Parallel Processing Letters

GRID-ENABLED NON-RIGID REGISTRATION OF MEDICAL IMAGES

RADU STEFANESCU, XAVIER PENNEC and NICHOLAS AYACHE*
INRIA Sophia-Antipolis, Epidaure Project, 2004 Route des Lucioles, BP 93
06902 Sophia-Antipolis, France

Received 30 May 2003
Revised 27 March 2004
Communicated by A. Apostolico

Preprint. To appear in the special HealthGrid 2003 issue of Parallel Processing Letters.

ABSTRACT

Over recent years, non-rigid registration has become a major issue in medical imaging. It consists in recovering a dense point-to-point correspondence field between two images, and usually takes a long time. This is in contrast to the needs of a clinical environment, where usability and speed are major constraints, leading to the necessity of reducing the computation time from slightly less than an hour to just a few minutes. As financial pressure makes it hard for healthcare organizations to invest in expensive high-performance computing (HPC) solutions, cluster computing proves to be a convenient solution to our computation needs, offering a large processing power at a low cost. Among the fast and efficient non-rigid registration methods, we chose the demons algorithm for its simplicity and good performances. The parallel implementation decomposes the correspondence field into spatial blocks, each block being assigned to a node of the cluster. We obtained an acceleration of 11 by using 15 2GHz PC's connected through a 1GB/s Ethernet network and reduced the computation time from 40min to 3min30. In order to further optimize the costs and the maintenance load, we investigate in the second part the transparent use of shared computing resources, either through a graphic client or a Web one.

Keywords: cluster computing, grid computing, medical imaging, registration.

1. Introduction

Registration is becoming an essential part of medical image processing algorithms. The purpose of registration algorithms is to recover a geometric transformation between medical images. Depending on the complexity of the transformation, several classes of registration can be distinguished. For example, *rigid registration* recovers the rotations and the translations of the objects in the images. *Non-rigid registration* recovers deformations.

*{Radu.Stefanescu,Xavier.Pennec,Nicholas.Ayache}@sophia.inria.fr

Applications can be found in many domains of medical imaging. For instance, the follow-up of a pathology may be realized using medical images acquired at different moments in time. The physician has to compare the images, detect and interpret the changes. The operation is non-trivial when the images are in two dimensions, and even more difficult in three dimensions. Furthermore, the visual examination produces qualitative, but not quantitative results, leaving the medical decision to a subjective evaluation. After a rigid registration step aimed at recovering the different position and orientation of the patient in the acquisition device, non-rigid registration allows the identification of the more subtle deformations that occurred in the tissues between the two time steps. An additional advantage of non-rigid registration over visual comparison is that it gives quantitative and hence more objective results. For this application, the computation time has to be small with respect to the clinical examination, typically one to two minutes.

Another domain of application for registration is brain Image-Guided Surgery (IGS)[1]. Since the brain contains very small but vital structures, accuracy is highly important. Therefore, the planning of the operation is carefully prepared on pre-operative images (acquired before the operation), and it takes hours to perform. The surgeon selects the safest trajectory towards the target, while avoiding vital locations inside the brain. At the beginning of the operation, the pre-operative image is rigidly registered to the patient's position and orientation. This allows the surgeon to precisely locate the entry point and the trajectory as previously planned. However, once the skull is open, some of the cerebro-spinal fluid (CSF) that normally surrounds the brain leaks out. Therefore, the soft brain tends to deform (brain shift). The changes in the brain shape should induce modifications of the pre-operative planning, since the locations of the operation target and the high risk locations have changed. By registering per-operative images (acquired during the surgical procedure) with the pre-operative ones that where used for planning the operation, the surgeon can recover the deformations that occurred due to the opening of the skull. These deformations are subsequently applied to the initial planning, in order to update it [2,3]. Since the registration is performed during the operation, computation times smaller than one minute are required.

Non-rigid registration is a computationally intensive problem. In order to find the solution, an iterative optimization process is necessary, and dealing with large deformations usually requires a large number of iterations. A common approach to achieve shorter computation times is to sacrifice the performance by either lowering the resolution of the images or reducing the number of iterations. Our choice was to implement the algorithm on a parallel computer, which enabled us to keep the performances intact, while drastically lowering the computation times. This leads us to the second issue: Funding is generally unavailable in healthcare organization for purchasing and, more importantly, maintaining expensive high-performance computing hardware, such as the shared memory computer used in [4]. An obvious alternative is therefore a cluster of networked personal computers, as proposed in [5]. Besides offering large computation power at a low cost, a cluster of workstations has the advantage of versatility: it allows the use of its nodes as individual workstations in regular day to day use. Furthermore, a cluster of bi-processor PC's allows the simultaneous use of the cluster for parallel jobs and individual use, and

such an environment will be present in many laboratories and clinical environments in the near future.

Over time, several non-rigid registration algorithms have been proposed. In 1981, Broit [6] used the linear correlation as a measure of similarity between the two images to match. Later, Bajcsy [7] differentiated this criterion and used a fixed fraction of its gradient as an external force to interact with a linear elasticity model. Thirion [8] proposed to consider non rigid registration as a diffusion process. He introduced in the images entities (demons) that push according to local characteristics. The forces he proposed were inspired from the optical flow equations. Recently, Cachier [9,10] proposed an algorithm which replaces the optical flow equation from [8] with a gradient descent on an energy function combining a similarity term between the images and a smoothness constraint. The transformation is represented by a dense correspondence field, that describes the mapping of each point of the image. Other algorithms use sparsely-controlled transformations: Rexilius [11] use a finite element bio-mechanical model in order to describe the deformation. Rueckert [12] uses the mutual information in order to find a deformation described using B-splines. These algorithms suffer from the fact that they use sparsely-controlled transformations: they are generally unable to retrieve fine deformations.

We chose Cachier's algorithm [9] due to its combination of precision, robustness and relatively low computation time (40 minutes on three-dimensional images of size $256 \times 256 \times 120$). Furthermore, the rather regular structure of the algorithm makes it a good candidate for parallelization. We begin this paper by briefly describing the demons algorithm in Section 2. In Section 3, we establish the parallel decomposition and the parallel algorithm and we finish by addressing the more delicate problem of the Gaussian smoothing of images. Section 4 is dedicated to the presentation of experimental results, while in Section 5 we discuss several issues raised by the implementation of our system in a grid environment.

2. The sequential algorithm

The goal of the non-rigid registration is to estimate for each point p in the *source image* J a corresponding or transformed point $T(p)$ in the *destination image* I. Let Ω_I and Ω_J be the subsets of $\mathbb{R}^3$ containing the coordinates of the points in I and J (we are considering three-dimensional images), then T is a mapping from Ω_J to Ω_I. Throughout this paper, each correspondence $T(p)$ will be described by its associated *displacement* $U(p) = T(p) - p$. Notice that $U(p) = [U_1(p), U_2(p), U_3(p)]$ is a three component vector field. An important aspect of the algorithm is the definition of how well two points match.

2.1. A similarity-based estimation of correspondences

We evaluate the degree of correspondence between a point p of the image J and a point q of the image I by measuring the squared distance $[J(p) - I(q)]^2$ between the image intensities of the two points. The larger this difference is, the more the images are different at these points. Thus, to evaluate the quality of the transformation T at point p, we simply need to compute $[I(p) - J(T(p))]^2$. By

integration over the image space, we obtain the *Sum of Squared Distances (SSD)* similarity criterion between the two images:

$$SSD(I, J, U) = \sum_p [I(p) - J(p + U(p))]^2$$

This criterion is optimized with respect to U using a gradient descent. At each step, a *correction field* CF proportional to the gradient of criterion is computed:

$$CF(p) = -\Delta t \cdot \nabla SSD(p) = -\Delta t \cdot [J(p + U(p)) - I(p)] \cdot \nabla J(p + U(p))$$

where Δt is the time step.

A simple version of the algorithm would iteratively compute and add the correction field to update the current estimation of the displacement. However, there is inevitably some noise in the image acquisition process that degrades the estimated transformation. Furthermore, the transformation should be continuous in order to have a geometric meaning.

The solution is to impose a smoothness constraint at each optimization step on both the correction and displacement fields. Since the correction field can be interpreted as a velocity, its smoothing amounts to a kind of fluid constraint, whereas smoothing the displacement field enforces an elastic behavior. An additional way to reduce the noise on the correction field is to smooth the image while computing the gradient ∇J. In order to smooth an image or a three-dimensional field, a simple, yet efficient solution is to convolve component-wise with a Gaussian. In practice, the gradient is computed by convolution with the derivatives of the Gaussian.

This yields a four-step iterative algorithm that successively estimates, regularizes and adds the correction field to the current estimate of the displacements, and finally regularizes this estimation (Algorithm 1). In this alternated optimization, an identity transformation $T(p) = p$ ($U(p) = 0$) is used as initialization.

Algorithm 1 Registration algorithm with small deformations.

1: Estimate a correction CF of the current displacement field U
2: Smooth CF
3: Add $U = U + CF$
4: Smooth U and go to step 1 until convergence

2.2. A multi-resolution approach

The previous algorithm described is local: it can only recover small deformations with respect to the standard deviation of the Gaussian applied to J before computing its gradient. Increasing the standard deviation enables the algorithm to recover larger deformations, but it lowers its accuracy.

In order to address this problem, a multi-resolution scheme is used, as shown in Figure 1. The algorithm begins by registering two low resolution versions of the images I_2 and J_2, obtained by reducing the resolution of the grid on which image intensities are sampled (subsampling process). On these subsampled images, one can retrieved only the low frequency deformations, but this can be done efficiently: local details of the images, that would attract the algorithm into local minima, are eliminated. By progressively increasing the resolution, one recovers finer and finer details (Algorithm 2). We call an *image pyramid* the data structure containing versions of an image at different resolution levels. Figure 1 presents two such pyramids.

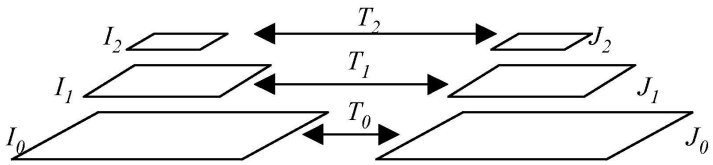


Fig. 1. Multi-resolution approach: The registration is first done using the lowest resolution versions I_2 and J_2 of I and J. After convergence, the resulting transformation U_2 is interpolated onto the higher resolution grid (over-sampling) and serves as an initial estimate for the registration between I_1 and J_1 at the next level. This recursive process is performed up to the resolution of the original images (I_0 and J_0), yielding the final solution U_0.

Algorithm 2 Sequential multi-resolution algorithm.

```
let n be the number of resolution levels.
let I_0, ..., I_{n-1} be the image pyramid built by subsampling I
//I_0 = I
let J_0, ..., J_{n-1} be the image pyramid built by subsampling J
//J_0 = J
let U_{n-1} be the identity displacement field.
for i = n - 1 down to 0:
    Use Algorithm 1 in order to compute the
      displacement field U_i that deforms J_i into I_i
    if i > 0:
        oversample U_i to the size of I_{i-1} and J_{i-1}
        let U_{i-1} = U_i
```

2.3. Gaussian filtering

For each resolution level, the gradient descent algorithm 1 is applied. The computation of the gradient ∇J can be done once for all, but we need to smooth the

correction and the displacement fields at each iteration. For efficiency reasons, this Gaussian filter is computed using a recursive implementation [13] rather than by convolution. The main advantage of this algorithm is that, unlike the convolution, the computation time does not depend on the standard deviation of the Gaussian. This time depends linearly on the image size. In one dimension, the algorithm pre-computes the coefficients α_j, β_j, γ_j and η_j of two fourth order recursive filters that corresponds to the required Gaussian's standard deviation. Then, filter (1) is applied forwards and filter (2) is applied backwards.

$$out_i^+ = \sum_{j=0}^{j<5} \alpha_j in_{i-j} + \sum_{j=1}^{j<5} \beta_j out_{i-j}^+ \quad (1)$$

$$out_i^- = \sum_{j=0}^{j<5} \gamma_j in_{i+j} + \sum_{j=1}^{j<5} \eta_j out_{i+j}^- \quad (2)$$

The filtered version of the signal is obtained by taking the sum $out^+ + out^-$ of the signal filtered by (1) and (2).

In more dimensions, one takes advantage of the separability property of the Gaussian, and successively filters the image along all directions. When applying the filter to some image in a certain direction, the image is decomposed into lines along that direction. Throughout this paper we will call the lines along the recursive filtering direction *scanlines*. Each scanline is considered as an one-dimensional signal that is filtered independently of the others using the forwards and backwards scheme (Algorithm 3).

Algorithm 3 Recursive Gaussian filtering.

compute the coefficients α_j, β_j, γ_j and η_j depending on σ
for each direction $d \in \{x, y, z\}$:
    for each scanline l along d:
        // filter forwards
        for each point i in l in *increasing* order:
            //Eq.(1)
            $out_i^+ \leftarrow \sum_{j=0}^{j<5} \alpha_j l_{i-j} + \sum_{j=1}^{j<5} \beta_j out_{i-j}^+$
        // filter backwards
        for each point i in l in *decreasing* order:
            //Eq.(2)
            $out_i^- \leftarrow \sum_{j=0}^{j<5} \gamma_j l_{i+j} + \sum_{j=1}^{j<5} \eta_j out_{i+j}^-$
        // write the result back into the line l
        for each point i in l:
            $l_i \leftarrow out_i^+ + out_i^-$

3. The parallel algorithm

The demons algorithm has a regular structure with three main algorithmic components:

- the subsampling of the images and the oversampling of the displacement field for the multi-resolution approach;
- the computation of the correction field;
- the regularization of the correction and displacement fields.

In each algorithmic component, each point of the image is processed in a similar manner. This makes the algorithm a good candidate for a parallelization using a data decomposition, rather then a task decomposition.

3.1. The parallel block decomposition

Since all the three operations mentioned above are performed on the displacement field, we chose to decompose this into *parallel slices perpendicular to one axis only* (say x to simplify explanation), each slice being assigned to one processor (Figure 2). The correction field is decomposed accordingly. We shall see below that, by adopting this block decomposition, each of the three algorithmic components can be parallelized efficiently, avoiding time-costly data redistributions. One could think to generalize the decomposition along the two other axes. However, our experience shows that this leads to slower results, and a much more complex code.

The one-dimensional block decomposition has also an advantage in a heterogeneous environment. By tuning the width of the slice assigned to each processor, one can provide a static load balancing. Thus, machines with different computation powers can be simultaneously used.

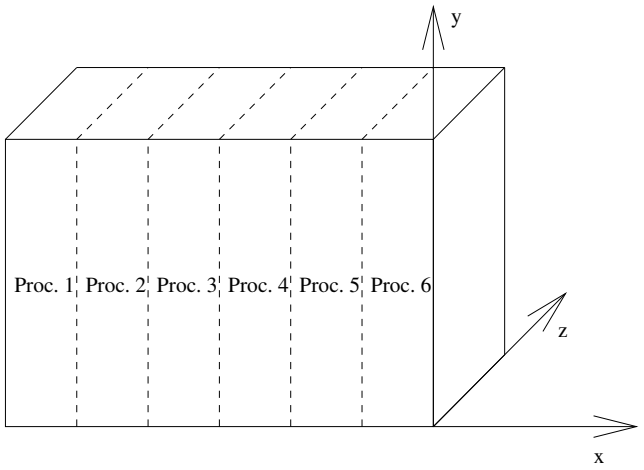


Fig. 2. The data decomposition of the displacement field with 6 equivalent processors: parallel slices perpendicular on the x axis.

3.2. The parallel multi-resolution algorithm

In this block decomposition framework, the images themselves are not distributed since they need to be known entirely by each node (see Section 3.3). This means that the preliminary part of the multi-resolution framework (the subsampling of the images) has to be performed sequentially. During the multi-resolution estimation itself, the displacement field has to be oversampled from one level to the next. As it is distributed among the processors, its oversampling has to be done in parallel. Since we use a tri-linear interpolation, processors only have to communicate to their neighboring nodes the values of the displacement at the border of their domain.

3.3. Estimation of the correction field

The second algorithmic component, the estimation of correction field at each point p, is the simplest operation to parallelize: we only need to know the value of the displacement field $U(p)$, the intensity of the destination image $I(p)$, and the intensity and the gradient of the source image at the point $p + U(p)$. Since images I and J are constant during the execution of the algorithm, each processor can memorize them entirely. The estimation can be performed by each processor

independently of the others, without any need for communication. Notice that even if p is within a given block of the decomposition, the displacement $U(p)$ can be arbitrarily large. Thus, each process has to access the whole image J and its gradient.

3.4. Send-borders: a parallel Gaussian filtering algorithm for small variances

The last algorithmic component to parallelize is the component-wise smoothing of the correction and displacement fields. We present below the case of scalar image, but for a vector field the operation would have to be repeated for each of its components. The Gaussian filter is separable, so convolving a three dimensional image with an isotropic Gaussian is equivalent to successively convolving the image with an one-dimensional Gaussian along each axis. By adopting a block decomposition *along one axis only*, the filtering along two directions can be done within each block without any communication. For filtering along the decomposition axis, one may benefit from the exponential decay of the Gaussian: a good approximation is to consider the Gaussian as null outside its $[\mu - 3\sigma, \mu + 3\sigma]$ interval (where we denoted with μ the Gaussian's mean and with σ its standard deviation). Therefore, when convolving a one-dimensional signal with a Gaussian, we can consider that the value of the filtered signal in some point p depends only of the values of the points of the initial signal within $p \pm 3\sigma$. This leads to the following simple algorithm (Figure 3):

1. Each process sends its borders of width 3σ (in gray in Figure 3) to its neighbors.
2. Each process receives the neighbors' borders (in dashed grey) and adds them to its own domain, obtaining an extended domain.
3. Each node recursively filters its own extended domain and then throws away the received borders, in order to obtain a domain the size of the initial one.

For efficiency reasons, processes use the sequential recursive filtering algorithm inside their own domains.

This algorithm has two drawbacks: First, the produced results are not rigorously correct since the value of a Gaussian is not perfectly null outside the 3σ interval. Using larger borders will make the parallel algorithm less efficient. Second, each process has to apply the filter to a domain that is larger than its own. And finally, the amount of data sent through the network is proportional to the filter's standard deviation.

3.5. Pipeline: A truly recursive Gaussian filtering

An alternative is to directly parallelize the 4th order recursive implementation of the Gaussian proposed by Deriche [13], as follows. Let us consider the lines of the image along the block decomposition direction: they can be filtered independently

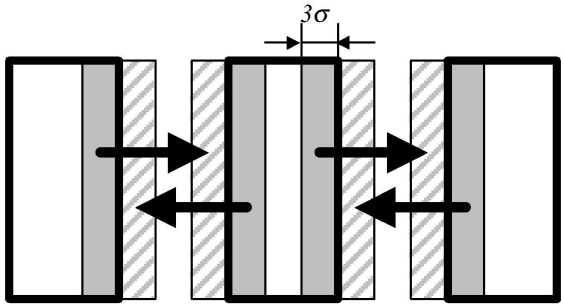


Fig. 3. The send-borders algorithm: Each process sends its 3σ-wide borders to its neighbors and then filters the enlarged domain.

of each other. Due to the recursive nature of the filter, computing the value of one point depends on the filtered version of the previous point when filtering forwards, and of the following point when filtering backwards. This means that the filtering of one single line cannot be done in parallel. However, different processors can deal with their parts of different lines simultaneously (Algorithm 4, Figure 4): At step 1, the left process begins processing its part of the first scanline. Meanwhile, processors 2 and 3 wait. Once processor 1 finished, it can pass on to processor 2 the contents of the 4 points (since the order of the filter is 4) that processor 2 needs in order to process its first point of its part of the first scanline. Process 1 filters its part of the second scanline while process 2 filters its part of the first scanline and process 3 does nothing. At the end of this step, process 1 passes the last 4 points of its part of the second scanline on to process 2, while the latter one sends the last 4 points of its part of the first scanline to the third process. This way, all the processes work simultaneously without filtering one scanline in parallel. The "process pipeline" however takes a number of steps equal to the number of processes before working at its full capacity. The full acceleration is achieved if the number of lines is much larger then the number of processes, which is usually true in a cluster of workstations.

3.6. Comparative analysis of the two parallel filtering algorithms

Two methods to achieve parallel recursive Gaussian filtering were proposed. We now quantify the algorithmic complexity of each of them: computational complexity, network usage and maximal acceleration (when the communication time is null). We assume below that we are filtering an image of size $N_x \times N_y \times N_z$ using M processors. The filter has a standard deviation σ. We assume that computing the recursive filter (Equations (1) and (2)) for each point is done in a time t. This time is the same for the sequential method and the two parallel algorithms.

Algorithm 4 The pipeline parallel Gaussian recursive filter

```
for each direction d:
    /* filter forwards */
    for each scanline l along d:
        if not the first processor along d:
            receive from the preceding processor
            along d its already-filtered last
              4 points
        filter forwards the line l
        if not the last processor along d:
            send the last 4 points to the
            succeeding processor along d
    /* filter backwards */
    for each scanline l along d:
        if not the last processor along d:
            receive from the succeeding processor
            along d its already-filtered first
              4 points
        filter backwards the line l
        if not the first processor along d:
            send the first 4 points of the line
            to the preceding processor along d
```

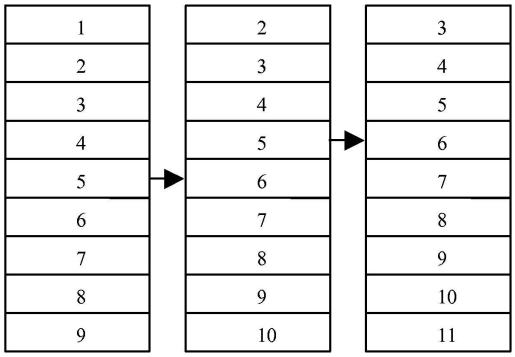


Fig. 4. The pipeline parallel filtering of a 2D image of 10 lines with 3 processors. Inside each line, the step at which it is processed is given. At the end of step 5, processor 1 has just finished filtering its part of the fifth line, and is sending the last 4 points to processor 2. Meanwhile, processor 2 has just finished filtering its part of the fourth line and is sending its last 4 points to processor 3 who has just finished filtering its part of the third scanline. At step 6, the three processors will filter their parts of lines 6, 5 and 4.

Computational complexity In the first method, each process filters its own domain plus the margins that the neighboring nodes sent. So the computation time is $N_y N_z \cdot \left(\frac{N_x}{M} + 6\sigma\right) \cdot t$. The second method does not filter borders, but there is a pipeline filling and emptying penalty. Therefore the computation time in this case is $\frac{N_x N_y N_z}{M} \cdot t + (M-1)\frac{N_x}{M} t$.

Network usage Another important quantifier of the algorithmic efficiency is the total amount of data sent through the network. In the first algorithm, each processor sends 3σ-wide borders to its neighbor, so the total amount of data sent by each processor is $6N_y N_z \sigma$. As we saw before, the amount of data sent by the second algorithm does not depend on σ, each processor sending only the last 4 points (original data and filter result). The total amount of data for this algorithm is $16N_y N_z$.

Finally, for the first algorithm each processor sends only two messages containing the two borders. For the second algorithm, the number of messages is larger $(2N_y N_z)$, which can make the algorithm inefficient. However, a trade-off between the number of messages to send and the maximum parallel acceleration can be made as follows: Rather than sending the last four points of a single line at a time, one can send the last four points of several lines in a single message at the cost of an increase of the time necessary to fill the pipeline. If we denote this number of lines with L, the number of messages for the second algorithm becomes $\frac{2N_y N_z}{L}$ and the computation time becomes $\frac{N_x N_y N_z}{M} \cdot t + (M-1)L\frac{N_x}{M} t$.

Maximal acceleration Even if we consider a null communication time, the acceleration is not linear in either of the cases. For the first algorithm, one must take into account the additional time needed to filter the two received borders. Therefore, the acceleration with M processors is:

$$A_1(M) = \frac{N_x N_y N_z \cdot t}{N_y N_z \cdot \left(\frac{N_x}{M} + 6\sigma\right) \cdot t} = \frac{M}{1 + \frac{6\sigma M}{N_x}}$$

Notice that for one and two processors, the law above is not true: For one processor no borders are added ($A_1(1) = 1$), whereas for two processors, only one border is added per processor ($A_1(2) = 2/\left(1 + \frac{6\sigma}{N_x}\right)$).

For the second algorithm, we must take into account the time necessary for filling and emptying the pipeline:

$$A_2(M) = \frac{N_x N_y N_z \cdot t}{(M-1)L\frac{N_x}{M} t + \frac{N_x N_y N_z}{M} t} = \frac{M}{1 + \frac{L(M-1)}{N_y N_z}}$$

Each of the two algorithms can be the most efficient in different situations. The first algorithm is efficient if the standard deviation of the Gaussian is low and the connection network has a high latency. The second one is more efficient in sparing

Table 1: Theoretical performances of the two parallel recursive Gaussian filtering algorithms.

Quantifier	Borders sending algo 1	Pipeline algo 2
Computation time	$N_y N_z \cdot \left(\frac{N_x}{M} + 6\sigma\right) \cdot t$	$\frac{N_x N_y N_z}{M} \cdot t + (M-1)L\frac{N_x}{M}t$
Amount of data sent	$6N_y N_z \sigma$	$16N_y N_z$
Number of messages	2	$\frac{2N_y N_z}{L}$
Maximal acceleration	$\frac{M}{1+\frac{6\sigma M}{N_x}}$	$\frac{M}{1+\frac{L(M-1)}{N_y N_z}}$

processor time and network bandwidth, especially if the standard deviation of the filter's Gaussian is high. The drawback of sending many messages can be dealt with by tuning the L parameter (number of lines sent in one message). In practice, L is determined experimentally, as a function of the network latency: it has high values when the latency is high, while low latency networks can tolerate small L's, thereby improving the maximal acceleration. The second algorithm has another advantage: The minimum width of a processor domain is 4 points, whereas in the case of the first algorithm the minimal width is 6σ. This enables our pipeline recursive filtering algorithm to properly work with a much higher number of processors. Another advantage of the second algorithm is, off course, the fact that it is more accurate. In our experiments we chose to use the pipeline recursive filtering over the borders sending algorithm.

4. Results

In this section, we present an application for which non-rigid registration is essential. The input data consists in magnetic resonance images of a size which is typical for many clinical applications, which allows us to simulate the algorithm behavior in real use.

4.1. Medical context

The application we present is related to functional neurosurgery for Parkinson's disease [14]. Electrodes are introduced in a small, deeply located, nucleus of the brain, called the *subthalamic nucleus*. This nucleus is targeted on pre-operative MRI acquisitions. During the intervention, which is performed in the operating room, outside the MRI unit, an electrophysiological study is performed with the electrodes to check the pre-operatively determined target position. This operation is time-consuming and *cerebro-spinal fluid* (CSF) leaks can lead to *pneumocephalus* (presence of air in the intracranial cavity). The pneumocephalus provokes a brain deformation that can potentially cause errors in the pre-operatively determined position of the nuclei. The purpose of the study [14] is to a posteriori validate the pre-operative planning by using non-rigid registration in order to determine the displacement of the subthalamic nuclei due to the per-operative pneumocephalus.

This is done after the intervention, and our time constraints are due to usability needs: several minutes.

4.2. Experimental setup

One pre-operative and one post-operative three-dimensional T1 MR images of sizes 256x256x120 were acquired (Figure 5). For each image, the area outside the brain was masked. This enables the algorithm to better register the brain, without taking into account the skull.

Two image pyramids of height 5 were built, and we used 40 iterations at each pyramid level, which is usually enough in order to achieve a good registration. The hardware platform consists of 15 2GHz Pentium IV PC's with 1GB of RAM, linked through a 1GB/s Ethernet network.

Figure 5(d) presents a detail of a 2D slice of the 3D post-operative image. Figure 5(e) presents the same detail of the slice at the same position in the pre-operative image after rigid registration. On the post-operative image one can see that, due to the fact that the two electrodes were not withdrawn simultaneously, the pneumocephalus has a stronger influence on the left hemisphere of the brain (on the *right* side of the images). The deformation produce by the pneumocephalus has been corrected by the non-rigid registration (Figure 5(f)).

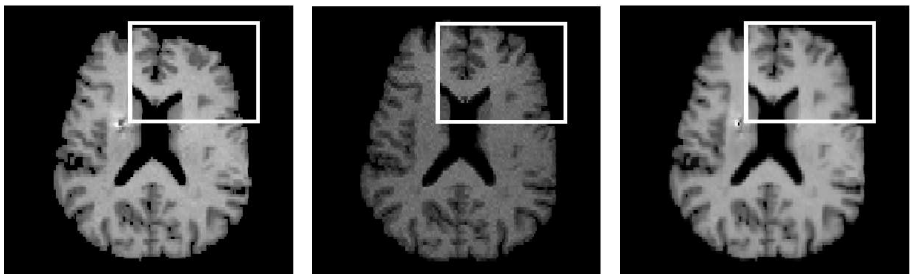
4.3. Performance results for the whole algorithm

The computation times and the parallel acceleration are presented in Figure 6. By using 15 processors, we obtained an acceleration of 11 (reducing the computation time from 40min to 3min30). This can be justified by the fact that some parts of the algorithm (creation of the image pyramids, computation of the gradients) are still sequential. For some configurations, the acceleration that was obtained was larger than the number of processors. We link this fact to the performance of the machines' memory. Since the algorithm uses large quantities of memory, cache misses are rather frequent when run sequentially. When spatial blocks are smaller, cache misses occur much less often, which largely improves the algorithm performances.

4.4. Performance results for the parallel filtering algorithms

The same setup as above was used, except that the computation time was only measured for the parallel filtering section, performing a Gaussian smoothing with a standard deviation of 5. Figure 7 shows the computation times for the two algorithms compared to the theoretical time estimated by our analysis in section 3.4.3. The t parameter has been estimated based on the computation time with one processor.

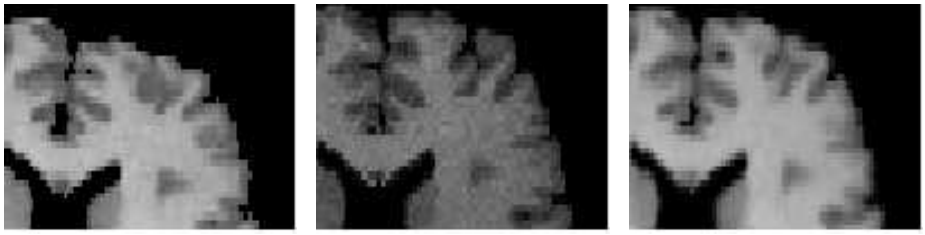
First, the pipeline recursive filtering algorithm is almost twice as fast when a large number of processors is used. Second, for each of the two algorithms, the



(a) Slice of the post-operative (source) image.

(b) Slice of the pre-operative (reference) image.

(c) Slice of the registration result.

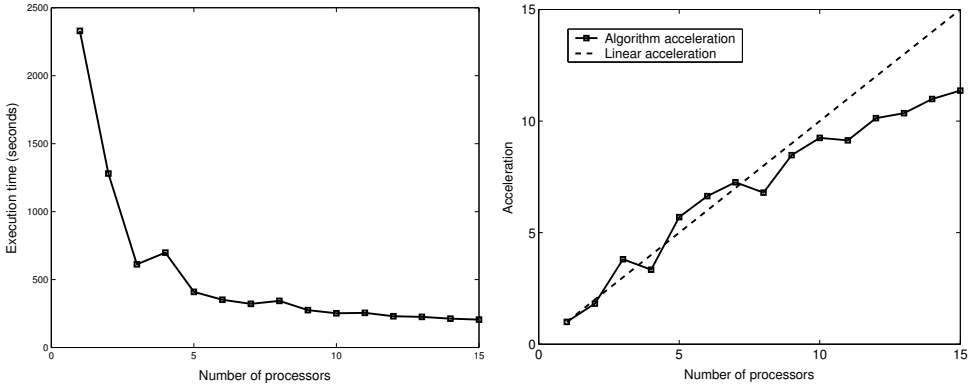


(d) Zoom of the post-operative (source) image.

(e) Zoom of the pre-operative (reference) image.

(f) Zoom of the registration result.

Fig. 5. Result of the non-rigid registration: The middle column displays a slice of the 3D pre-operative image. The left column represents the slice at the same position in the post-operative image, after rigid registration. The right column displays the result of the registration of the post-operative image on the pre-operative one. The upper row (Figures (a), (b) and (c)) presents complete slices of the three-dimensional post-operative, pre-operative and result images. The lower row (Figures (d), (e) and (f)) presents a zoom on a significant area of the same slice. One can see the strong pneumocephalus caused by the intervention at the upper right corner of the brain in the post-operative image (Figure (a),(d)). From Figure (c) and (f), one can see that the deformation has been corrected, and that the post-operative image has been deformed to best match the pre-operative one (Figure (b) and (e)).



(a) Computation time graph shows a reduction from about 40 minutes to 3min30.

(b) Algorithm acceleration (continuous line) compared to the ideal linear acceleration (dotted line). The acceleration is larger then the ideal one for some configurations and keeps growing until up to 15 processors.

Fig. 6. Experiment results: computation time and parallel acceleration.

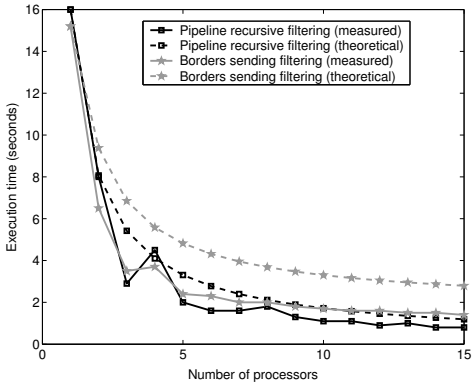


Fig. 7. Recursive filtering times: The second algorithm scales better than the first. The measured computation times are smaller than the predicted ones.

measured time is almost always above the estimated one. In the case of the second algorithm, an acceleration of 20 has been measured with 15 processors, whereas our model predicted 13.4. We believe that this is also due to the low performance of the personal computer memories (cache misses effect).

When performing this experiment, we chose a σ of 5, since a larger value would not have allowed to use 15 processors with the first algorithm (the processor domains would have been smaller than the borders they had to send). In practice, this is a great advantage for the pipeline recursive filtering algorithm.

4.5. Controlling the registration software through a graphic interface

In our current system [15], the user has access to the algorithm functionality through a graphic user interface running on a visualization workstation within the clinical environment. By using an SSL-based secure connection, the graphical client connects to a registration service running on a cluster within our laboratory. In order to preserve the patient anonymity, the client only sends the image data that is strictly necessary for the registration (image size and intensity values). Additional information already present inside the files (especially the one concerning the patient's identity) is not sent through the network. During the registration, the user receives real-time information about the status of the algorithm, such as an intermediate result. The user is thus able to fully control the registration and, if needed, modify the registration parameters. Since the user and the computation cluster are at distant locations and linked through a low-performance network, the data is compressed before being sent through the network, and decompressed upon receipt. Our tests show that the software has reasonable response times (about 30 seconds per image update) even if we use a network as slow as a 512kbits/s DSL modem. Of course, the response time is largely improved by the usage of a faster network.

4.6. Grid in a heterogeneous environment

Up to now, we considered the case of an algorithm running on an homogeneous cluster. Performing the registration on a cluster that includes remotely located nodes raises some problems:

Fault tolerance In such a cluster communications can fail more easily than in one that is entirely locate in a single place. Some authors have addressed this issue, and there are communication libraries, such as MPICH-V [16], that can recover errors.

Performance For efficiency reasons, each node should communicate the same quantity of data in the same time interval. If the nodes at distributed at remote locations, they should be linked by a long distance high speed network. Such a network is not in widespread use in the clinical environment,

and its cost largely exceeds the price of the PC's that we use of the registration. Therefore, we believe that such a solution would be possible to implement, but not financially interesting.

It is however possible for the user to be remotely located from the cluster. By using a Web interface and Globus, we have tested the software with a cluster located in a different city than the user [17]. After authentication, the user fills a Web form with the algorithm parameters and the images file names, and gets back the results at the end of the registration. This type of interaction has the advantage of not requiring anything else but Internet access and a Web browser on the client side. Experiments showed that, by using a low performance long distance network to connect the user to the cluster, only a few minutes are added to the total execution time.

5. Discussion

We currently use our parallel non-rigid registration software on two hardware platforms. Users can either execute it on the dedicated cluster described in Section 4.2, or, if computation nodes are unavailable, they can use workstations available in the laboratory. In the later case, the parallel algorithm shares bandwidth with several network-intensive applications, such as NFS. Our experience proves that the parallel implementation solves the computation time problem, but it creates other needs:

- Usability through a graphics user interface (GUI): since the registration is often integrated into a larger computational chain that generally contains visualization, the challenge is to use the computational power provided by the cluster while keeping the graphics performance intact [15];
- Access to data: data sources such as databases and medical image acquisition equipment should be easily accessible by the computation cluster, even in the presence of security mechanisms such as firewalls;
- Security and confidentiality issues arise when using long distance networks or externalizing the images;
- Transparent management of the high-performance computing resources: systems already in use, such as Globus, provide access to distant computing resources;
- The availability and the automatic localization of computing resources.

We believe that there are three low-cost solutions:

Community resources Large organizations can afford to purchase cluster nodes, and locate them either in a centralized manner (a single data center) or in a distributed one (clusters located inside different organization departments). The costs essentially consist in purchasing and maintaining a large hardware infrastructure. The solution has several advantages: it provides a powerful computation infrastructure; and this structure is completely under the control of the organization, which eliminates most of the possible security issues. The infrastructure challenges are related to the usability through a GUI, and the ease of connectivity to data sources.

The laboratory grid Another method to have easy access to computation power is to use the workstations already in place. If bi-processor workstations with sufficient memory and a reasonably fast network are available, they can be used as a low cost pool of computing resources. The main advantage consists in the acquisition price (one bi-processor workstation is cheaper than two mono-processor ones) and the maintenance cost (fewer computers to administer). This platform poses no security problem, and its high integration implies easier access to data sources and graphics workstations. The main drawback is the system performance. Furthermore, since the computation cluster has to be relatively homogeneous, the organization should employ a centralized hardware acquisition policy [18].

The purchase of computing power The most flexible alternative is the purchase of computing power from a professional provider. For the healthcare organization there is little or no initial investment, and the infrastructure is maintained by professionals that insure the quality of service. The healthcare organization pays the service on a contract basis. Two issues have to be addressed in this scenario. First, the software running on the provider's cluster should have access to data sources and visualization equipment inside the healthcare organization. Second, the computing power provider and the healthcare organization must trust each other. Indeed, regulations concerning the confidentiality of medical data make it hard for hospitals to externalize medical images. Reciprocally, the computing power provider has to insure the security of its own systems while running on them software coming from various sources, and whose functioning is a priori not known.

To summarize, the first of the above scenarios ("community resources") has the advantage of providing a high computing power, but at a relatively high cost. The second one ("the laboratory grid") has a low cost, but it can only offer limited computing power. The third scenario offers the best tradeoff between cost and computing power, but it raises several interoperability and security issues.

6. Conclusions et perspectives

We successfully managed to transform the long and cumbersome non-rigid registration process into a task that lasts only a few minutes, which was our goal. We achieved this by using a low-cost cluster of workstations, linked through a regular Ethernet network. The algorithm uses the network in an efficient manner and avoids one-to-all and all-to-all communications. We presented two algorithms that perform recursive Gaussian filtering. The first one can take advantage of very high latency networks. The second one can efficiently tackle the case of very wide Gaussians, sparing processor time and network bandwidth. It can also adapt to high latency networks by tuning one parameter. The parallel software currently runs in our laboratory, either on a dedicated cluster or on production workstations in place, and it can be controlled through a graphic interface or a Web one. However, in a real clinical environment, several additional issues have to be addressed, such as the transparent access to data sources and computing resources, and the security of the system. These are currently central issues for medical grid environments.

Acknowledgments

This work has been partially supported by the French Region of Provence-Alpes-Côte d'Azur through the fellowship of Radu Stefanescu. Professor Didier Dormont from the Neuroradiology Department and LENA UPR 640-CNRS, Salpêtrière Hospital, France, kindly provided the images used in this paper.

References

[1] D.T. Gering, A. Nabavi, R. Kikinis, N. Hata, L.J. O'Donnell, E.L. Crimson, F.A. Jolesz, P.M. Black, and W.M. Wells, An Integrated Visualization System for Surgical Planning and Guidance using Image Fusion and Open MR, in *Journal of Magnetic Resonance Imaging*, 13:967–975, 2001

[2] X. Pennec, A. Roche, P. Cathier, and N. Ayache, Non-Rigid MR/US Registration for Tracking Brain Deformations, in *R.S. Blum and Zh. Liu, editors, Multi-Sensor Image Fusion and Its Applications*. Marcel Dekker Inc., 2005. In press.

[3] O. Skrinjar, A. Nabavi, and J. Duncan, Model-driven brain shift compensation, in *Medical Image Analysis*, 6(4):361–374, 2002

[4] T. Rohlfing, C.R. Maurer Jr., Non-Rigid Image Registration in Shared-Memory Multiprocessor Environments with Application to Brains, Breasts, and Bees, in *IEEE Transactions on Information Technology in Biomedicine*, 7(1): 16-25, 2003

[5] H. Muller, Grid Computing at the University Hospital of Geneva, in *Proc. of HealthGrid 2003*. European Commission, DG Information Society.

[6] C. Broit, Optimal Registration of Deformed Images, *PhD thesis, University of Pennsylvania*, August 1981

[7] R. Bajcsy and S. Kovačič, Multiresolution Elastic Matching, in *Computer Vision, Graphics and Image Processing*, 46:1-21, 1989

[8] J.-P. Thirion, Image Matching as a Diffusion Process: an Analogy with Maxwell's Demons, in *Medical Image Analysis*, 2(3), 1998.

[9] P. Cachier, E. Bardinet, D. Dormont, X. Pennec and N. Ayache, Iconic Feature Based Nonrigid Registration: The PASHA Algorithm, in *CVIU – Special Issue on Nonrigid Registration*, 89:272–298, 2003.

[10] X. Pennec, P. Cachier and N. Ayache, Understanding the "Demon's Algorithm": 3D Non-Rigid registration by Gradient Descent, in *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI'99)*, LNCS 1679, pages 597–605, 1999. Springer Verlag.

[11] J. Rexilius, S.K. Warfield, C.R.G. Guttmann, X. Wei, R. Benson, L. Wolfson, M. Shenton, H. Handels and R. Kikinis, A Novel Non-Rigid Registration Algorithm and Applications, in *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI'01)*, LNCS 2208, pages 923–931, 2001, Springer Verlag.

[12] D. Rueckert, L. Sonoda, D. Hill, and D. Hawkes, Non-Rigid Registration of Contrast-Enhanced MR Mammography, in *IEEE Trans. Medical Imaging*, 18(8):712-721, 1999.

[13] R. Deriche, Recursively Implementing the Gaussian and Its Derivatives, in *Proc. of The Second International Conference On Image Processing*, pages 263-267, 1992.

[14] E. Bardinet, P. Cathier, A. Roche, N. Ayache, and D. Dormont, A Posteriori Validation of Pre-operative Planning in Functional Neurosurgery by Quantification of Brain Pneumocephalus, in *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI'02)*, LNCS 2488, pages 323–330, 2002, Springer Verlag.

[15] R. Stefanescu, X. Pennec, and N. Ayache, A Grid Service for the Interactive Use of a Parallel Non-Rigid Registration Algorithm, in *Proc. of HealthGrid 2004*. European Commission, DG Information Society.

[16] G. Bosilca, A. Bouteillier, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selhikov, MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes, to appear in *Proc. of ACM/IEEE International Conference on Supercomputing SC 2002*

[17] D. Hill, X. Pennec, M. Burns, M. Parkin, J. Hajnal, R. Stefanescu, D. Rueckert, and J. Montagnat, Intraoperable Medical Image Registration Grid Service, in *Proc. of HealthGrid 2004*. European Commission, DG Information Society

[18] R. Ziegler. Pharma GRIDS: Key to Pharmaceutical Innovation?, in *Proc. of HealthGrid 2004*. European Commission, DG Information Society.