Efficient Load-balancing and Communication Overlap in Parallel Shear-Warp Algorithm on a Cluster of PCs

Frédérique Chaussumier[1], Frédéric Desprez[2], and Michel Loi[1]

[1] LHPC, ENS-Lyon, F-69364 Lyon cedex 07, France
[2] LIP, ENS-Lyon, F-69364 Lyon cedex 07, France
[fchaussu,desprez,mloi]@ens-lyon.fr

Abstract. In the medical field, volume rendering provides good quality 3D visualizations but is still not enough interactive for a day-to-day practice. The most efficient sequential algorithm is the shear-warp algorithm. It renders up to 10 images per second for a small dataset. The goal of this paper is to present an efficient parallel implementation of the shear-warp algorithm for a distributed memory architecture, a cluster of PCs connected with a high speed network. This highly irregular algorithm led us to implement a dynamic load balancing algorithm. Furthermore, to reduce the overhead due to data redistribution, we overlap communications with computations using MPI's asynchronous communications. Using a good load-balancing and communication overlap, our implementation generates real-time 3D medical images with a good quality and a high resolution.

1 Introduction and motivations

Real-time rendering is an important goal in visualization applications. As a matter of fact most of these applications require the generation of a sequence of images for different orientations of the volume. Consequently real-time rendering could enable a continuous visualization of the volume as its orientation changes. Moreover higher and higher resolution datasets combined with the high computational cost of direct volume rendering makes it difficult, if not impossible, for sequential implementations to deliver the required level of performance. Therefore, such applications have been parallelized not to trade off image quality for speed. Nevertheless, Lacroute [Lac95a] developed the shear-warp algorithm that exploits coherence in the volume and image space. This algorithm is currently acknowledged to be the fastest sequential volume rendering algorithm.

The goal of this paper is to present an efficient parallel implementation of the shear-warp algorithm for a distributed memory architecture, a cluster of PCs connected with a high-speed network and using a light weight and fast communication layer. This new parallel implementation is load balanced and overlaps communication with computation using the MPI standard.

This paper is organized as follows : in the first part, we describe and analyze the shear-warp algorithm. The second part exhibits the main problems associated with the parallel formulation. It focuses on architecture, task partitioning and communications patterns. In the third part, we propose a new dynamic load balancing algorithm for the shear-warp algorithm adapted to interactivity. In order to improve the scalability of the algorithm, we discuss, in the fourth part, the possibility of implementing communication overlap in this algorithm. The last part gives the results we obtained for load balancing and scalability.

2 The shear-warp algorithm

To be able to reach real-time performances, we chose to parallelize the shear-warp algorithm because it is reported to be the fastest volume rendering algorithm so far that does not compromise quality, almost 4-7 times faster than an efficient ray-casting algorithm. We focused on the compositing step because it has the highest computational cost.

2.1 Description

Volume rendering [Kau96] is the process of creating a 2D image directly from 3D volumetric data so that no information contained within the data is lost during the rendering process. For example, in computed tomography scanned data, useful informations are not only contained on the surfaces but also within the data. Therefore it must have a volumetric representation, and must be displayed using volume rendering techniques.

Lacroute and Levoy [LL94] described a fast volume rendering algorithm called the shear-warp factorization. It is based on an algorithm that factors the viewing transformation into a 3D shear (parallel to the data slices), a projection to form an intermediate but distorted image, and finally a 2D warp to form an undistorted final image. The shear-warp factorization has the property that rows of voxels in the volume are aligned with rows of pixels in the intermediate image. Consequently, a scanline-based algorithm has been constructed that traverses the volume and the intermediate image synchronously, taking advantage of the spatial coherence present in both volume and image. Lacroute and Levoy optimized the original algorithm by using spatial data structures based on run-length encoding for both the volume and the image and also taking advantage of early ray termination. An implementation running on an SGI Indigo workstation renders a 256^3 voxel data set in one second.

This algorithm is based on three main steps :

1. the computation of the shading lookup table,
2. the projection of the volume data into the intermediate image,
3. the warping of the intermediate image.

The projection of the volume data into the intermediate image dominates the cost of the sequential algorithm. It takes over 80% of the total amount of time for a whole execution [AGS95].

Therefore in this paper we focus on the compositing step which is the projection of the volume data into the intermediate image.

2.2 Analysis

The Run-Length Encoding (RLE) data structure is a sparse data structure that contains only non-transparent voxels for the object and non-saturated pixels for the image. Using a RLE, we skip empty voxels and saturated pixels. This implies that scanlines may have widely different amount of data associated with them.

Data repartitions in both object and image are highly irregular. They depend on the scene and the viewpoint. For instance, Figures 1, 2, and 3 illustrate how the data repartition in the intermediate image depends on the viewpoint. The default viewpoint is the zero-degree rotation angle. We compare the data distribution of respectively 5, 10 and 20-degree rotation angle to the default viewpoint. We can notice that the bigger the rotation is, the more different the data repartition is.

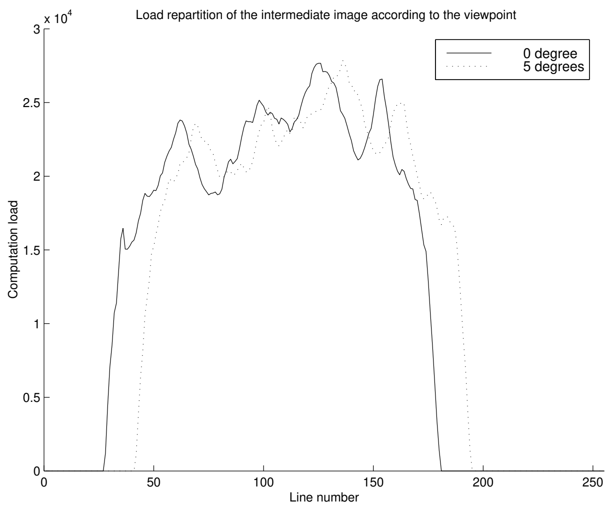

Fig. 1. Data repartition in the intermediate image (5-deg. rotation angle).

Finally, we outline that the shear-warp algorithm leads to a highly irregular application because of the RLE data structure. Accordingly, in the parallel algorithm, computation and communication are very irregular as well.

3 Parallel algorithm

In this section we exhibit the main problems associated with the parallel implementation of the shear-warp algorithm. The shear-warp algorithm augmented

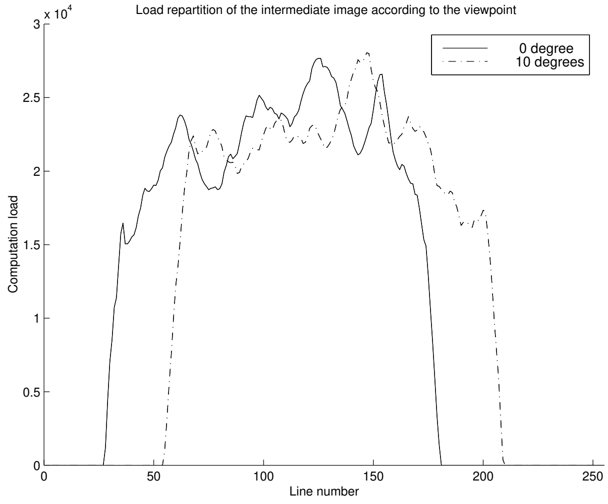

Fig. 2. Data repartition in the intermediate image (10-deg. rotation angle).

with early ray termination and run-length encoding yields to an excellent per-frame sequential rendering times. This algorithm forms the basis of our parallel formulation. The critical issues in any parallel algorithm are concurrency, minimization of communication overhead, and a good load-balancing among processors.

3.1 Related work

Some authors have already proposed parallel formulations of this algorithm for both shared and distributed memory architectures.

Architecture To get real-time performance, both Lacroute [Lac95b] and then Jiang and Singh [JS97] parallelized the shear-warp algorithm on a a 16-processor SMP SGI Challenge. Unlike distributed memory architectures, this architecture supports fine-grain and low-latency communications adapted to the irregular communication and computation patterns of the shear-warp algorithm. They render a 256^3 voxel data set at over 10 frames per second.

Amin et al. [AGS95] have implemented the shear-warp algorithm for a distributed memory architecture. With a 128-processor TMC CM5, they could render a 256^3 voxel data set at 12 frames per second. It is comparable to the results obtained on the 16-processor shared memory architecture. Nevertheless, they restricted the utilization of the shear-warp algorithm by allowing only one-degree rotations to change the viewpoint. Despite of this restriction their algorithm is not scalable: the speedup is 30 for 128 processors.

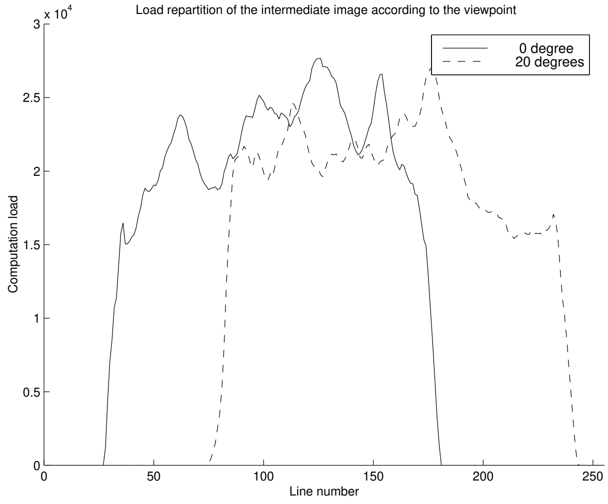

Fig. 3. Data repartition in the intermediate image (20-deg. rotation angle).

Task shaping The two general types of task partitions for parallel volume rendering algorithms are object partitions and image partitions. In an object partition, each processor gets a specific subset of the volume data to resample and composite. The partial results from each processor must then be composited together to form the image. In contrast, using an image partition each processor has to compute a specific portion of the image. Each image pixel is computed only by one processor, but the volume data must be moved to different processors as the viewing transformations changes. As a matter of fact, it is very important not to limit the size of the data volume. In the medical field, standard volumes are composed of 512^3 voxels, which means at least 135 Mbytes without compression. Thus, we chose to distribute the data on every processor because the replication for such volumes is impossible on standard machines. All existing implementations have designed their parallelization using an image partition that takes full advantage of the optimizations in the rendering algorithm. Moreover, for a shared memory architecture data motion is less significant. The partitioned image is the intermediate image created during the shear step. Then the unit of work can be individual pixels, scanlines of pixels, or rectangular pixels. In [Lac95b], it is shown that the best shape is scanlines of pixels because it minimizes the overhead due to decoding the run-length data structure. It also maximizes the spatial locality both in the intermediate image space and object space.

Load-balancing Given that the fundamental unit of work is a group of contiguous scanlines of the intermediate image, minimizing load imbalances gives three options: a static contiguous partition, a static interleaved partition and a dynamic partition. For a shared memory architecture, Lacroute chose to use a distributed task queue and a dynamic stealing. This solution is too expensive for a distributed memory architecture. It generates a prohibitive communication overhead. Consequently for such an architecture Amin et al. determined heuristics based on adaptative load balancing scheme. But because their utilization restriction that considers only one degree rotations they finally conclude that they only needed a static load balancing.

Our approach is to implement the Shear-Warp algorithm on a distributed memory architecture because of the good scalability of this type of architecture. On the one hand one of our major goals is to achieve real-time performances with higher resolution data sets (particularly 512^3). On the other hand, we believe it is important not to restrict the user utilization and to allow him to change arbitrarily the viewpoint. Thus, our new implementation proposes a dynamic load-balancing that do not depend on the previous rendering.

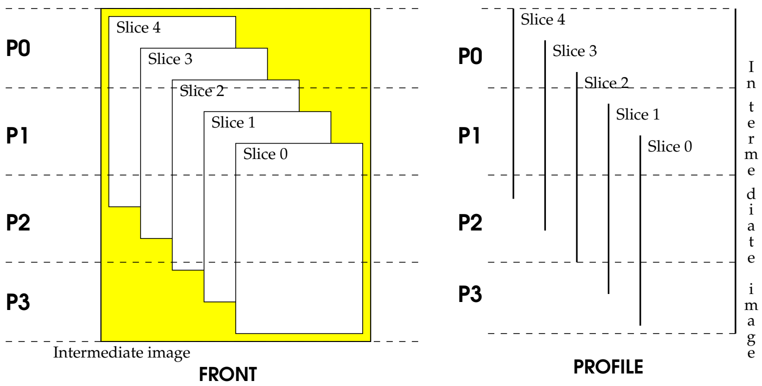

Fig. 4. Volume distribution in an image partition.

3.2 Parallelization

The decreasing costs and increasing performance of both computer hardware and network now offer a great potential for distributed network computing. Furthermore, one of the major benefits of distributed computing is its scalability in terms of the amount of computing power and resources available for large-scale applications. Those are the reasons why we chose to parallelize the shear-warp algorithm on a cluster of PCs interconnected with a high-speed Myrinet network from Myricom. Because of the distributed memory architecture, we had to determine an explicit data distribution (and redistribution) that minimizes communication but keeps a good load-balancing.

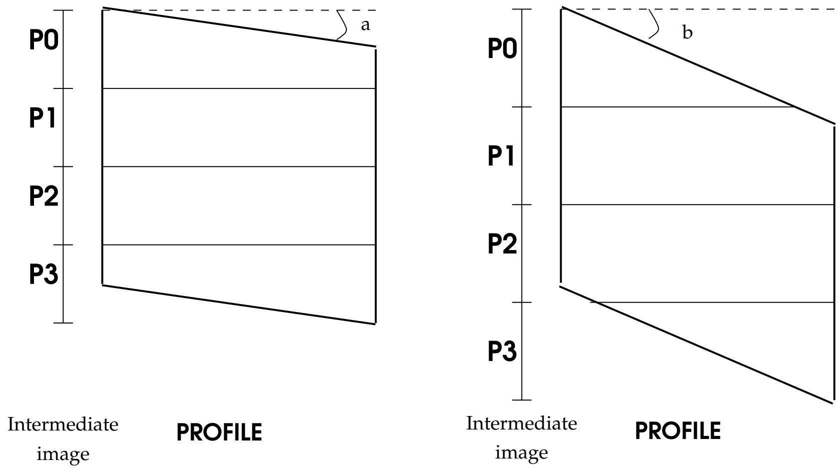

Fig. 5. Shearing the volume from a to b.

Data distribution Explicit data distribution is a difficult problem when an image partition is used because the portion of the volume required by a particular processor depends on the viewpoint. One naive solution is to replicate the data in every processor's memory, but this design severely limits the maximum size of the volume and does not solve the redistribution problem.

To distribute data volume in an intermediate image partition the volume is first sheared and then distributed by slices orthogonal to the rays. Each processor can now compute its portion of the intermediate image through its assigned volume segment. The resulting intermediate images on different processors are disjoint and can be independently warped. Figure 4 shows a simple intermediate image partition with 4 processors. The corresponding sheared volume, made up of 5 slices, is partitioned as illustrated on the figure : processor 3 owns a few scanlines in the first slice, processor 2 owns scanlines in every slice, ...

Generated communications The main overhead of this algorithm results from communications of volume data when the volume is sheared. Figure 5 shows the deformation of the volume and its corresponding intermediate image when the shear changes from a to b. The generated communications are illustrated in Figure 6. Every processor receives data corresponding to the shaded scanlines from its neighbor processors except the first and the last ones.

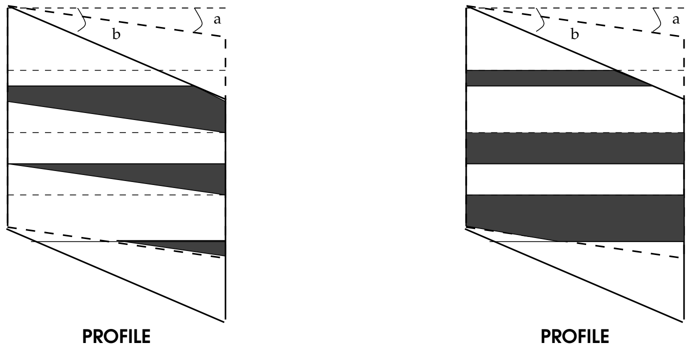

Fig. 6. Data received respectively from previous and next processors.

Communication patterns In our parallel Shear-Warp algorithm, we need two types of communications:

- A gather of partial images into the final image, and
- a personalized all-to-all communication for the data redistribution when the viewpoint has changed.

We implemented the personalized all-to-all communication in $p - 1$ steps, where p is the number of processors, as follows: at each step each processor sends data to a step-far processor in the increasing processor number and receives data from a step-far processor in the decreasing processor number. Figure 7 illustrates this scheme for 4 processors.

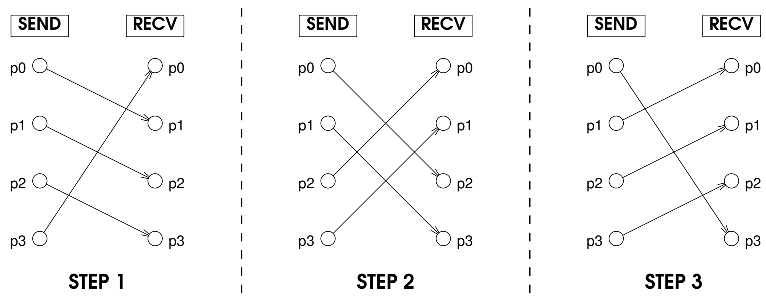

Fig. 7. Personalized all-to-all communication for 4 processors.

Overall Algorithm We implemented the overall algorithm but we only focused on the data distribution and redistribution and the composition that are written in italic (that takes most of the computation time).

```
Procedure Render()
  InitialDistribution()
  Foreach viewpoint do
    Computation of a part of the shading lookup table (LUT)
    Multidistribution of the shading LUT
    (* Each processor now owns the whole LUT. *)
    (* Each processor owns the parts of each slice *)
    (* that are necessary for its parts of computation. *)
    Foreach voxels' slices from front to back do
      If I own the data
        Composite(data, part_image)
    EndFor
    image = Warp(part_image)
    Gather(image, root)
    If p == root
      Display(image)
    Personalize-all2all(volume)
  EndFor
End
```

4 Dynamic Load-Balancing

The requirement of an arbitrary rotation of the voxels' cube implies that we should implement a dynamic load balancing mechanism. As we shown in Figures 1, 2, and 3, load-balancing strictly depends of the viewpoint.

In an image partition, every processor has to compute a specific portion of the image. This portion of image results from the projection of the volume data into this portion of image. Therefore, a naive partitioning of the image that assigns an equal portion to each processor yields to a bad load-balancing. Furthermore, it is impossible to determine in a static way the accurate amount of voxels needed to generate this portion.

Consequently, we used the elastic load-balancing algorithm given in [MR91] to determine the load and to get a good load-balancing accordingly. This algorithm consists in computing a local partial load for each processor. Then each processor broadcasts its partial value and adds its value with the ones received. At this moment, every processor knows the global load. By dividing this global load by the number of processors, each processor finds its elementary load. Then every processor has to get the data necessary to its computation.

In the Shear-Warp algorithm, every processor has to compute an array containing its local contribution for each line of the intermediate image. This array is then broadcasted. Each processor adds the arrays received with its own. The resulting array contains the computational load repartition for each line of the intermediate image. They can then obtain the global load. By linearly distributing the intermediate image, they can balance the load through the processors.

5 Overlapping communication with computation

In order to reduce the overhead due to data redistribution, we studied the possibility of introducing communication overlap in the compositing phase. Because of the irregularity of the application, this presents a considerable challenge.

In addition to the irregular communication and computation patterns of the shear-warp algorithm, we had to deal with communication layers. As a matter of fact, none of our implementations of the MPI standard provide a real asynchronous communication routine.

5.1 Communication and computation patterns

So far every communication generated by the data redistribution is done before the volume composition as shown by the following pseudo-code:

```
For k=0, nbSlices do
  For step=0, nbProcesseurs do
    If must-send(k, me+step)
      send(block_scanlines, me+step)
    If must-receive(k, me-step)
      receive(block_scanlines, me+step)
    EndFor
EndFor

For k=0, nbSlices do
  Composition(block_scanlines(k))
EndFor
```

Consequently it is possible to overlap the communication of the slice $k+1$ with the composition of the slice k. Figure 8 shows an example of communication and computation pattern with 4 processors and 3 slices. Without overlap, the processor waits for each slice for every processor's data before starting the computation. It is represented by the case a. The first overlap step (case b) consists in starting the computation of a slice as soon as every processor sent its corresponding data. Then in the figure, we have improved the total execution time of A. The second overlap step (case c) waits for a processor to send its data and begins immediately its computation corresponding to the received part of slice. Then in the figure, we have improved the total execution time of B ($B > A$).

5.2 Experimental platform

The cluster of PCs we used has 8 PowerPC 604e clocked at 200 Mhz interconnected with a high speed network Myrinet.

The communication layer BIP (Basic Interface for Parallelism) [Pry98] implemented on the Myrinet network delivers the maximal performance achievable by the hardware to the application. A Myrinet host interface is based on a LANai

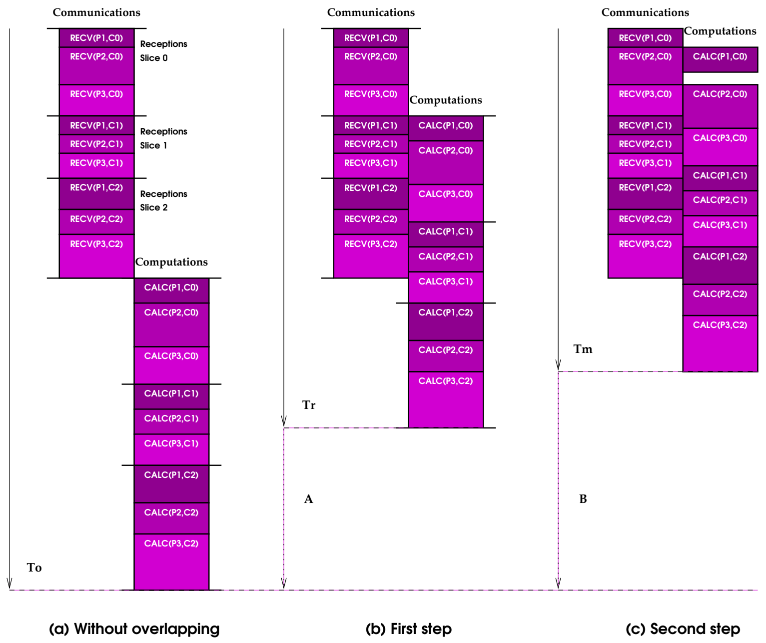

Fig. 8. Two possible ways to overlap communication with computation.

chip (containing a processor, a packet interface and a DMA) and SRAM memory of 128 KB. With the BIP communication layer most of the communication management is handled by the LANai.

The LAM implementation of the MPI standard for a cluster of PCs does not provide communication overlap whereas BIP provides communication overlap thanks to its simple interface and to the communication processor (LANai). Figure 9 and 10 illustrate respectively the capability of the MPI standard and native BIP to overlap communications by computations on our target machine. The test program is executed by two processors. They first exchange data and execute some computation. The exchanged message sizes vary from 0 to 3 KB. The computation time for both blocking and non blocking versions is obviously the same. For both MPI and native BIP execution, we first execute blocking communications, then we deduce the ideal curve and execute non-blocking communications. Using MPI, the non-blocking curve is comparable to the blocking curve. With native BIP the non-blocking curve is comparable to the ideal curve.

5.3 Implementation

The BIP interface only allows one communication at a time to take a full advantage of the Myrinet network. Because of this restrictions, we only could implement the second possibility of overlapping presented in Section 5.1.

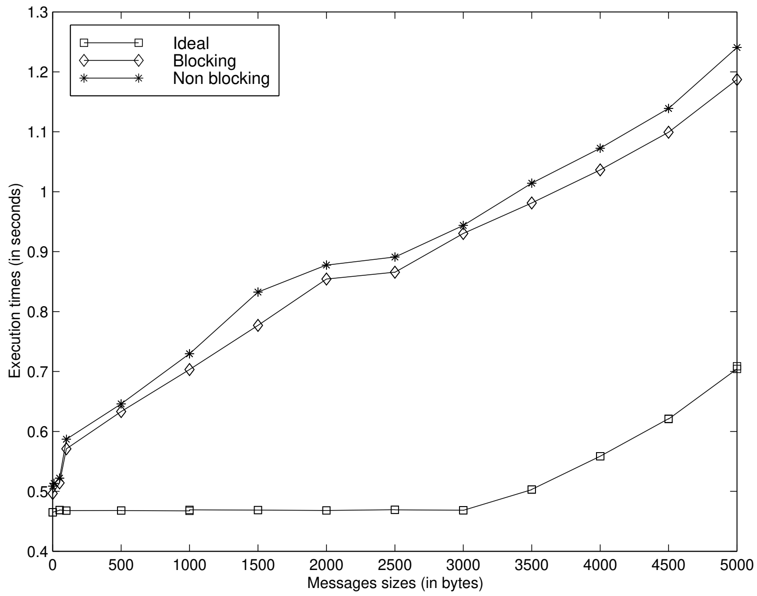

Fig. 9. Overlapping communication with MPI

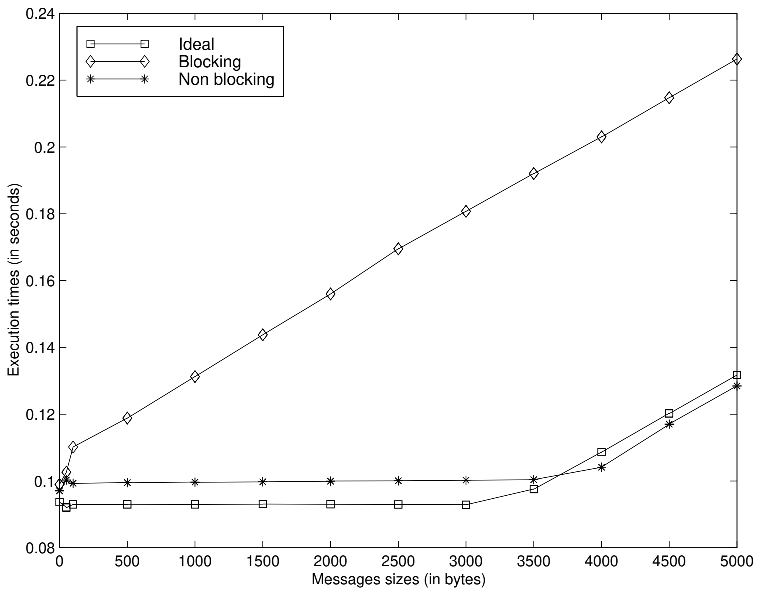

Fig. 10. Overlapping communication with respectively native BIP

6 Results

6.1 Efficient load balance

Figures 11 and 12 compares respectively the workload of each processor for an execution with processors respectively in the case of a static allocation and a dynamic redistribution.

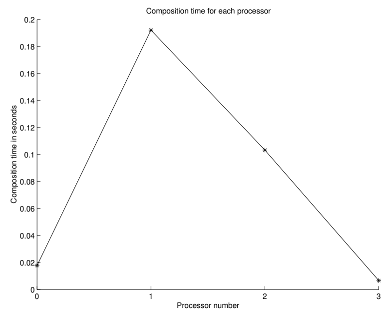

Fig. 11. Workload of each processor: static allocation.

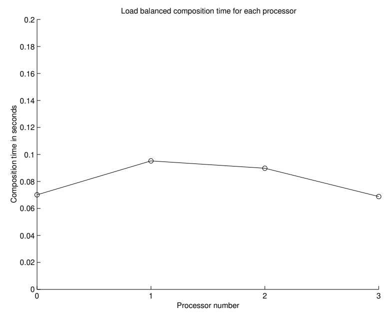

Fig. 12. Workload of each processor: dynamic redistribution.

Using a static allocation, we distribute the slices with a block-cyclic distribution the lines. Figure 11 shows that the central processors have the whole load. On the contrary, with the dynamic load balancing algorithm, the data is well balanced. The slight variations in Figure 12 are due to the granularity of the data.

6.2 Scalable composition

In our parallel implementation, we focused on the compositing step. We first implemented the compositing step using blocking MPI primitives. The very bad scalability of this implementation, as shown in the Figure 13, is due to data redistribution overhead. We therefore decided to implement communication overlap. The figure shows that the implementation using asynchronous communications is almost perfectly scalable. We have a very good overlap of the communications because of the BIP layer and because we could find independent computation and communication in the Shear-Warp algorithm.

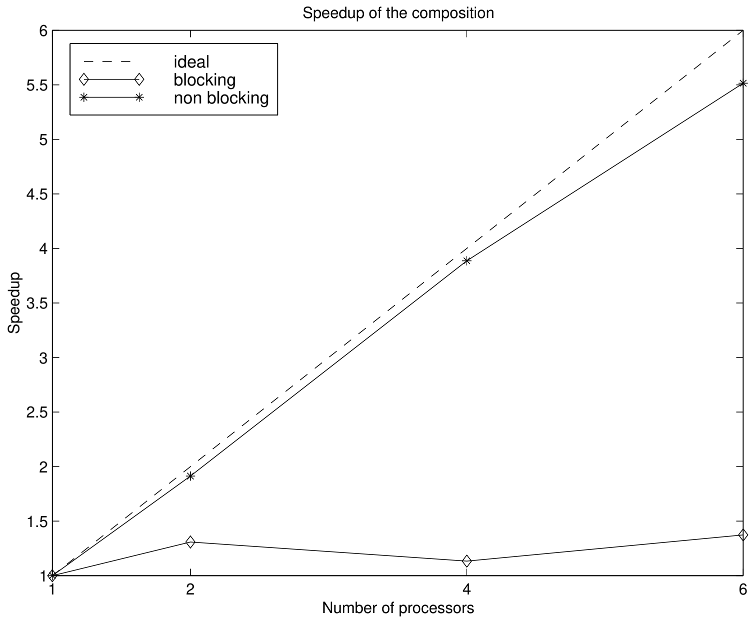

Fig. 13. Speedup of the compositing phase.

Those curves are obtained on a colored high resolution dataset, 512^3 voxels. The total execution time for such a dataset is 1.5 s on 4 processors.

7 Conclusion

The first goal of this application is to provide to physicians a good quality and resolution visualization from medical datasets in real-time with a low-cost distributed memory machine.

In this paper we have presented an high performance and scalable version of the Shear-Warp algorithm implemented on a cluster of PC. Our parallel approach of the Shear-Warp algorithm improves the interactivity of the application by using an adapted load balancing algorithm and by overlapping communication. It then allows the user to get a 3D representation with any viewpoint in real-time. Even with a sparse data-structure and irregular communication patterns, we are able to get performances that are comparable to implementations on "classical" parallel machines.

The optimizations presented in this paper can also be used in other irregular applications, and thus we would like to create a library to overlap communications and computations for this kind of applications.

References

[AGS95] Minesh B. Amin, Ananth Grama, and Vineet Singh. Fast Volume Rendering Using an Efficient Parallel Formulation of the Shear-Warp Algorithm. In *Proceedings 1995 Parallel Rendering Symp.*, 1995. `ftp://ftp.cs.umn.edu/dept/users/kumar/parallel_rendering.ps`.

[JS97] Dongming Jiang and Jaswinder Pal Singh. Improving Parallel Shear-Warp Volume Rendering on Shared Adress Space Multiprocessors. In *Proceedings of the 1997 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 1997.

[Kau96] Arie E. Kaufman. Volume Visualization. *ACM Computing Surveys*, 28(1):165–167, March 1996.

[Lac95a] Philippe Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of The Viewing Transformation.* PhD thesis, Stanford University, 1995. `http://www-graphics.stanford.edu/~lacroute/`.

[Lac95b] Philippe Lacroute. Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization. In *Proceedings of the 1995 Parallel Rendering Symposium*, 1995. `http://www-graphics.stanford.edu/~lacroute/`.

[LL94] Philippe Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Computer Graphics*, volume 28, pages 451–458. Stanford University, July 1994. `http://www-graphics.stanford.edu/~lacroute/`.

[MR91] Serge Miguet and Yves Robert. Elastic Load Balancing for Image Processing Algorithms. In H.P. Zima, editor, *Parallel Computation*. First International ACPC Conference, 1991.

[Pry98] Loïc Prylli. *BIP Messages User Manual for BIP 0.94*, June 1998. `http://www-bip.univ-lyon1.fr/bip.html`.