# OPTIMIZATION AND PARALLELIZATION OF A GEOGRAPHICAL STEREO VISION CODE

**Sylvain Contasssot-Vivier and Serge Miguet**
ERIC, University Lumière Lyon 2, Bâtiment L,
5, Avenue Pierre Mendès-France, 69676 BRON Cedex, France
{scontass,miguet}@univ-lyon2.fr

**ABSTRACT**

This paper describes the sequential and parallel versions of a stereo vision algorithm. In this study, we address the particular stereo vision problem of accurate Digital Elevation Models (DEMs) reconstruction from a pair of images of the SPOT satellite. We start from an existing algorithm made by M.Memier, we optimize it while focusing on the cross-correlation problem based on a statistical operator. And we propose a parallel implementation with special care to load balancing.

**keywords :** Stereo Vision, Computational Optimization, Parallelism, Load Balancing.

## 1 INTRODUCTION

The stereo vision problem has been intensively studied in the domain of robotics [Aya91], producing several correlation techniques. In this framework, the depth information which has to be reconstructed is most often sparse, since in most of the cases, the robot just needs some essential depth information to avoid obstacles. On the contrary, in the case of DEMs production, we want to obtain depth maps as dense as possible. Several techniques exist to provide such dense maps but they are often time consuming and not so accurate.

In the scope of our study, the input data consist in two images from SPOT satellite of a particular region taken from different points of view. From these images, we extract the three-dimensional information by finding couples of corresponding points and computing 3D coordinates using camera information. The general way to find couples of corresponding points is to take one image as the reference. The choice of the reference image is arbitrary and is not subject to any loss of generality. Then, for each pixel in this image, we try to find its counterpart in the other image. Hopefully, we can restrict the search domain of counterparts by transforming input images in epipolar geometry. This geometry, based on optical principles, has the very interesting feature to align the corresponding points on same lines of images. Then, the search domain is drastically reduced to at most one image line. Nonetheless, the input data size may be very large especially from satellite imagery which produces $6000 \times 6000$ images, involving important computation times as well as very large memory demand. To solve this problem, we propose then to use the computational power of parallel MIMD machines. Our parallel implementation has been done using PPCM[1] library allowing us to run the program over different parallel machines.

We start from a stereo vision code developed by Michel Memier [Mem91] in the scope of his thesis. His program, based on template matching, is rather accurate but is time and memory consuming. Hence, we propose in a first step, to optimize this sequential code. Then, the second step consists in parallelizing our optimized version. Our aim being the possibility to compute accurate DEMs from entire SPOT images (6000 $\times$ 6000 pixels).

The first part of the article describes the initial algorithm of Memier . In Section 3, we present the optimizations and transformations we made on this initial algorithm. A parallel version is proposed in section 4. Finally, experimental results are given and interpreted in section 5 and allow us to conclude and give several perspectives as future work which may complete this study.

[1] Portable Parallel Communication Module : Environment developed by the parallel image processing group of the Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France

# 2 MEMIER'S ALGORITHM

In this paragraph, we just give a brief description of the algorithm and the reader should refer to [Mem91] for a detailed description. In our study, the left image will be called the reference image of the correlation. Since we focus on the correlation step, input images are assumed to be in epipolar geometry implying that corresponding points are on a same line in the two images.

## 2.1 General algorithm

Memier's algorithm uses an area based correlation. It tries to match regions (surrounding windows of pixels) by computing a similarity coefficient, also known as cross-correlation coefficient (see [FHM+93, Bro92]). For each reference pixel, there is a fixed number of possible candidates in the second image, defined by the geometric constraints of the stereo viewing system. This number of pixels/candidates to scan is called the maximal parallax. In order to save time, Memier uses a two-stage correlation: a first correlation is performed on reduced images generating estimated disparities which are used in a second correlation on full resolution images with a smaller search range. Moreover, to get more coherent results, Memier uses an order constraint over the couples of corresponding points in his first correlation. This leads to a global matching of associated epipolar lines using dynamic programming. After the second correlation, 3D points are reconstructed and the DEM can be sampled. The algorithm is then composed of the following stages:

1. reduce initial images by a ratio $R$ and compute similarity coefficients for the initial number (maximal parallax) of candidates using a square correlation window of size $N_1$,
2. perform global matching of epipolar lines using dynamic programming: this is done by searching a path of minimal cost (in terms of dissimilarity) in the 2D graph obtained by connecting all the points of the two associated epipolar lines,
3. resample disparities to get their estimations at full resolution level using natural cubic splines,
4. rescale disparities and shift each pixel of right image according to its own disparity: this allows the use of a smaller search domain in the second correlation,
5. perform the second correlation on full resolution images with initial left image and modified right image using a smaller range of candidates and a correlation window of size $N_2$. Also, compute final disparities by adding estimated ones and residual ones (obtained in the second correlation),
6. reconstruct 3D points according to the disparities and the geometry of the cameras,
7. sample the 3D points on the grid of the DEM: the DEM is made of a regular grid of points whose altitudes are known, but at the previous stage, we only obtain a set of 3D points non-regularly organized in the 2D plane of top view (XY plane). Hence, a sampling of these 3D points must be done using a given size of cells and a threshold to discard points with too important altitude divergence in relation to the other points inside each cell.

Consequently, the initial code of Memier consists in a group of programs (written in Pascal), each of them implementing one of the steps above and taking on their input, the output of the previous step. To verify the validity of this algorithm, we have compared it to the existing approaches in the domain. It would be too long to develop this bibliographical study here (some good reviews can be found in [Aya91, FHM+93, Zha93]), but our conclusion is that on one hand it can produce good quality results: all the 3D points it generates are ensured to have a high probability to be valid and accurate and this is confirmed by the topological point of view of geologists, whereas most of other techniques generate a set of 3D points, sometimes larger than Memier's one, but which includes more erroneous values. And these kinds of results are far more difficult to exploit for geologists. On the other hand, this algorithm is quite slow, the interpolation function is not well adapted, and it requires very large quantity of memory. Hence, it represents a good starting point for our study but should not be directly parallelized. It is why, our first task has been to optimize the sequential version and try to enhance the results by increasing the number of generated points without loosing validity or accuracy.

# 3 SEQUENTIAL ALGORITHM

As said in the previous section, there are mainly three points which should be enhanced in our available Pascal code. The following paragraphs describe the improvements made on each of these features. First of all, we must point out that whereas Memier's code is written in Pascal, C language has been chosen to implement our version for mainly two reasons: Pascal language does not provide parallel libraries allowing fast development of multi-processors oriented code, and although Pascal language can handle dynamic memory allocation, C language allows more convenient and powerful memory management and internal optimization code.

### 3.1 Computation time

The major drawback of Memier's algorithm is that statistical computations needed for the correlations are made independently which is very time consuming. Considering the size of our input image (typically 2000×2000), the two-stage matching with reduced images stays rather slow. Yet, since the algorithm performs the similarity computations in a regular scan order into epipolar lines (from left to right), and using a fixed window size, we can see in Fig. 1 that there is a common part of computations between two following pixels. Thus, the idea is to decompose the com-

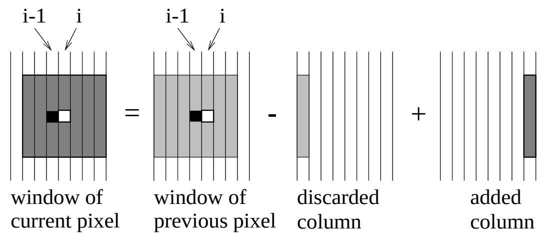


Figure 1: Window updating of one pixel from its predecessor

putations by columns to allow an incremental updating of statistical information of each window. This is possible because the sums of the correlation coefficient can actually been decomposed by sums on columns. This technique has also been used and fully detailed in [FHM+93]. We could also apply the same kind of dependences to deduce statistics for windows of one scanline from its previous line. Nevertheless, this modification would be quite expensive in memory and would represent a far less important gain of time than the previous one since the avoided computations would just concern the first reference point of each line. So, we decided to use only the optimizations inside the lines but not between lines. Hence, let see what is the difference, in terms of complexity, between the initial similarity measurement and our optimized version for one epipolar line: we consider the window size $N \times N$, the search range $M$, and the image width $W$. Computations of variance and covariance are both in $O(N^2)$, so computations for one reference pixel and its $M$ candidates are in $O(M \times N^2)$. Finally, Memier's algorithm is then in $O(W \times M \times N^2)$ on one epipolar line. Concerning our algorithm, it is divided in two parts which are the line initialization step and the process of the other reference points. Initialization step consists in the computation of the first reference point in the line and has the same complexity $O(M \times N^2)$ as previous algorithm since we have to totally compute the variances and covariances of the windows from scratch. Nevertheless, calculation of each following reference point is just in $O(M \times N)$ since we use the incremental updating. This leads to a general complexity of $O(M \times N^2 + W \times M \times N)$ to process an entire line. Hence, we can see that if $W \gg N$ (which is the common case), we almost gain a factor $N$ in relation to the initial code. This analysis also shows that inter-lines incremental computations would just allow to gain $M \times N^2$ operations per line which is negligible.

### 3.2 Quality of results

We propose here to use another interpolation method, allowing the generation of discontinuities in reconstruction of approximated disparities. To achieve such feature, we use a similar approach to Catmull-Rom [CR74]. Hence, our one dimensional function splits each existing data $P_i$ in two new surrounding items $P'_i$, $P'_{i+1}$ according to the following constraints:

$$\frac{P'_i + P'_{i+1}}{2} = P_i \tag{1}$$

$$slope([P'_i P'_{i+1}]) = slope([P_{i-1} P_{i+1}]) \tag{2}$$

Our discontinuity management is based upon the relative distance of the two neighbors $P_{i-1}$ and $P_{i+1}$ from the current item $P_i$. When this distance is too large, $P'_i$ and $P'_{i+1}$ are computed according to the slope of $P_i$ and its closest neighbor in term of values. Since the vertical interpolation would create buffering problems (see paragraph 3.4), we decided to reduce the images only in the horizontal direction. The loss of time implied in the first correlation is partly compensated by the higher speed of our interpolation method.

The effect of these changes on the quality of the results is a more dense disparity map having at least the same accuracy as before since we do not change the matching strategy.

### 3.3 Memory requirements

Memier's program is not efficient considering memory management since arrays are statically allocated, and the entire data set to be processed in a stage is loaded in memory. Of course, we must note that Memier's preoccupation was not to produce an optimized program in a goal of exploitation but just to validate his method. Our principal work to develop efficient memory management has been to adapt buffering to each stage of the algorithm in order to store in memory just needed data for computations. This was possible according to the regular order and independent nature of computations between consecutive epipolar lines. Here are the principal changes made over each stage:

1. the first correlation is made by buffering only the $N_1$ consecutive lines of images needed to make the correlation of one epipolar line. The buffer is cyclically updated to process the following epipolar lines;
2. the graph used in dynamic programming is reduced to just the area actually needed: the diagonal strip of height equal to the maximal parallax;
3. resampling and rescaling of disparities is straightforward for horizontal direction and requires only one row of data;
4. shifting the right image is also straightforward and just requires disparities of one epipolar line at a time;
5. the second correlation is quite similar to the first one and uses a buffer with $N_2$ lines. Nonetheless, this stage also requires a buffer of $\lfloor \frac{N_2}{2} \rfloor + 1$ lines to store the estimated disparities (obtained in stage 3) which have to be added to the residual ones (computed here). Its size comes from the initialization step of the second correlation where the $\lfloor \frac{N_2}{2} \rfloor$ first input lines can not be matched (but just used to process the following lines). Hence, when initializing the correlation window of height $N_2$, only the $\lfloor \frac{N_2}{2} \rfloor + 1$ last lines are relevant to second matching and require estimated disparities;
6. the reconstruction step can be done independently for each point, but several points can be stored in memory at a time;
7. DEM's sampling has not been subject to modifications here.

All these modifications have considerably decreased the memory consuming of Memier's program. The initial code could only be run on machines with about 128 Mb of memory whereas our version uses less than 10 Mb for $1951 \times 1951$ images.

Finally, we obtain a set of fast and small sequential programs corresponding to the different stages of the reconstruction algorithm. Nevertheless, the different programs communicate with each others using files which generally have huge sizes. Moreover, in the context of parallelization, it would be far better to have only one program implementing the entire algorithm. Hence, the final step in the design of our sequential code is to pipeline all the stages one after the other in a single program.

## 3.4 Pipelined code

The pipelining phase of the different programs is a common process which consists to directly link the output of one stage to the input of the following stage. The main constraint here, is to keep as most as possible the buffers made in the previous paragraph. Fig. 2 shows the pipeline's structure and input data required for each stage. The first problem we encounter concerns the vertical interpolation: for each point to vertically resample, we need at least one valid disparity (matched point) above and below this point in its column. But the possibility to have invalid disparities inside the columns imply arbitrary vertical buffer size. To avoid this problem and considering the speed of our algorithm, we decided to reduce initial images only in the horizontal direction (implying just one epipolar line at input of interpolation in Fig.2). Concerning the DEM's sampling, it has not actually been integrated in the pipeline mainly because this operation can be performed several times for a same set of 3D points according to the size of the grid cells and the threshold for local altitude dispersion. Hence, it is not interesting to redo all the matching computations when only changing the sampling parameters of the DEM's grid.

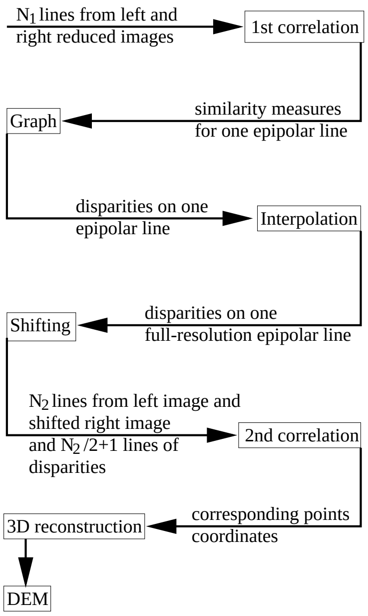


Figure 2: Pipelined version of the general algorithm

# 4 PARALLEL ALGORITHM

In the remainder of the article, we consider MIMD[2] parallel machines as a collection of $P$ processors numbered from 0 to $P-1$, each of them having its own memory and communicating with explicit message passing. Moreover, we consider the indexing of image lines beginning at 0.

Parallelization of our sequential program is rather straightforward due to the relative vertical independence of our computations between lines. Our main problem resides then in the choice of the strategy for work and data repartition. We could choose a brute force strategy with entire data duplication over the processors but this would lose the benefit of our memory optimizations. Hence, we only consider strategies of data distribution over the processors in order to have only needed data for local processings. The first idea which then comes to mind is to equally divide the images in horizontal strips and then assign each strip to a processor. Nevertheless, this equal data repartition may lead to work imbalance since for example, all the lines in the first and last strip will not actually be processed (due to the height of correlation windows). The problem comes from the fact that this strategy does not take into account the work repartition. Hence, the method that seems to be the best suited is to use load balancing to equally distribute work rather than data.

## 4.1 Load balanced repartitions

The aim of this method is to distribute data in order to have an equal work repartition over the processors. To do so, we must define a work unity which will correspond to the smallest task assignable to a processor, and consider the spatial distribution of these items inside the images. Seeing previous remarks on vertical independence, global nature of the matching inside epipolar lines, and their equivalence in terms of matching work, we naturally define epipolar lines as work unity. In the following paragraphs, we describe the different data repartitions acting at the strategic levels of the pipeline.

### 4.1.1 First correlation

We know that the $\lfloor \frac{N_1}{2} \rfloor$ first and last lines of images can not been matched in the first pass. Consequently, the range of actually matched lines is $[\lfloor \frac{N_1}{2} \rfloor \ldots H-1-\lfloor \frac{N_1}{2} \rfloor]$ where $H$ is the height of the images. Then, a first step towards load balancing is to equally divide *this* range into $P$ strips rather than the whole height of images since it represents the actual amount of work. The correlation windows imply to add the $\lfloor \frac{N_1}{2} \rfloor$ surrounding lines on both sides of each strip (when relevant). It is important to see the difference between the work repartition (strips) and the corresponding data repartition (strips with additional lines) needed to perform each strip. Considering Fig. 2, we can see that all the tasks up to the second correlation are optimally load balanced by this new data distribution since epipolar lines are equivalent in terms of CPU consuming for all of these stages. Nonetheless, it does not yield a balanced second correlation stage and another repartition must then be performed.

### 4.1.2 Second correlation

At this stage, we encounter the same problem than in the first correlation, i.e the need of $\lfloor \frac{N_2}{2} \rfloor$ lines before and after the lines to process. Moreover, since this second matching is done over re-adjusted lines of right image, it can just use lines which have been processed in the first pass. Consequently, the same load balancing scheme as before is performed for this stage, but taking into account the new context of work and data repartition. This context is that the number of lines to consider is now $H'$, and $N_1$ is replaced by $N_2$. Also, since estimated disparities are needed in this stage, they have to be redistributed as well as the lines of images. We can notice that the maximal number of usable processors is conditioned by the number of lines to process in this correlation ($H''$) in order to have at least one line per processor. Once this step is achieved, the stereo points can be reconstructed in the 3D space.

### 4.1.3 3D reconstruction

Here, the workload equivalence of the lines is no longer true because of the variable number of valid matches from one epipolar line to another. It is then replaced by the stereo points equivalence. Thus, the load balancing task is to equally redistribute the valid stereo points among the processors. This is done by the help of the commonly used bucketing operation over the points and a multi-distribution communication.

As explained in paragraph 3.4, the DEM's sampling has not been integrated in the sequential pipeline to avoid long recalculations when computing several DEMs from a same input data set. Nevertheless, the relative high speed of our parallel version allows us to optionally include it.

[2] Multiple Instructions Multiple Data

#### 4.1.4 DEM sampling

The sampling of the 3D points into a given grid can be made independently cell by cell. Moreover, the calculation time of each cell is relative to the number of 3D points lying in it, which is variable from one cell to another. The ideal solution to balance this stage would be to distribute the cells in order to have almost the same number of points on each processor. Nonetheless, this would require a global knowledge of all the points and the use of an heuristic implying additional communications and computations. Then, our idea is to use the same method as for matchings, i.e dividing the grid into horizontal strips and distributing the data according to the strips. This again requires bucket sorting of the points according to the axis of the lines and a multi-distribution. This scheme is not perfect but we can expect a quite homogeneous repartition of the points into the grid in most of the cases.

This paragraph achieves the review of all redistribution processes required to obtain a complete load balanced parallel program. We briefly remark that the parallel programming environment we used is the PPCM library. The reader should refer to [CLM92, FMP95] for a detailed description of this library. Its three main features are the portability on several parallel machine (including PVM and MPI), the integration of specific functions for load balancing and the possibility to use virtual topologies as communication network. In our parallel program, we particularly used the multi-distribution functions (for the two last stages) and also a module called ParList (for the correlations) which allows to redistribute linearly ordered data (such as lines of images) with a minimal volume of data motion.

This terminates the description of our parallel algorithm. The following section gives a comparison of both sequential and parallel algorithms according to experimental results.

# 5 RESULTS

## 5.1 Sequential algorithms

Comparison of execution times of the Memier's code and our optimized version allows to have an idea of the computational improvements detailed in paragraph 3.1. The input data consist in two epipolar images of $1951 \times 1951$ pixels, the sizes of the correlation windows are $N_1 = N_2 = 17$, the search ranges are respectively 15 and 3 and the reduction factor for the first correlation is $R = 4$. The times (in seconds) given in Table 1 for each stages of the two versions, have been obtained on a SPARC server 1000. This table confirms the efficiency of our optimizations. We note that the ratio between the times for the first correlation is near $\frac{N_1}{R}$ and those for the second correlation is closed to $N_2$. These results can be easily deduced from our complexity comparison in paragraph 3.1. Also, the initialization step required by our pipeline to compute the first lines of the buffers is quite inexpensive. The 3D reconstruction is larger in our version because the number of matched points is larger. Now, let us analyze the parallel running time of this algorithm.

| Stages \ code | Memier | Pipeline |
|---|---|---|
| initialization | / | 5 |
| 1st correlation | 3160 | 860 |
| graph | 151 | 35 |
| interpolation | 514 | 7 |
| rescaling and shifting | 38 | 7 |
| 2nd correlation | 14231 | 969 |
| 3D reconstruction | 195 | 992 |
| **TOTAL** | **18289** | **2879** |

Table 1: Times (in seconds) of the different stages of the sequential versions

## 5.2 Parallel results

In order to compare sequential and parallel versions, we use input data with the same size as in the previous paragraph. The parallel experiments have been done on a Cray T3E machine which is composed of 256 DEC-$\alpha$ EV5.6 at 375 Mhz with 128 Mb, and using a 3D-Torus interconnection network with 100 Mbits/s bandwidth per unidirectional link.

Because of memory problems encountered on one processor with the sampling of the DEM, we can not use the times on one processor as the reference for the speedup analysis of the complete process. Then, we compute relative speedups by taking as reference the times on two processors. Since parallel calculations finishing at the 3D reconstruction step present valid performances on one processor, we have compared real speedups to relative ones (deduced from the 2-processors execution time) and we found very similar results. We can then reasonably think that relative speedups are also valid for computations including DEM production.

One of the most important feature of a parallel program is its load balance since it directly influences the performances. Moreover, we would like to verify the efficiency of our redistributions made in section 4.1. The histogram in Fig. 3 shows the computation times over the processors for a configuration of 16 elements. The very high similarity of these times confirms the efficiency of our load balancing scheme. Another important feature of

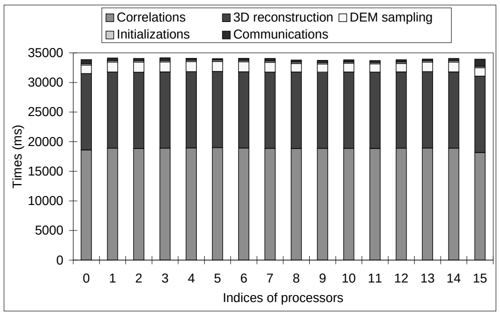


Figure 3: Computation times over the processors for a configuration of 16 elements.

a parallel program is its scalability, i.e its ability to adapt itself to efficiently run on different number of processors. This ability is commonly measured by the speed up which is shown in Fig. 4 for the two executions including or not the DEM sampling step, in function of the number of processors. We can observe that our parallel algorithm is effec-

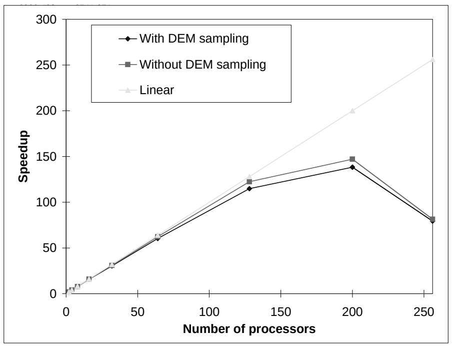


Figure 4: speedup in function of the number of processors for computations including or not DEM sampling.

tively scalable up to about 128 processors, but the curves drastically bend after 200 processors. This is due to the increasing influence of the overheads of parallelism when the size of local input data becomes small. When this is the case, we obtain on each processor less useful work to do whereas the load balancing computations stay the same and the redistributions may even increase. Effectively, since the strips of data are thinner, at each redistribution, the new data assigned to a given processor have a smaller probability to already been on this processor and this implies an increase of the volume of data communicated between processors. Moreover, bottlenecks are more likely to appear because some of our data redistribution functions used in the algorithm (ParList module) are designed to work on a linear array (or eventually a ring) whereas the actual architecture of the Cray T3E is a 3D Torus. This difference between the logical and real interconnection networks implies unexpected collisions between messages. We could solve this problem by re-numbering the processors in a more suitable way (by mapping a linear array on a 3D torus) but unfortunately, we do not have any control and information over the physical numbering of the processors. Also, we could have used another parallel machine with a more suitable network. Nonetheless, as can be seen in Fig. 4, the problems of the overheads appear after 128 processors and we did not have access to another machine having as many processors. Finally, we give in Fig. 5, the repartition of the computation times in function of the number of processors. It can be seen that the proportion of the overheads mainly due to the communications (the *communications* in Fig. 5 also include the computations of the data partitions and the construction of the messages) dramatically increases with the number of processors. Although this implies a loss of parallel efficiency, we could expect to have better results for larger images since large size of input data would require more processors to reach small number of lines in local memory thus putting off the curve bending. Finally, to give an idea of the ab-

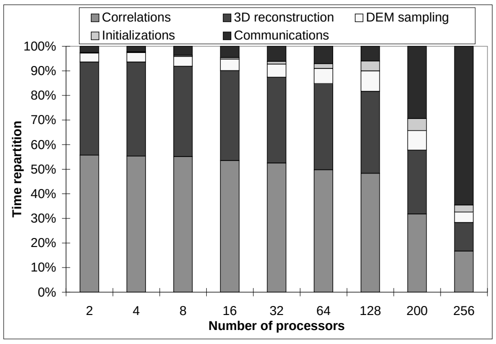


Figure 5: Computation times repartition in function of the number of processors.

solute performances of our parallel program, total and separated execution times are given in function of the number of processors in Fig. 6 using a logarithmic scale. This figure also points out the parts which are not scalable in the algorithm. Effectively, one can see that the computations relative to correlations, 3D reconstruction and DEM are actually scalable since they regularly decrease when the number of processors increases whereas the initializations and especially the communications are definitely not scalable and even increase beyond 128 processors. It can also be observed the direct effect of these parts on the total execution time. Nevertheless, this figure also indicates that the algorithm reaches good absolute perfor-

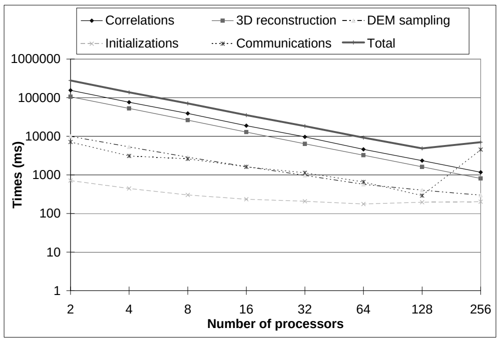


Figure 6: Execution times in function of the number of processors.

mances since our best times are under 7 seconds for complete processing of $1951 \times 1951$ images.

# 6 CONCLUSION

Two algorithms for DEMs reconstruction using stereo satellite images have been described. The first one is a sequential optimized version of an existing algorithm. The second one is a parallel version of this optimized sequential algorithm. The initial algorithm compared to existing approaches has a quite good quality, but it is very slow and memory consuming. Different ways to improve the density and accuracy of the results have been discussed. Particularly the interpolation functions to be used for the disparities. Nevertheless, other improvements not seen here are possible. The most interesting would be to take into account the vertical coherence between consecutive lines to increase the robustness of the matches. But this technique would imply a complete reorganization of our pipeline since the buffers would be drastically changed. This study has pointed out that sequential optimizations and pipelining are really efficient and indicated to reach reasonable execution times. Concerning the parallel algorithm, the different overheads of our algorithm, mainly composed of the communications, have been exhibited and analyzed. It has been pointed out these overheads could be reduced when processing larger images and/or when using a more adapted interconnection network. We have finally shown that parallelism is very well suited and should be recommended to this kind of computations since it allows us to expect very fast processings and the ability to manage very large images like the $6000 \times 6000$ provided by SPOT satellite.

Concerning our future works, we would like to provide a theoretical execution model of the parallel algorithm. This would allow us to find the optimal number of processors to use for a given size of input data and correlation parameters. Also, now that we have efficient algorithms for satellite stereo vision, we would like to extend them to aerial imagery which represents an even larger field of applications. The modifications to do will concern the epipolar alignment step (not discussed here) and the 3D reconstruction step which directly depends on the camera model used.

# References

[Aya91] N. Ayache. *Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception.* MIT Press, Cambridge, MA, 1991.

[Bro92] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, December 1992.

[CLM92] Henri-Pierre Charles, Jian-Jin Li, and Serge Miguet. A portable parallel toolkit for 3D image processing. *The European Transactions on Telecommunications*, 3(6):31–42, 1992.

[CR74] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, pages 317–326, 1974.

[FHM+93] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real time correlation-based stereo: algorithm, implementatinos and applications. Technical Report 2013, Institut National De Recherche en Informatique et en Automatique (INRIA), 06902 Sophia Antipolis, France, 1993.

[FMP95] Fabien Feschet, Serge Miguet, and Laurent Perroton. *Parallélisme et Applications Irrégulières*, chapter 7 : ParList : une structure de donnée parallèle pour l'équilibrage des charges, pages 177–201. HERMES, 1995.

[Mem91] M. Memier. *Stéréophotogrammétrie numérique : calcul de MNT par corrélation automatique d'images SPOT.* PhD thesis, Université Joseph Fourier Grenoble I, 1991.

[Zha93] Z. Zhang. Le problème de la mise en correspondance : L'état de l'art. Technical Report 2146, Institut National De Recherche en Informatique et en Automatique (INRIA), 06902 Sophia Antipolis, France, 1993.