur des architectures parallèles : étude et mise en oeuvre*. PhD thesis, IFSIC, Rennes, June 1989.

[15] F.V. Reeth, W. Lamotte and E. Flerackers. Ray tracing speed-up techniques using MIMD architectures. *Programming and computer software*, 18(4), pages 173-181, July 1992.

[16] T. Whitted. An improved illumination model for shaded display. *Communication of the ACM*, 23, pages 343-349, 1980.

[17] N. Wilt. *Object-oriented ray tracing*. Wiley, 1994.