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

Frédérique Chaussumier, Frédéric Desprez, Michel Loi

Laboratoire de l'Informatique du Parallélisme

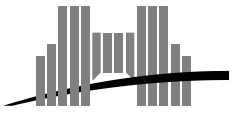
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

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

Frédérique Chaussumier
Frédéric Desprez
Michel Loi

May 1999

Research Report N° 99-28



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

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

Frédérique Chaussumier
Frédéric Desprez
Michel Loi

May 1999

Abstract

In the medical field, volume rendering provides good quality 3D visualizations but it is still not interactive enough 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 report 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.

Keywords: Volume rendering, Shear Warp algorithm, dynamic load-balancing, communication overlap.

Résumé

Dans le domaine médical, le rendu volumique offre des visualisations 3D de bonne qualité mais il n'est pas suffisamment intéractif pour une utilisation quotidienne. L'algorithme séquentiel le plus efficace est l'algorithme du Shear-Warp. Il peut rendre jusqu'à 10 images par seconde pour un petit jeu de données. Le but de ce rapport est de présenter une implémentation parallèle efficace de l'algorithme du Shear-Warp pour une architecture à mémoire distribuée, un cluster de PC connectés avec un réseau rapide. Cet algorithme très irrégulier nous a conduit à implémenter un algorithme d'équilibrage des charges dynamique. De plus, afin de réduire le surcoût des redistributions, nous recouvrons les communications avec les calculs en utilisant les communications asynchrones de MPI. Avec l'équilibrage des charges dynamique et le recouvrement des communications, notre implémentation génère des images médicales 3D en temps réel avec une bonne qualité et une haute résolution.

Mots-clés: Rendu volumique, algorithme du Shear-Warp, équilibrage des charges.

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

Frédérique Chaussumier and Michel Loi
LHPC
ENS-Lyon, 46 Allée d'Italie
69364 Lyon cedex 07
France
(fchaussu,mloi)@ens-lyon.fr

Frédéric Desprez
LIP et INRIA Rhône-Alpes
ENS-Lyon, 46 Allée d'Italie
69364 Lyon cedex 07
France
Frederic.Desprez@inria.fr

May 1999

*A shorter version of this report appears in the proceedings of the EuroPAR'99 conference.

1 Introduction and motivations

Real-time rendering is an important goal in visualization applications. 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. Lacroute [6] 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.

Direct volume rendering techniques are effective tools for exploring 3D scalar data. Unlike surface-rendering methods, direct volume-rendering methods can be used to visualize 3D scalar data without conversion to intermediate geometric primitives. By assigning appropriate colors and opacities to the scalar data, one can render objects semi-transparently to expand the amount of 3D information available at a fixed position. Volume-rendered images can also be surimposed upon surface-oriented icons or textures allowing simultaneous scalar and vector field composite visualizations.

Representing a surface contained within a volumetric data set using geometric primitives can be useful in many applications, however there are several main drawbacks to this approach. First, geometric primitives can only approximate surfaces contained within the original data. Adequate approximations may require an excessive amount of geometric primitives. Therefore, a trade-off must be made between accuracy and space requirements. Second, since only a surface representation is used, much of the information contained within the data is lost during the rendering process. For example, in Computer Tomography, useful informations of scanned data are contained not only on the surfaces, but within the data as well. Therefore, it must have a volumetric representation, and must be displayed using volume rendering techniques.

The goal of this report is to present an efficient parallel implementation of the Shear-Warp algorithm on a distributed memory architecture, i.e. 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 communications with computations using asynchronous communications.

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

2 The Shear-Warp volume-rendering 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, and which is 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 Volume-rendering algorithms

Volume rendering [5] 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.

Volume-rendering can be achieved using an object-order, an image-order or a domain-based technique. Object-order volume rendering techniques use a forward mapping scheme where the volume data is mapped onto the image plane. In image-order algorithms, a backward mapping scheme is used where rays are cast from each pixel in the image plane through the volume data to determine the final pixel value. In a domain-based algorithm, the spatial volume data is first transformed into an alternative domain, such as compression frequency and wavelet, and then a projection is generated directly from that domain.

2.1.1 Object-order algorithms

Object-order techniques involve mapping the data samples onto the image plane. One way to accomplish a projection of a surface contained within the volume is to loop through the data samples, projecting each sample which is part of the object onto the image plane. For instance, the splatting method [12] is an object-technique technique (see [4] for an analysis of algorithms on the Cray T3D). Splatting is accomplished by projecting each voxel onto the view plane in a front-to-back traversal with respect to the view plane. With a parallel projection, the footprint (or projection) of a voxel on the view plane will be constant for all voxels. Since a voxel will not project onto exactly one pixel, a filter is used to spread out the color and transparency to neighboring pixels.

2.1.2 Image-order algorithms

Image-order volume-rendering techniques are fundamentally different from object-order rendering techniques. Instead of determining how a data sample affects the pixels on the image plane, in an image-order technique we determine for each pixel on the image plane which the data samples contribute to it.

The most famous image-order technique is the ray-tracing technique [2]. In the ray-tracing method, rays are shot from the eye location and for each ray, the values of color and opacity are found at evenly spaced intervals along the ray by trilinearly interpolating from the color and opacity values at the eight corners of the voxel within which the ray is sampled. The color of a particular pixel is obtained by accumulating the values of color as the path along the ray that is shot from that pixel is traversed.

The two characteristics that distinguish object-order and image-order algorithms are the order in which an algorithm traverses the volume and the method used by an algorithm to project the voxels to the image. In image-order algorithm, outer loops iterate over the pixels in the image whereas outer loops of object-order algorithms iterate over voxels in the object being rendered. Then each type of algorithm has performance advantages and drawbacks and some of them are more compatible to particular acceleration techniques. The main disadvantage of image-order algorithms is that they do not access the volume in storage order since the viewing rays may traverse the volume in an arbitrary direction. Therefore, those algorithms spend more time calculating the location of sample points (for instance the voxels indices in the innermost loop body). Acceleration techniques based on spatial data structures further aggravate this problem. A second problem is that this type of algorithms have higher memory overhead because they do not have good spatial locality. In contrast, object-order algorithms can stream through the volume in storage order. However, accurately computing the filter footprint and the resampling weights is expensive because the filter

is view dependent. In a forward-projection algorithm, the filter footprint must be scaled, rotated and transformed arbitrarily depending on the viewpoint. Moreover, in a perspective projection view, this footprint changes from voxel to voxel. Thus, it is difficult in an object-order algorithm to implement a filter that is both efficient and that produces high quality results.

2.1.3 Acceleration techniques

The two main acceleration techniques that do not affect image quality are coherence acceleration using spatial data structures and early ray termination [7]. We do not want to trade off image quality for speed then we do not consider techniques like subsampling the volume for instance.

An established acceleration technique for volume rendering is to exploit coherence in the volume by using a spatial data structure. For a given visualization data set, typically there are clusters of voxels that give useful informations to the image and other clusters that are irrelevant. The purpose of a spatial data structure is to encode this type of coherence getting rid of irrelevant voxels. Rendering algorithms use data structures like octrees, pyramids, run-lengths encoding, to be able to skip transparent voxels rapidly.

A second common acceleration technique for volume rendering is a technique called early ray termination. This optimization is most easily implemented in a ray tracing algorithm: the algorithm traces a ray in front-to-back order and terminates the ray as soon as the accumulated ray opacity reaches a threshold value close to full opacity. The goal of this optimization is to reduce or eliminate samples in occluded regions of the volume.

To conclude with, object-order algorithms can efficiently traverse a spatial data structure to find the non-transparent voxels, but resampling is complicated because the filter is view dependent. Image-order algorithms must perform more work to traverse the spatial data structure but resampling is simpler and these algorithms can take full advantage of early-ray termination.

2.2 Shear-Warp sequential algorithm

Lacroute and Levoy [8] described a fast volume rendering algorithm called the Shear-Warp factorization. It is based on an algorithm that factors the viewing transformation. This results in an efficient projection voxels to image.

The Shear-Warp factorization relies on the transformation of the volume to an intermediate coordinate system called the "sheared object space". By construction, in sheared object space, all viewing rays are parallel to the third coordinate axis. An object-order algorithm based on this transformation is composed of two main steps:

1. a compositing step called *Shear* that transforms the volume data to sheared object space by translating each slice and then composites the resampled slices together in a front-to-back order. This step projects the volume into a 2D distorted image in sheared object space;

2. a second step called *Warp* to transform the distorted image to image space by warping it. This second resampling step produces the correct 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.

2.2.1 Application analysis

The implementation of the Shear-Warp algorithm can be divided in three functional units: computation of a shading lookup table, the compositing step and the warp of the intermediate image (see the pseudo-code below). Amin et al. [1] give an analytical model of these three steps.

```
Procedure Rendering()
  Foreach viewpoint do
    Computation the shading lookup table
    Foreach slice in the volume do
      Composite(slice, tmp_image)
    End foreach
    image = Warp(tmp_image)
    Display(image)
  End foreach
```

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 [1].

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

2.2.2 Data structure

Lacroute and Levoy optimized the original algorithm by using spatial data structures based on run-length encoding (RLE) for both the volume and the image. This encoding is used for volume that have a high data coherency, it results in a good compression. It is actually the case for images obtained with MRI[1] and CT[2] that are considering here. The volume in the Shear-Warp algorithm is a stack of 2D slices. We define a principal coordinate axis that is perpendicular to the slices and each slice is an array of voxels determined by their opacity. We also define a low threshold value of opacity. A voxel in the volume that has a smaller opacity than the threshold is considered transparent and does not bring any information to the final image. Symmetrically, we define a high threshold value of opacity. A pixel in the image that has a higher opacity value is considered as saturated. In the Shear-Warp, we do not want to store either transparent voxels or saturated pixels.

The RLE 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.

2.2.3 Performance analysis

The good performances of the Shear-Warp algorithm are due to compositing method along the rays. As a matter of fact, the factorization of the viewpoint transformation makes possible the simultaneous scan of both the image and the volume then the disadvantages of image-order algorithm are eliminated. Moreover, MRI or CT images contain up to 70 percent of transparent voxels ([11] describes the main medical acquisition modalities and their results). The RLE data structure for the volume allows to skip transparent voxels then computation data structure scan are reduced. Moreover the image is also run-length encoded to skip saturated pixels. When a pixel is saturated no computation is done; it is the implementation of the early ray termination. Both techniques are

[1] Magnetic Resonance Imaging.
[2] Computer Tomography.

Figure 1: Rendering example.

combined; consequently the Shear-Warp algorithm developed by Lacroute is ten times faster than the original one [6].

Thanks to all these optimizations, an implementation running on an SGI Indigo workstation renders a 256^3 voxel data set in one second. A rendered image is given in Figure 1.

2.3 Shear-Warp parallel algorithm

In this section, we point the main problems associated with the parallel implementation of the Shear-Warp algorithm. The Shear-Warp algorithm yields to excellent per-frame sequential rendering times. Nevertheless, despite of the good performances of the sequential Shear-Warp algorithm, the computational load is too high for real-time rendering. Moreover, the image is totally recomputed every time the viewpoint changes. That is the reason why some authors have proposed parallel formulations in order to reduce the computation time and handle the resolution of higher data sets.

Those previous papers have focused on the target machine implementation. To get real-time performance, both Lacroute [7] and then Jiang and Singh [3] parallelized the Shear-Warp algorithm on 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. [1] 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.

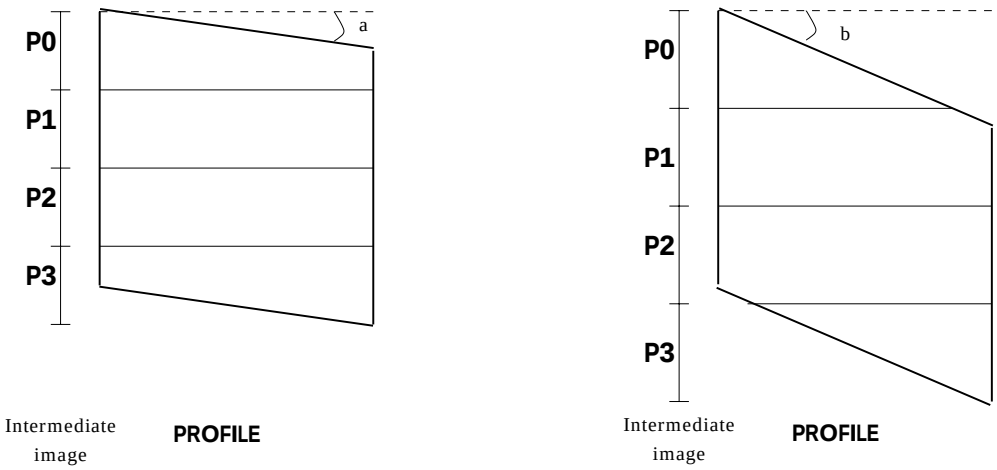


Figure 2: Shearing the volume from a to b.

However, 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. We think that this bad speedup can be improved.

The algorithm augmented with early ray termination and run-length encoding forms the basis of the parallel formulation. The critical issues in any parallel algorithm are concurrency, minimization of communication overhead, and a good load-balancing among processors.

2.3.1 Previous parallelizations

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 to 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 movements are 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 [7], 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.

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 the memory of every processor, but this design severely limits the maximum size of the volume and does not solve the redistribution problem.

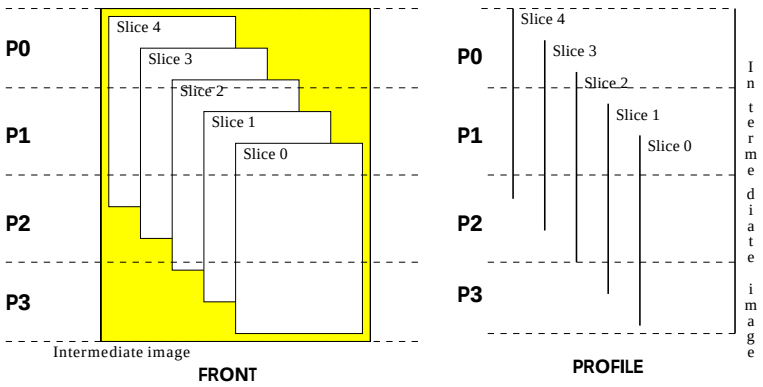


Figure 3: Volume distribution in an image partition.

To distribute data volume in an intermediate image partition the volume is first sheared and then distributed by slices that are orthogonal to the rays. Now each processor can 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 3 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, ...

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 an 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 its good scalability. On 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 that 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.

2.3.2 Communication analysis

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 benefit 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 also that ensures a good load-balancing.

Figure 4: Data received respectively from previous and next processors.

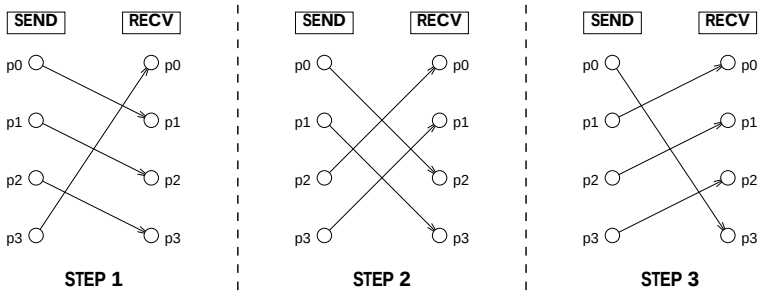


Figure 5: Personalized all-to-all communication for 4 processors.

The main overhead of this algorithm results from communications of volume data when the volume is sheared. Figure 2 shows the deformation of the volume and its corresponding intermediate image when the shear changes from a to b. The generated communications are given in Figure 4. Every processor receives data corresponding to the shaded scanlines from its neighbor processors except the first and the last ones.

Communication patterns In our parallel Shear-Warp algorithm implementation, 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 5 illustrates this scheme 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 take 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)
    Personalized-all2all(volume)
  EndFor
End
```

3 Optimization

Volume-rendering algorithms are well-adapted to data-parallelism because of the huge amount of data. In this case, computation phases are followed by communication phases and so on. The previous Shear-Warp parallelizations follow this scheme. Executing several renderings, compositing and warp phases are followed by communication phases where data necessary for the next rendering are transfered. Moreover, those phases are synchronous and thus every processor executes the communication phase at the same time and waits until every other processor has finished to begin the next computation phase. Consequently, the parallelization overhead is mainly due to communications and the program is not scalable. To reduce this overhead we have to carefully choose a data distribution that minimizes communications. Moreover, because this distribution changes as the viewpoint changes, another solution consists in overlapping communications with computations. This is only possible when computations are independent from the communications to overlap. In this part, we study the possibility of using such a technique. This section presents the different parallelization choices we made to parallelize the Shear-Warp on a pile of PCs, the measures we made to evaluate those choices.

The first improvement made on the existing parallel versions is to release the utilization constraints. We are indeed looking forward to generate an image independently of the previous image and viewpoint. Secondly, we want to reduce as much as possible the memory utilization in order to use bigger datasets.

The second optimization consists in improving the scalability by implementing the overlap between computations and communications every time a gain can be obtained.

Step	Time in ms
Shading	34
Compositing	755
Warp	42
Total	851

Figure 6: Sequential execution time.

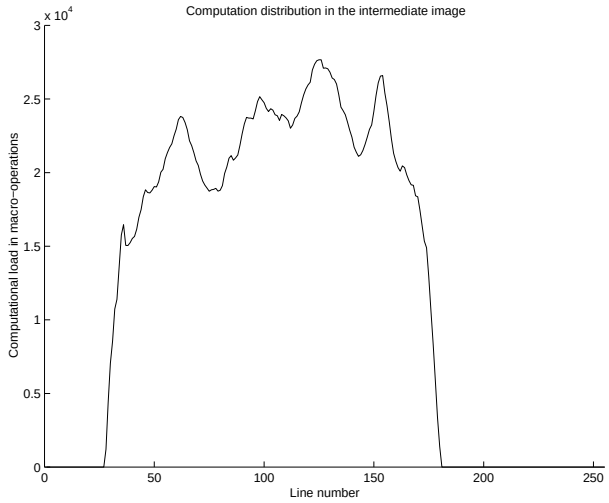


Figure 7: Computation distribution in the intermediate image.

3.1 Quantitative analysis

3.1.1 Sequential behaviour

We executed the Shear-Warp algorithm on several data sets. The following measurements concern a pretty small data set of 15 MB, 256^3 voxels in the volume.

In Figure 6, we measured the execution time for every functional step. Shading, Compositing and Warp times represent less than 10% of the total amount of time.

The compositing step The compositing step is the projection of the data volume on the intermediate image. It consists in resampling and accumulating an opacity along a ray from the image plan to the observer. It is a loop on every voxel of each slice. The sequential algorithm simply scans the run-length encoded volume because rays are parallel to the main axis (thanks to the shear step).

The computational load is highly irregular in the image to be composited. Figure 7 shows the computation distribution in macro-operations in the intermediate image.

Moreover, the run-length encoding structures brings additional irregularity. On the other hand if the volume is divided in slices, then each slice may have widely different amount of data associated with it. As a matter of fact, the first or last slices of the volume are often nearly empty (it corresponds to the beginning and the end of the data acquisition) while central slices have the maximum amount of data. Moreover, the RLE size is proportional to the amount of significant data. Consequently, the first slice might only get a few bytes while a central slice might be nearly one Megabyte large.

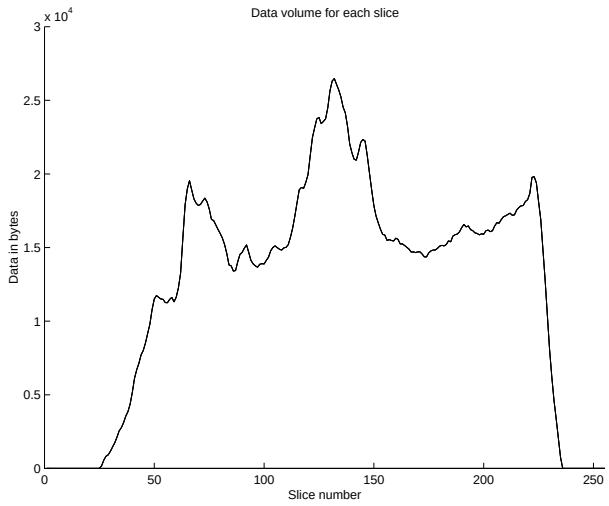


Figure 8: Data volume for each slice.

In the sequential algorithm, the irregular distribution of the computation of every line of the intermediate image is entirely due to original data whereas the computation distribution of the computation for each slice is due to the RLE data structure. The parallel version of the Shear-Warp emphasizes the irregularities. That is the reason why load-balancing is necessary.

3.1.2 Parallel behaviour

The Shear-Warp algorithm augmented with early ray termination and run-length encoding forms the basis of the parallel formulation. We distribute the computation of the lines of the intermediate image among the processors. First we distribute regularly and linearly the lines in contiguous blocks of scanlines of even dimension.

In the next step, we determine the necessary data of the volume necessary to execute this computation. It is easy to find to what lines in the intermediate image a slice of the volume contributes to.

Figure 11 represents a volume slice in the sheared object space. The computation of the intermediate image is divided on two processors. The scanlines of pixels in the intermediate image are parallel to the scanlines of voxels of the volume slice. Because of the bilinear filter of the compositing step, the volume scanlines that surround a pixels' scanline contribute to its computation. Consequently, in Figure 11, processor 0 needs the first four lines of the current slice.

In this way, each slice is divided among the processors in blocks of scanlines. Figures 9 and 10 show, for each processor, what number of scanlines (respectively of data) every processor owns for each slice. For instance, Figure 9 shows that processor 0 may have almost no scanline of the last slices while the processor 3 may have almost no scanline of the first slices. Figure 10 shows that central processors 1 and 2 have almost the entire data.

As illustrated by Figure 18, the computational load is similar to the data distribution. Then we have to find an appropriate load-balancing method.

Finally, we outline that the Shear-Warp algorithm is a highly irregular application because of the RLE data structure. Accordingly, in the parallel algorithm, computation and communication

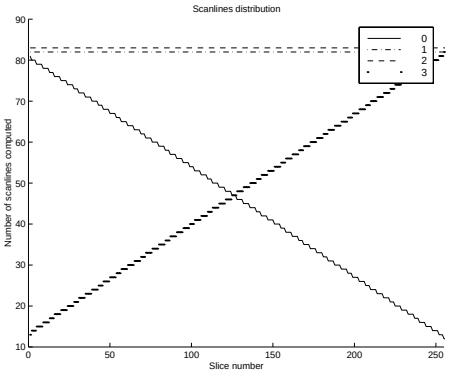


Figure 9: Scanlines distribution.

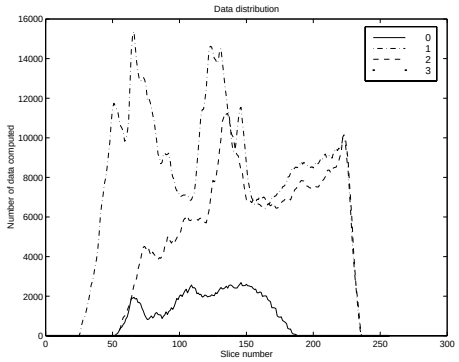


Figure 10: Data distribution.

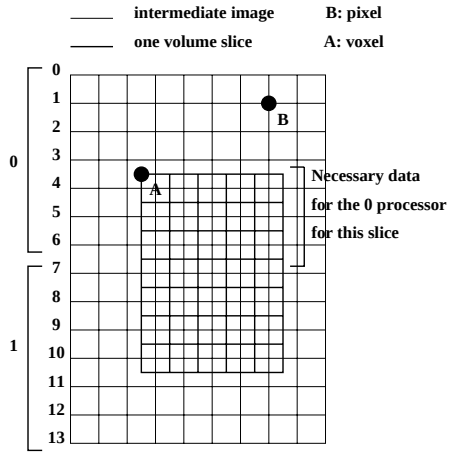


Figure 11: Slice in the sheared object space.

are very irregular as well.

3.2 Dynamic load-balancing

The requirement of an arbitrary rotation of the cube of voxels implies that we should implement a dynamic load-balancing mechanism. As shown in Figure 12, load-balancing strictly depends of the viewpoint. For instance, Figure 12 illustrates 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 angles to the default viewpoint. We can notice that the bigger the rotation is, the more different is the data repartition.

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 [9] to determine the load and to get a good load-balancing accordingly. This algorithm consists in computing a local partial

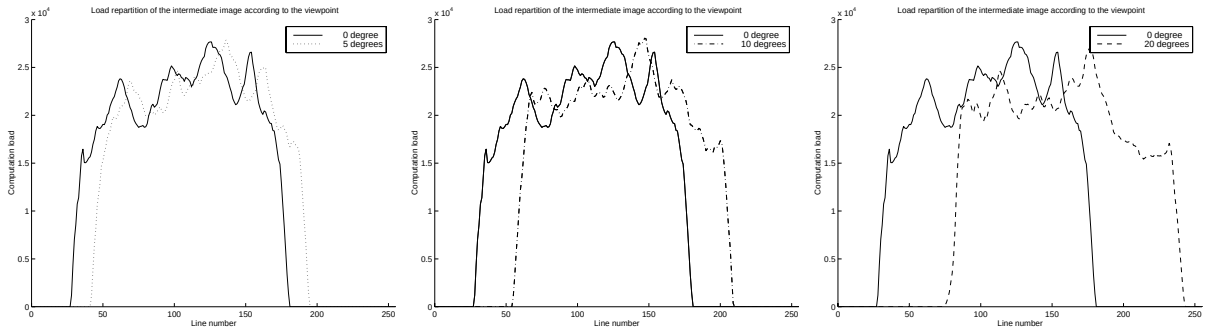


Figure 12: Data repartition in the intermediate image respectively for 5-deg., 10-deg. and 20-deg. rotation angles.

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 broadcast. Each processor adds the arrays received with its own. The resulting array contains the computational load repartition for each line of the intermediate image. Then they can obtain the global load. By linearly distributing the intermediate image, they can balance the load through the processors.

Figures 13 and 14 give the scanlines and data load on four processors with the elastic load-balancing. Compared to Figures 9 and 10, these loads seem more balanced for every processor. Nevertheless the amount of data on every processor for each scanline is still very irregular.

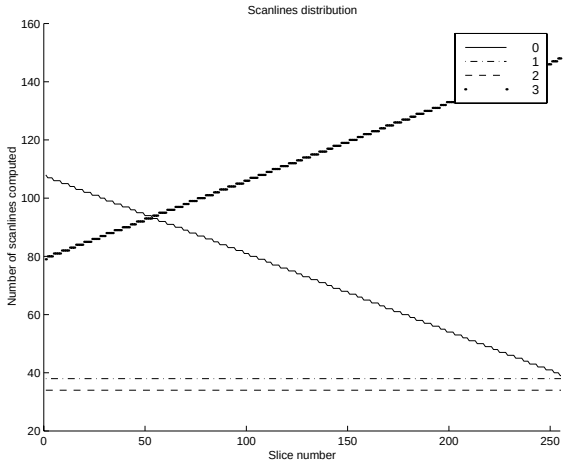


Figure 13: Scanlines distribution.

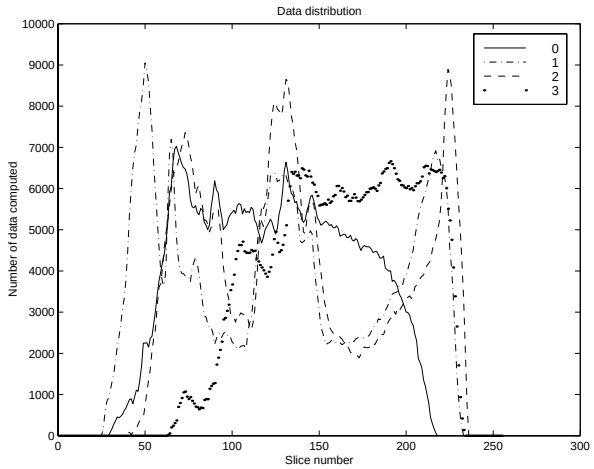


Figure 14: Data distribution.

3.3 Overlapping communications with computations

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 have to deal with communication layers. As a matter of fact, none of our implementations of the MPI standard provide a real asynchronous communication routine.

3.3.1 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
```

Therefore, it is possible to overlap the communication of slice $k+1$ with the composition of slice k. Figure 15 shows an example of communication and computation pattern with 4 processors and 3 slices. Without overlap, for each slice the processor waits for every processor's data before starting the computation. It is represented by 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. We obtain a gain 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. We obtain a gain of B (with $B > A$).

3.3.2 Experimental platform

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

The communication layer BIP (Basic Interface for Parallelism) [10] 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 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[3] 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 16 and 17 illustrate respectively the capability

[3] `http://www.mpi.nd.edu/lam/`

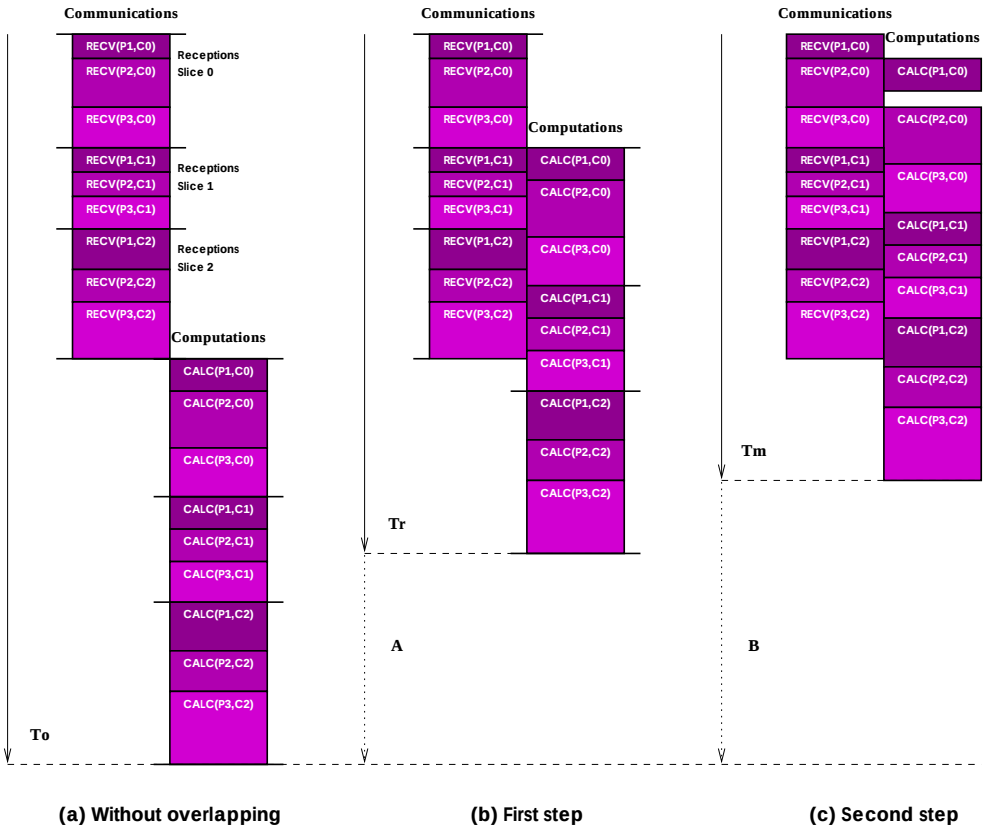


Figure 15: Two possible ways to overlap communications with computations.

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 times for both blocking and non blocking versions are obviously the same. For both MPI and native BIP executions, 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.

3.3.3 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 3.3.1.

4 Results

4.1 Efficient load-balancing

Figures 18 and 19 compare respectively the workload of each processor for an execution with processors respectively in the case of a static allocation and a dynamic redistribution.

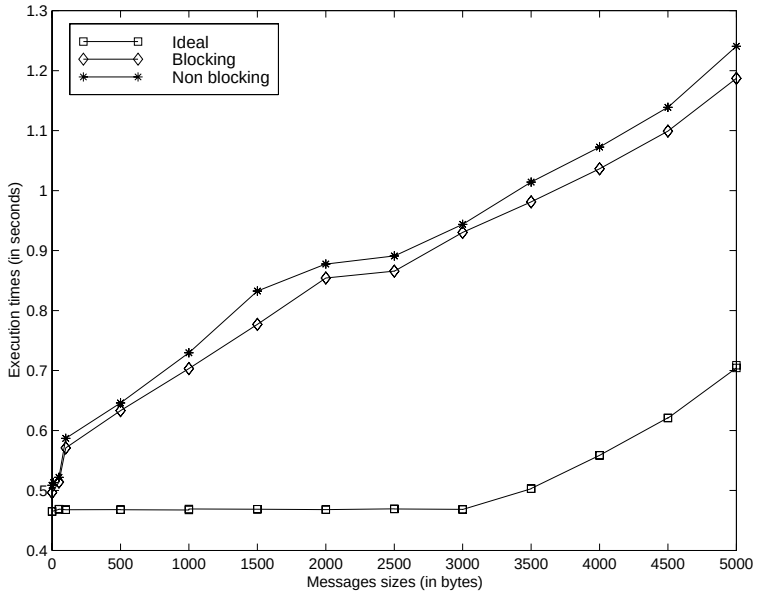


Figure 16: Overlapping communication with MPI.

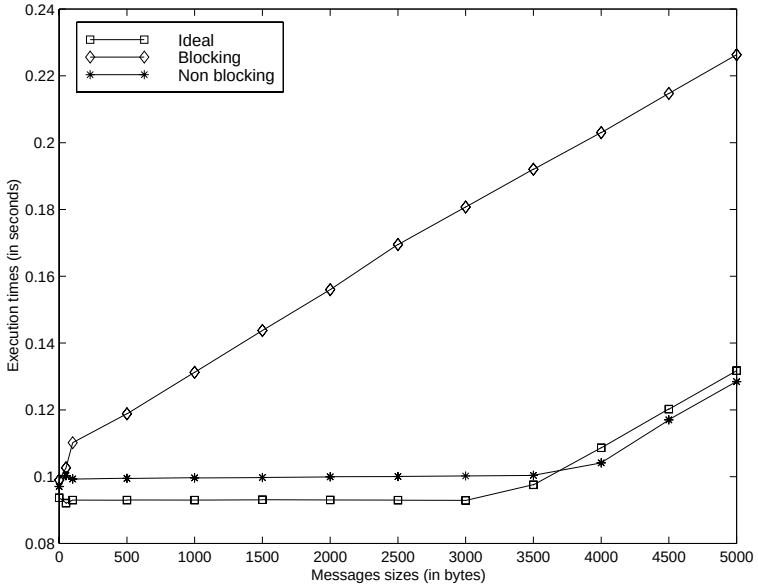


Figure 17: Overlapping communications with respectively native BIP.

Using a static allocation, we distribute the slices with a block-cyclic distribution the lines. Figure 18 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 19 are due to the granularity of the data.

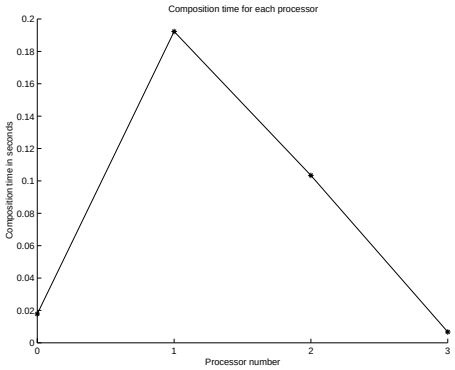


Figure 18: Workload of each processor: static allocation.

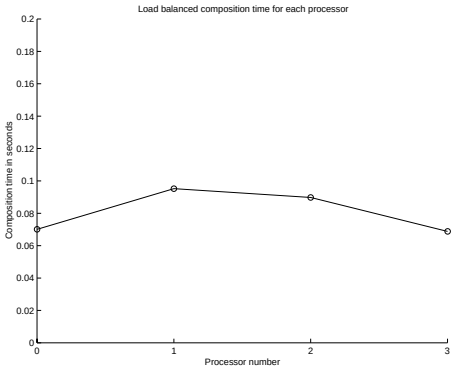


Figure 19: Workload of each processor: dynamic redistribution.

4.2 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 22, is due to data redistribution overhead. Therefore, we 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 thanks to the BIP layer and of course because we could find independent computation and communication in the Shear-Warp algorithm.

Figure 20 represents computation and communication times according to the rotation angle relative to the initial viewpoint for a four-processors execution. We can notice that the computation time is constant. As a matter of fact, whatever the angle is, the global computation volume is the same. In contrast, communication time is strictly increasing according to the viewpoint angle. This curve is quickly linear. We can then reach a total overlap of the communications with the computation at the intersection point of the two curves; here for a relative angle of 17 degrees.

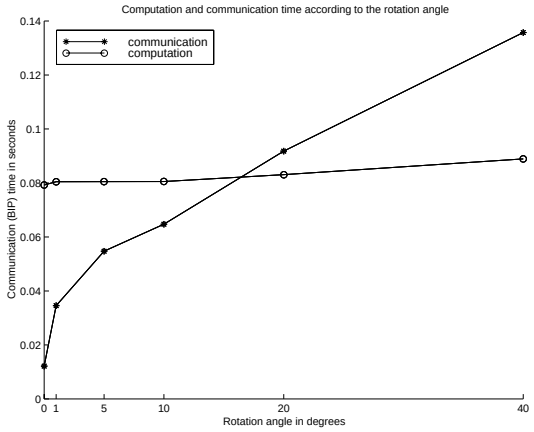


Figure 20: Computation and communication times as a function of the rotation angle.

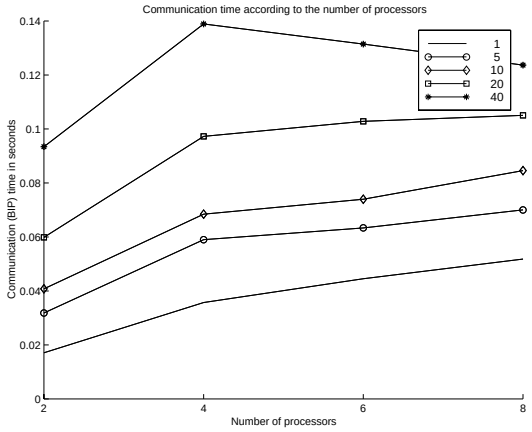


Figure 21: Communication time as a function of the number of processors.

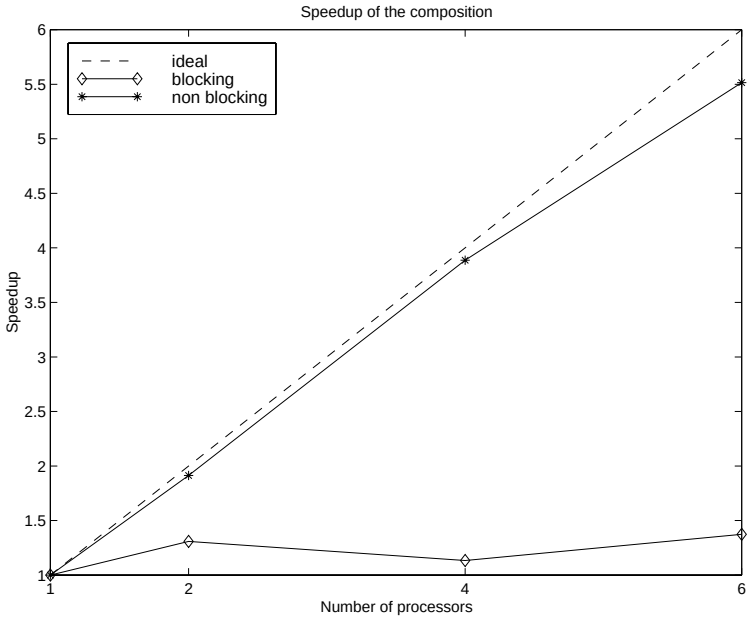


Figure 22: Speedup of the compositing phase.

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.

5 Conclusion and future work

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

In this report, we have presented an high performance and scalable version of the Shear-Warp algorithm implemented on a cluster of PCs. Our parallel approach of the Shear-Warp algorithm improves the interactivity of the application by using an adapted load-balancing algorithm and by overlapping communications. It 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 report 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

[1] 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`.

[2] John Danskin and Pat Hanrahan. Fast Algorithms for Volume Ray Tracing. In *Proceedings of the 1992 Workshop on Volume Rendering*, pages 91–98, 1992. `http://graphics.stanford.EDU/papers/volume/`.

[3] 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. `ftp://ftp.cs.unc.edu/pub/users/sc/ppopp97/45.ps.Z`.

[4] Greg Johnson and Jon Genetti. Medical Diagnosis Using the Cray T3D. In *1995 Spring Proceedings (Cray User Group)*, pages 70–77, 1995.

[5] Arie E. Kaufman. Volume Visualization. *ACM Computing Surveys*, 28(1):165–167, March 1996. `ftp://ftp.cis.ohio-state.edu/pub/yagel/sig97/09_ari.paper.vv.ps.gz`.

[6] 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/`.

[7] 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/`.

[8] 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/`.

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

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

[11] David Sarrut. Imagerie Médicale et Segmentation du Cerveau. Technical report, Laboratoire ERIC, Université Lyon 2, 1997. `http://eric.univ-lyon2.fr/francais/RR.html`.

[12] Lee Alan Westover. SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm. Technical Report TR91-029, Department of Computer Science, University of North Carolina - Chapel Hill, July 1991. `ftp://ftp.cs.unc.edu/pub/publications/techreports/91-029.ps.tar.Z`.