Robust Registration of Multi-Modal Medical Images: Towards Real-Time Clinical Applications

Sébastien Ourselin, Xavier Pennec, Radu Stefanescu, Grégoire Malandain, Nicholas Ayache

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Robust Registration of Multi-Modal Medical Images: Towards Real-Time Clinical Applications

Sébastien Ourselin, Xavier Pennec, Radu Stefanescu, Grégoire Malandain, Nicholas Ayache

N° -4333

December 2001

THÈME 3

ISSN 0249-6399 ISRN INRIA/RR--4333--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Robust Registration of Multi-Modal Medical Images: Towards Real-Time Clinical Applications

Sébastien Ourselin, Xavier Pennec, Radu Stefanescu, Grégoire Malandain, Nicholas Ayache

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet EPIDAURE

Rapport de recherche n° -4333 — December 2001 — 28 pages

Abstract: Over the last years, high performance computing has become a key step to introduce computer tools in the medical field, especially in image guided surgery and therapy (IGT). Among these tools, there is a special need for real time registration algorithms. By real time, we mean faster than the image acquisition to add an acceptable overhead (say under 1mn for IGT applications). To tend toward such small registration times, one usually simplify and adapt algorithms so that they become application and data specific. This process involves a lot of designing and programming work for each application, and reduces the generality and robustness of the method. Our goal in this paper is to show that a general registration algorithm (here the block-matching scheme of [ORPA00]) can be parallelised on a cheap and standard parallel architecture with a reasonably small amount of additional programming work, thus keeping intact the algorithm performances and generality.

For medical applications, we show that a cheap cluster of bi-processor PCs connected by an Ethernet network is a good trade-off between the power and the cost of the parallel platform. Portability, scalability and safety requirements led us to choose OpenMP to program multi-processor machines and MPI to coordinate the different nodes of the cluster. The resulting computation times are very good on small and medium resolution images (resp. 19 seconds and 45 seconds on 5 bi-processors), and they are still acceptable on high resolution MR images (resp. 1mn35 for 5 bi-pro and 70 seconds for 10 bi-pro). One can obtain even smaller times by stopping the algorithm one step before the final level in the multi-scale pyramid, at the cost of a slightly degraded accuracy (relative precision below the voxel size): the registration of high resolution MRI is down to 1 mn on a cluster of only 2 bi-processor PCs.

Key-words: Registration, real-time, medical imaging, parallelism, image guided therapy

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Recalage robuste d'images médicales multi-modales : vers des applications cliniques en temps réel

Résumé : Au cours des dernières années, la rapidité des outils informatiques de traitement d'images médicales est devenue un paramètre important pour leur utilisation clinique. C'est particulièrement le cas pour la thérapie assistée par l'image où l'un des principaux problèmes est le recalage tridimensionnel rapide : typiquement, nous souhaiterions fusionner des images per-opératoires de taille 256x256x60 en moins d'une minute. Pour tendre vers des temps de calcul aussi faibles, on peut simplifier et adapter les techniques à chaque type de donnée, créant ainsi des algorithmes dédiés à une tâche spécifique. Ceci demande un travail important de reprogrammation (un algorithme pour chaque application) et réduit la robustesse et la généralité de la méthode. Notre but dans cet article est de montrer qu'on peut aussi valablement accélérer un algorithme de recalage général (en l'occurence une approche par appariement de régions développée dans [ORPA00]) sans en dégrader les performances en le parallélisant avec peu de reprogrammation sur une architecture d'un coût réduit.

Pour des applications médicales, nous montrons qu'une grappe de PC bipro standards, reliés par un réseau ethernet, réalise un excellent compromis entre le prix de revient et la puissance de calcul de la plateforme. Des considérations de portabilité, de stabilité et de performances nous ont conduit à choisir les logiciels OpenMP pour programmer les machines multi-processeur et MPI pour coordonner la transmission de données entre les machines. Les temps de calculs obtenus sur des images de basse et de moyenne résolution sont respectivement de l'ordre de 20 et 45 secondes sur une grappe de 5 PC bi-processeurs, et restent raisonnables sur des images de haute résolution (de l'ordre de 1mn35 pour 5 bi-pro et 1mn10 pour 10 bi-pro). Nous pouvons obtenir des temps de calcul encore plus faibles en stoppant l'algorithme avant le dernier niveau du schéma multi-échelle, au prix d'une perte de précison inférieure à la taille du voxel : le recalage des images de haute résolution prend alors moins d'une minute sur seulement 2 PC bi-processeurs.

Mots-clés : Recalage, temps réel, imagerie médicale, parallélisme, thérapie guidée par l'image

Contents

1 Introduction

One major concern in image-guided surgery is the simultaneous need for high performance – hence time-consuming algorithms – and the time constraints imposed by their used in the operating room. Over the last years, High Performance Computing (HPC) has become a key step to introduce computer tools in the medical field, especially in image guided surgery [Jol97] and neuro-surgical intra-operative imaging [Kik00a]. All these cases can be classified as Image-Guided Therapy (IGT), whose challenge is to introduce image processing tools in the clinical environment for planning, targeting, and monitoring [Kik00b]. In neurosurgery, for craniotomies, pre-operative guidance (using pre-operative images) allows the surgeon to select the best trajectory that more safely invades the tissue, using stereotactic systems. This step drastically reduces the surgery time in the operating room. Moreover, to complement the pre-operative planning, the surgeon uses an intra-operative guidance, in order to control his trajectory. Usually, image-guided surgery systems track surgical instruments during the operation and render the tracked devices within intra-operative images. Nevertheless, these image-guided surgery systems are limited by the *static knowledge* of the anatomical brain patient structures. Indeed, the positions of anatomical structures change when the skull is opened, when the cerebrospinal fluid (CSF) leaks, or when the tumor tissue is removed [WFG+00]. Intra-operative imaging (interventional images) is being developed to solve these problems, and also to detect complications during surgery, such as bleeding. Typically, fluoroscopic, sonographic and more recently ultrasound images (US) are used thanks to their low cost and simplicity of use [AVV90, BYT+97, PCA01]. However, due to the low signal-to-noise ratio in US images, the applications in image-guided surgery are currently very limited. Thanks to technological developments, it is now possible to have 3D interventional imaging with a high-resolution and hight contrast [GSS+96], such as CT interventional images [LPA84] (resection of glial brain tumors). During the last five years, MRI-guided interventional procedures [SJR95] are being developed and the quality of the open-MR images (see Figure 1) is always improving, particularly by the line scan imaging [KMM+01] [1].

1.1 Registration in the context of Image Guided Therapy

A typical example is a surgical and visualisation system developed at the Brigham and Women's Hospital based on open-MR images guidance completed by the 3D

[1] In contrast to 2D Fourier transform imaging, line scan imaging is a technique that acquires the line of an MR image in multiple independent spin-echo shots.

Slicer software package (directly integrated in the operating room) [Ger99, GNK+99]. This navigation tool has been used in 45 neurosurgical interventions [GNK+01]. During each craniotomy, the positions of anatomical structures change, and 3 to 5 new datasets were acquired. The registration between pre-operative and intra-operative data during craniotomy was completed within five minutes and provided to the surgeon. Although intra-operative brain deformation is non-rigid, rigid registration was sufficient in these cases, since the primary use of the pre-surgical data was to guide the initial approach [GNK+01]. The 3D Slicer Software uses the maximisation of mutual information for the rigid registration [WVA+96].

Figure 1: *Per-operative image acquisition using open MR for image-guided surgery application. Image courtesy of Surgical Planning Laboratory, Brigham and Women's Hospital, Boston, USA.*

In a recent paper, Netsch *et al.* presented a multi-modal registration algorithm [NVMW01] based on local correlation (LC) [WRN+99] optimised by a Gauss-Newton technique. Using a voxel selection based on the largest variance, they drastically reduce the computation time using only 10% of image voxels. Moreover, using a local similarity criterion with a low complexity, they achieved fast registration of MR and CT volume. The size of MR images were $256 \times 256 \times 128$ voxels and the CT images were $512 \times 512 \times 87$. With 10% of the image voxels, the registration procedure takes about one minute. Nevertheless, the optimisation procedure used is not robust to the outliers since it is a least-square minimization [RL87].

Recently, we presented a multi-modal registration algorithm, also based on a local similarity measure, which explicitly takes into account the presence of outliers [ORPA00]. We believe that our algorithm could lead to faster and more robust

results while considering more image information. Indeed, one has to be careful when removing data to speed up the computations, since this also reduces the final accuracy and robustness. Our idea is that the registration algorithm could be easily parallelised on a cheap hardware platform in order to gain in computation time while keeping intact the algorithm's performance.

1.2 Real-time and parallelism

When talking about real time constraints, one have to specify the typical time interval. Indeed, registering 5 images per second is quite different from one image every 15 minutes. In [CJ99], Cox *et al* rigidly register fMRI 3D time-series of 80 low resolution images of the brain ($128 \times 128 \times 30$ voxels) in about one minute. In this case, the registration of an individual image is 2 to 3 times faster than its acquisition. In other words, the algorithm is *real time* if the registration is faster than the medical need. In Image Guided Therapy, the acquisition time is typically from 1 to 5 mn per 3D image, but with a much higher resolution ($256 \times 128 \times 60$ in [GNK+01]). Thus, we need to keep the registration time under say 1 mn to add an acceptable overhead to the image acquisition time. Moreover, with the future evolution of the MR acquisition systems, the acquisition time will likely decrease for an increasing resolution.

A simple but very expensive solution for speeding up computations is to use a powerful parallel computer, such as a massively multi-processor architecture (see Figure 2), as done in [CRM96], using a DECmpp (MasPar). This solution has the another disadvantage of needing a dedicated engineer to maintain the machine. Our goal in this paper is to show that a registration algorithm (here the block-matching scheme of [ORPA00]) can be parallelised on a cheap and standard parallel architecture with a reasonably small amount of additional programming work.

Other important requirements for the parallel environment are portability and scalability, i.e. the capacity of the software to be adapted to other architectures and different network configurations, and safety, since the software is intended to be used in the safety-critical environment of a surgical operation. Thus, the choice of a mature and well-tested environment is important.

1.3 Paper organisation

We detail in Section 2.1 the possible hardware platforms for parallel computing. The tradeoff between the price, the genericity, and the computer power leads us to select a cluster of multi-processor PC, connected by Ethernet. Section 2.2 presents the main

Figure 2: *Left: Wildfire Cluster with scalable memory, architecture built on SUN E5k and E6k chassis, 36 250MHz UltarSPARC-II CPUS and 9GB RAM (source: [Kik00a]); Right: SGI Origin 3400, architecture built on 32-processor shared-memory system and 64GB RAM under IRIX 6.5 (source: http://www.sgi.fr/servers/).*

features of the chosen software environments, namely OpenMP to exploit several processors on a single machine, and MPI (Message Passing Interface) to coordinate the different machines of the cluster.

Then, we recall in Section 3 the principles of the registration algorithm, a block matching technique detailed in [ORS+01, ORPA00]. Section 4 focuses on successive parallelisations of the main time consuming steps: the computation of the vector field and the image resampling. Finally, we present and analyse in Section 5 the gain in computation time with respect to the number of processors and the size of the data. Results show that we finally obtain registration times of less than one minute (which was our aim for IGT applications) for high resolution images on a cluster of only 2 bi-processor PCs. With lower resolution images (the Vanderbilt database), we even obtain a registration time of 15 seconds on the same cluster configuration.

2 The parallel environment

2.1 Hardware choices

One important question is the choice of the hardware platform, as it implicitly determines the programming tools used to parallelise the algorithm. From the hardware point of view, a parallel computer is essentially made of processors, memory, and an

interconnection network, a device that connects processors between each other and with the memory. The main difference between parallel computers is in the number of processors and the way they are connected to the memory. We can distinguish two large categories: shared and distributed memory computers.

Shared memory computers have all the processors connected to the same memory. This model has the advantage that all the processors can work on the same memory block and they do not need to explicitly exchange information. However, such computers are limited to just a few processors. Even worse, there are some limits on the amount of sequential memory that can be addressed by a processor (on some 32 bit operating systems, the internal memory is limited to 4 gigabytes). These computers are also known as Symmetric Multi-Processors (SMP).

Distributed memory computer are composed of *nodes*, each of them containing a processor and a certain amount of memory. Nodes interact by sending each other messages through the interconnection network. By this method, there is virtually no limit on the number of processors that a parallel computer can contain. The main drawback of this approach is that synchronisation and information exchange through a network typically take much longer. Another disadvantage is the fact that global data structures have to be replicated on every node for performance reasons. This sometimes leads to a larger total memory need. Classically, distributed memory computers contained dedicated hardware and use expensive interconnection networks. A newer choice in high performance computing is that of computer cluster, made out of low cost workstations (often standard personal computers) linked together through a network, usually Ethernet.

A generic choice. For medical applications, and more especially IGT, we believe that a cluster of symmetric multi-processor (typically PCs) connected by an internal network is a good trade-off between the power and the cost of the parallel platform. The system can be used as a research tool or as an embedded system in a clinical environment. In this article, we used up to 10 PCs of our lab (bipro Pentium III 933 Mhz operating Linux 2.2.18/libc2.1), connected by a fast Ethernet Network (100 Mbps using 2 interconnected 1Gbps switches). Such a hardware configuration is already present in many labs and hospitals.

2.2 Software choices

Since we have a sequential version of the algorithm, we looked for a method to parallelise it with minimum reprogramming. A parallel programming model that seemed appropriate is the Single-Program-Multiple-Data (SPMD) approach, in which all the processors involved execute the same program, but on different data. We used an OpenMP compliant C compiler to program multi-processor machines and the Message Passing Interface (MPI) to coordinate the different nodes of the cluster. One of the main advantages of using OpenMP and MPI is the portability over many different kind of SMP clusters. Both OpenMP and MPI are standards and do not depend on the machine architecture, operating system, and network topology. Moreover, they both use the same programming model (SPMD). We now detail the main features of OpenMP and MPI.

OpenMP

OpenMP consists in a set of compiler directives and library functions that can be used to specify the behaviour of a program when executed on shared memory multiprocessor computers [Ope98]. The OpenMP standard exists for the C, C++, and Fortran languages. All the code sections in this paper use the OpenMP C standard. A large part of the OpenMP C standard is implemented as "pragma" compiler directives, which eases the parallelisation of sequential C code and enables the compilation of C programs using OpenMP by standard C compilers. We now describe two of the core notions of OpenMP: *parallel sections* and *parallel "`for`" statements*.

A parallel section is a piece of code executed by all the processors of some node. The only difference between the flows of instructions executed by different processors is the presence of a unique identification number for each node.

```
#pragma omp parallel
{
  ... this part is executed in parallel by all the processors
}
```

By default, all variables that are global to the parallel section are shared by all the processors. In order to avoid the concurrent and possibly incoherent modification of some of the variables by different processors, it is possible to declare them as private, in which case the variable is replicated on all processors.

The parallel "`for`" statement seems to be the most powerful feature of OpenMP. It enables the parallelisation of "`for`" loops. The only constraint is that all iterations have be independent of each other, which means that the result of the execution of the "`for`" loop does not depend on the order of execution of the different iterations. This way, the execution of the "`for`" can be done in parallel by the available processors.

```
#pragma omp parallel for
for(i=0; i<n; i++)
{
  ...
}
```

MPI

MPI is a standard for communication libraries between the nodes of a cluster [MPI95]. There are MPI bindings for the C/C++ and Fortran languages. As in OpenMP, each MPI node has a unique identification number called the *rank*. Among the most powerful functions provided by MPI, we can:

- **send** a message to a node,
- **receive** a message from a node,
- **broadcast** a message from a node to all other nodes,
- **scatter** subparts of a "list" to different nodes,
- or **gather** the subparts of a "list" from nodes.

An important difference between MPI and OpenMP is that each MPI process has its own data and variables, as the processes are not supposed to be able to share their memory.

The programming model

As stated in Section 2.1 we use a cluster of multi-processor workstations. Using MPI, we run a UNIX *process* on each machine of the cluster. Each process uses OpenMP to start several *threads* on its machine. A thread is a light weight UNIX process that shares the same memory space with all other threads issued from the same *parent process*. Since processes do not have access to each other's memory, we run them on different workstations, while threads that share the same memory

space are spawned on the same machine. For efficiency reasons, we choose to run on each machine a number of threads equal to its processor number. The kernel of the operating system automatically assigns different threads to different processors. Processes running on different workstations collaborate to solve the problem. Among them, there is a special process called the *master* that does everything that cannot be done in parallel, such as input/output (I/O) operations and tasks that have be done sequentially. All the other processes are called *slaves*. In our program the master is *not dedicated*, which means that it also does everything that regular slaves do.

3 The sequential algorithm

The algorithm we parallelise in this article computes a parametric transformation (rigid, similarity, or affine) only from correspondences between very similar areas in both images. Firstly, we find point-to-point correspondences between the floating image (image to register) and the reference image. Then, we find the transformation that best superimposes the matched points. The point correspondences are obtained using a block matching strategy. This procedure is extensively described in [ORS+01] for the rigid registration of anatomical sections, in [ORPA00] for the multi-modal rigid registration of medical images, and in [POA00] to compute the mid-sagittal plane of the brain. In order to quickly approach the desirable optimum and to extend the capture range of the search, a multi-resolution scheme is used. In the above papers, we used a coarse to fine strategy for the block sizes. Here, we decided to use a small and constant block size ($4 \times 4 \times 4$ voxels) and to subsample the original images by (at most) a factor two in each axis at each level of the multi-resolution pyramid.

At the low resolution level, the search for correspondences is started either from the identity or from any provided initial transformation. Then, the registration parameters of each resolution level are used as the initialisation of the search at the next pyramid level, until the highest level is reached. As we will discuss in Section 5.3, an acceptable registration can be obtained before the highest resolution. Within each pyramid level, the optimisation is iterative, and we found experimentally that 3 iterations were sufficient to obtain an approximate convergence. For each iteration, the rigid transformation is computed by robustly minimising the distance between matched points using Least Trimmed Squares (LTS) [RL87]. The global organisation of the algorithm is briefly described in Fig. 3.

In order to further improve the robustness and speed-up in the algorithm, we first sort the image points using a local variance (computed for each block). This way, we

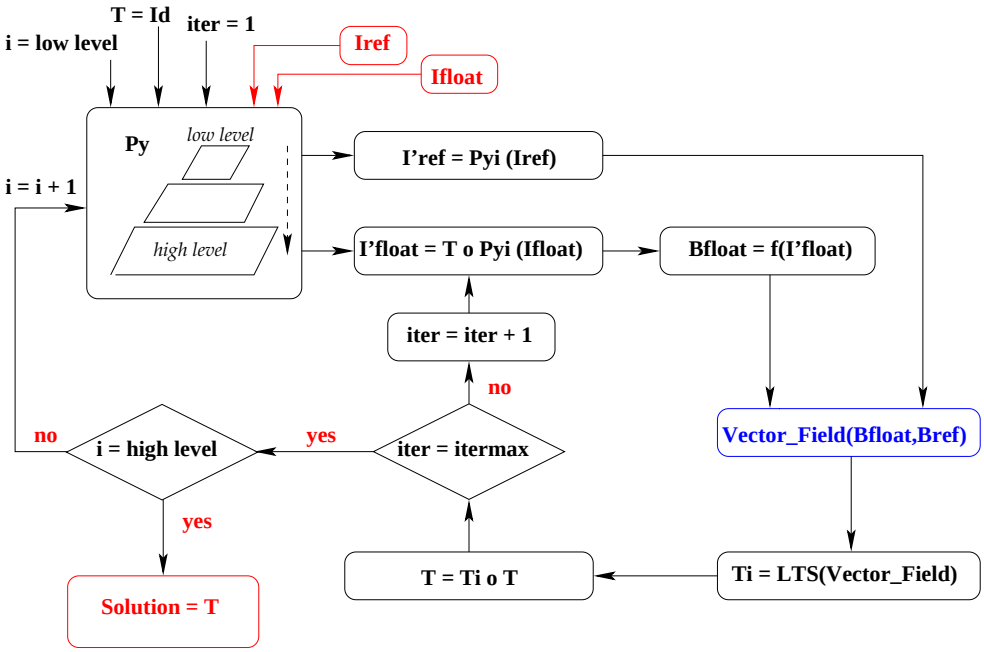

Figure 3: *Simplified diagrams of the registration algorithm. We notice in blue the vector field computation, that we would like to parallelise.*

can focus on a subset of the image representing the relevant voxels. The local variance is also used by [NVMW01] and is very attractive especially for large data volumes. By default, we use all the points for the low level (the variance percentage is equal to 100% for the low resolution level), and we divide by two the number of relevant voxels at each pyramid level, with a lower bound (typically 20%). This means that all the image information is taken into account for computing large displacements (at the low pyramid level), and the algorithm adaptively focuses on relevant image parts when estimating a more precise motion.

4 Parallel implementations

A profiling of the sequential version of the algorithm showed that the program spent most of its computation time (about 93%) in the vector field computation (Fig. 7, case of one mono-processor workstation). The remaining time was shared between image resampling, LTS minimization, and I/O operations. Hence, our first concern

was to parallelise the vector field computation. We investigate in Section 4.1 an MPI implementation and show in Section 4.2 how OpenMP can be used to further improve the computation times with multi-processor machines. Then, using this parallel vector field computation, we found that image resampling was the limiting operation. With a large number of processors, it even took almost all of the computation time. As a consequence, the second concern was to parallelise the image resampling, detailed in Section 4.3.

4.1 MPI implementation of the vector field computation

Since this is the most time consuming step of the algorithm, it is important to make all processors participate. The algorithm 1 explains the loop for a single pyramid level. Let $List_{blocks}$ be the list of selected blocks (those which have a high enough variance, see Sec. 3) and N_{blocks} its size.

Algorithm 1 (sequential implementation of the vector field computation)

1: **for** each iteration **do**
2: **for** $i = 1..N_{blocks}$ **do**
3: **compute** the block $\mathcal{B}'_i$ that best matches block $\mathcal{B}_i$
4: **end for**
5: compute the new transformation T_{new}
6: $T \Leftarrow T_{new} \circ T$
7: resample the current floating image using T_{new}
8: **end for**

For the parallel algorithm, we choose a master slave implementation (see Section 2.2). Each process has a copy of the initial images, and applies locally the resampling in order to limit the communication time. The master divides the list of N_{blocks} blocks into N_{procs} sub-lists where N_{procs} is the total number of machines, including the slaves and the master. Each process computes N_{blocks}/N_{procs} block displacements, and sends its local sub-list Sub_i of correspondences to the master. Then, the master computes the new transformation and broadcasts it to all the slaves (algorithm 2).

4.2 OpenMP implementation of the vector field computation

Thanks to OpenMP, we may further reduce the communication time when using bi-processor workstations. Indeed, we can use MPI to distribute processes on each

Algorithm 2 (parallel implementation of the vector field computation)

1: **for** each iteration **do**
2: **parallel for** $i = 1..\frac{N_{blocks}}{N_{procs}}$ **do**
3: **compute** the block $\mathcal{B}'_i$ that best matches block $\mathcal{B}_i$
4: **end parallel for**
5: **Slaves:** send the sub-list Sub_i to the master
6: **Master:** $List = Sub_1 \cup Sub_2 \cup Sub_3 \cup ... \cup Sub_{N_{procs}}$
7: **Master:** compute the new transformation T_{new}
8: **Master:** $T \Leftarrow T_{new} \circ T$
9: **Master:** broadcast T to all the processes
10: **Slaves and Master:** resample the current floating image using T
11: **end for**

processor, but the interesting feature of OpenMP is that sharing the memory between threads on the same machine reduces the communication time. Thus, the idea is to use OpenMP within each bi-processor machine, and MPI to manage the cluster of machines. In Figure 4, we present the OpenMP computation time for different pyramid level from $32 \times 32 \times 32$ to $256 \times 256 \times 124$ on a bi-processor machine. We expected an acceleration of about 50%, and we obtained approximately 48%. In practice, using OpenMP instead of MPI did not significantly reduce the computation time, but decreased the necessary amount of memory. In the MPI case, we have to replicate the input data for each processor of a multi-processor machine, and that could be very constraining when using large images in floating point representation.

4.3 OpenMP implementation of image resampling

As the number of processors used becomes high, the resampling of images takes a higher percentage of the total computation time (see Fig. 6). An MPI implementation of the resampling would ask each process to take a part of the image, resample it and then send it to every other process. The communication time would certainly be larger than the computation time for the sequential implementation.

With an OpenMP implementation, each process (i.e. machine) does the resampling of all the points, but it shares the load among all the processors available on the machine. The current implementation uses this last approach. Since the resampling function processes all the points in a single "`for`" loop, the parallel version uses the *parallel "`for`" statements* of OpenMP, and the algorithm 3 is replaced by the algorithm 4. In figure 5, we present the computation time for different resampling

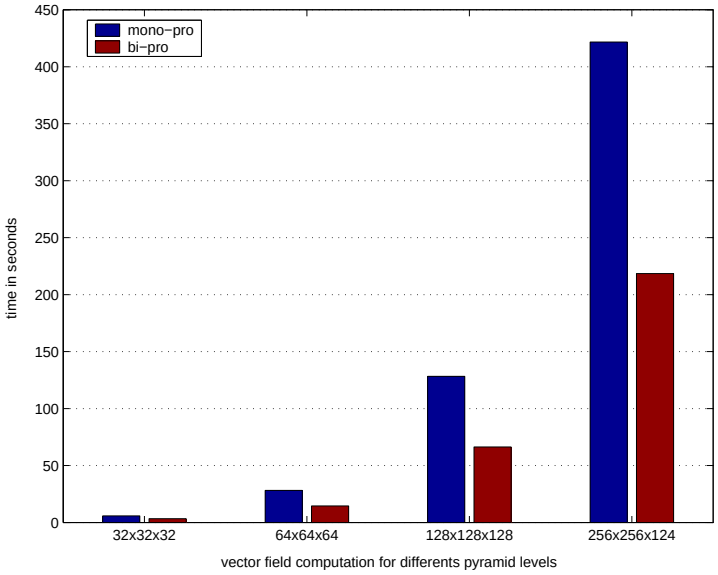

Figure 4: *OpenMP implementation of the vector field computation on mono- and bi-processor machines for different levels of the pyramid.*

of images with sizes ranging from $64 \times 64 \times 64$ to $512 \times 512 \times 512$. We expected an acceleration of about 50%, and we obtained 40%. The difference between theory and experimental results are essentially due to the bounded memory access speed (in our case 133 Mhz for 933 MHz Pentium III bi-processor).

Algorithm 3 (sequential implementation of image resampling)

1: **for** each point of the image **do**
2: resample point using the transformation
3: **end for**

Algorithm 4 (parallel implementation of image resampling)

1: **parallel for** each point of the image **do**
2: resample point using the transformation
3: **end parallel for**

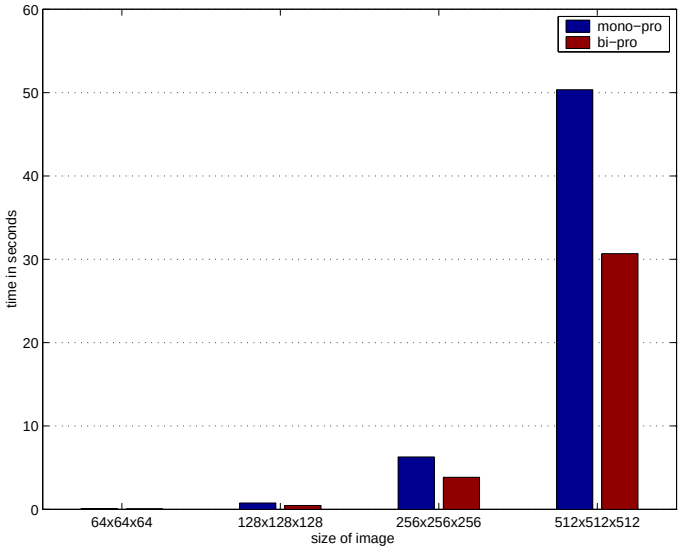

Figure 5: *OpenMP implementation of image resampling on mono- and bi-processor machines for different image sizes.*

5 Experiments

We illustrate our algorithm with two data sets. The first one is an MR/CT multimodal case from the Vanderbilt database [Wa97]. These images have a low resolution ($256 \times 256 \times 26$ voxels of size $1.25 \times 1.25 \times 4$ mm^3 for the MR and $512 \times 512 \times 28$ voxels of size $0.65 \times 0.65 \times 4$ mm^3 for the CT) but they represent a typical multimodal registration in a clinical application. To simulate an IGT application, we also used sets of pre- and post-operative T1 weighted MR images, acquired by La Pitié Salpêtrière Hospital (Paris) in the context of the treatment of Parkinson's disease by deep brain stimulation (DBS). These images typically have about $256 \times 256 \times 124$ voxels of size $0.9375 \times 0.9375 \times 1.4$ mm^3.

We registered the CT image on the MR-T1 weighted image, and the post-operative image on the post-operative image. As explained in Section 2.1, we used up to 10 PCs of our lab (bipro Pentium III 933 Mhz operating Linux 2.2.x/glibc2), connected by a fast Ethernet Network (100 Mbps using 2 interconnected 1Gbps switches). We applied our algorithm with mono-processor and bi-processor architecture independently, from one to ten machines. For each case, we made 100 regis-

trations in order to estimate the mean time of the vector field computation and the mean time for rest of the program.

5.1 Computation times of the parallel section

We present in Figure 6 the wallclock time for a typical registration problem, as a percentage of the computation time with one single CPU workstation, with respect to the number of machines of the cluster. The first observation is that the total computation time is drastically reduced with the first machines (82% of gain for 4 bi-processor machines), and decreases much more slowly with each additional machine (only a 10% gain for the next 6 machines).

The second observation is that the computation time of the vector field drops from 93% of the total computation time (for a mono-processor machine) to 51% (for a cluster of 10 bi-processor machines), reaching the computation time of the sequential part (which is obviously constant). This effect is more visible in Fig. 7, where we display the percentage of the time spent in the vector field computation

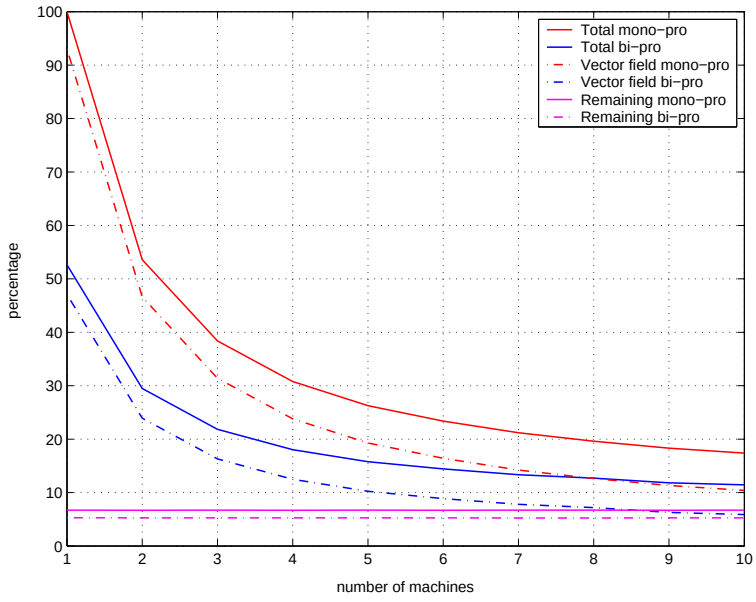

Figure 6: *Computation time of the algorithm using mono-processor and bi-processor from 1 to 10 workstations. We can see the time percentage for the parallel (vector field computation) and sequential (remaining) part of the program.*

and in the remaining sequential part. This means that we cannot gain much more by using additional machines: we need to investigate the parallelisation of the "constant time operations" to further improve the time performance. This will be discussed in Section 6.

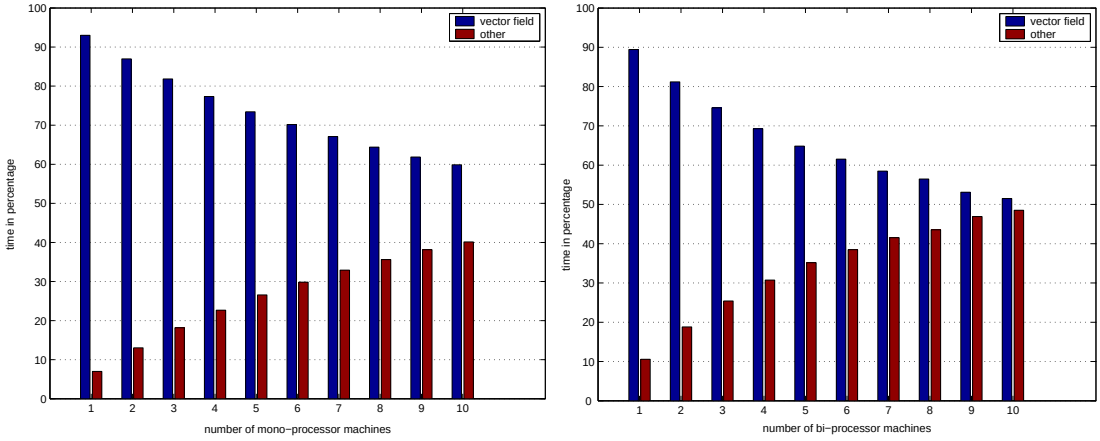

Figure 7: *Percentage of the time spent in the vector field computation and in the remaining sequential part, for 1 to 10 mono-processor machines (on the left), and 1 to 10 bi-processor machines (on the right).*

In Fig. 7, it is obvious that using bi-processor machines is much more efficient than the same number of mono-processor machines. From a pecuniary point of view, it is also rather obvious that bi-processor PCs are "cheaper" (currently, we can buy roughly 4 bi-processor central units for the same price as 5 mono-processor ones). To evaluate more precisely the gain of bi-processor machines (and incidentally the gain of OpenMP with respect to MPI), we need to compare the gain for the same number of processors.

5.2 Performance analysis

Figure 8 shows the speedup of the parallel implementation of our algorithm as the number of CPUs increases. The speedup is normalised by the execution time on a single CPU workstation. This experiment was done using 1 to 10 mono-processor machines (a single processor of each workstation has been used), and 1 to 10 bi-processor machines (the corresponding curve is interpolated for an odd number of processors).

To evaluate the quality of the performance, we also plot the theoretical speedup factors provided by Amdahl's law. This law simply states that, if $F\%$ of the code is sequential, and the remaining code is parallelised on N processors, the maximum speedup factor (with no communication delays) is:

$$S = \frac{1}{F + \frac{1-F}{N}}$$

Using the "constant time" values displayed in Figure 6, we estimated that $F \approx 7\%$ for the mono-processor case, and a slightly inferior value of 6% for the bi-processor case, thanks to the OpenMP parallelisation of the image resampling (Section 4.3).

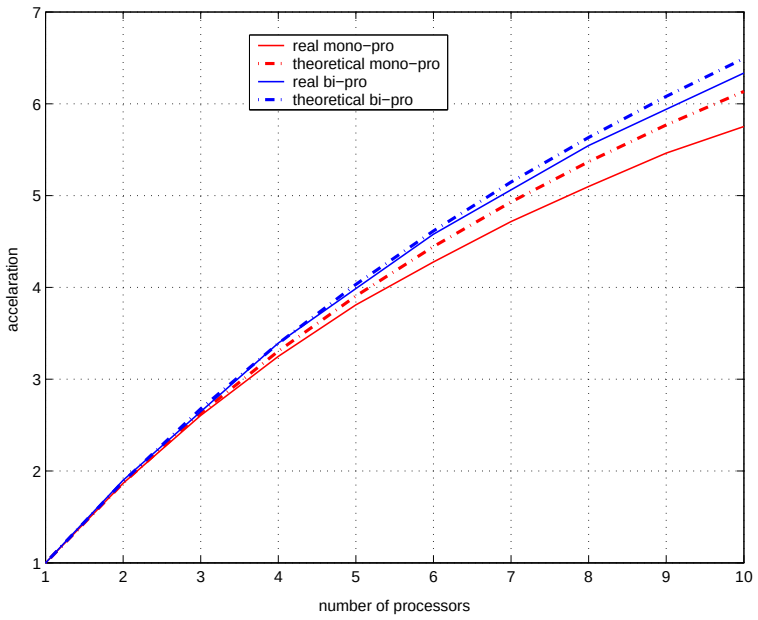

Figure 8: *Speedup of the implementation, normalised to the execution time on a 1 CPU Pentium III 933 Mhz. We can see the difference between theoretical and real acceleration due to the communication times.*

It is interesting to note that the Amdahl's law provides an upper limit on the speedup factor, in our case $\lim_{N\to\infty} S \approx 14.29$ for mono-processor workstations, and $\lim_{N\to\infty} S \approx 16.67$ for bi-processor ones. In practice, however, there is an overhead coming from communication times between the machines and the observed speedup will be less than predicted by Amdahl's law.

As expected, the measured speedup is higher for bi-processor machines (the sequential part takes a smaller time) but it is also relatively closer to its theoretical curve. This can be explained by a lower communication overhead thanks to the shared memory (OpenMP part). There are indeed two steps of the algorithm where the master has to communicate *simultaneously* with all the processes: for gathering the matchings from the slaves and for broadcasting the newly computed transformation. In such cases, the communication time increases with the total number of nodes, which is two times larger for a cluster of mono-processors than for a cluster of bi-processors.

As a conclusion, it is definitely cheaper, faster, and more efficient to use bi-processor machines. Now, the question is how many (bi-processor) machines do we need to register our images in less than a given time (typically 1 mn for IGT applications) ?

5.3 Influence of the data size

In the previous sections, we were only interested in the speedup factor, but it is clear that the computation time depends on the volume of the data being registered. We report in the following table the registration times (in seconds) for different cluster configurations and two main types of data. Moreover, we verified on other data that the computation time was directly proportional to the size of the data: we indicate below the mean computation time per million of voxels.

Data type	Sequential	One bi-pro	5 bi-pro.	10 bi-pro.
Vanderbilt (1.7 10^6 voxels)	118 s	63 s	19 s	14 s
Salpêtrière (8.1 10^6 voxels)	600 s	316 s	95 s	69 s
Time for 10^6 voxels	72 s	38 s	11.5 s	8.3 s

If the computation times are excellent for small image volumes, even with a reduced cluster of 1 bi-processor PC (here a Pentium III 933 Mhz), we need at least 10 machines to reach a computation time of about 1 mn for very large data volumes.

To highlight the importance of the image size we used here, we note that the IGT application presented in [GNK+01] used intermediate image size ($256^2 \times 60 = 3.9\ 10^6$ voxels), registered in about five minutes. Extrapolating our computation times to this image size would give respectively 2mn30, 45, and 33 seconds with 1, 5, and 10 bi-processor machines, which are all acceptable computation times in this application.

5.4 Trade-off between precision and computation time

Even if we finally obtained registration times of the order of one minute, which was our aim for IGT applications, one could think of applications where we need to register higher resolution images and/or reduce the number of machines. In this case, it seems difficult to further improve the computation times with parallelism without modifying the parallel algorithmic scheme (see Section 6). However, the pyramidal approach used in the algorithm provides a multi-scale transformation estimation. Moreover, the highest pyramid level takes about 73% of the total computation time (Figure 9). This suggests that we could drastically speed-up the registration by stopping the algorithm before it reaches the last pyramid level, at the cost of a lower precision on the transformation. The aim of this section is to estimate this loss in precision, and the corresponding speedup, with respect to the pyramid level.

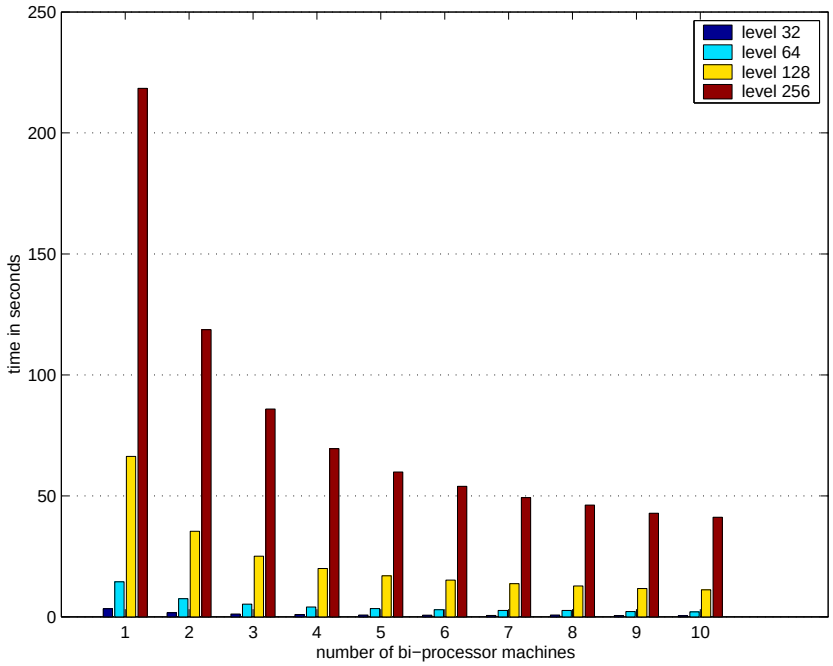

Figure 9: *Computation time of each pyramid level with 1 to 10 bi-processor workstations. One finds that the first level takes 1% of the total computation time, the second level 4%, the third level 22%, and the last level 73%.*

To estimate the loss in precision of the transformation with respect to the pyramid level, we used the transformation T_0 obtained with the high level as the reference (our "ground truth" for this experiment), and we want to compute the residual error on the transformation T_i at another level i. As a "difference of transformations" is difficult to interprete, we choose a set of representative points C_j (we took points at 1/4 and 3/4 of each maximal coordinate of the image), and we computed the mean localisation error for each transformation as follows:

$$RMS(T_i) = \sqrt{\frac{1}{n}\sum_j \|T_o.C_j - T_i.C_j\|^2}.$$

We report in the following table the relative precision of each pyramid level for the three high resolution datasets of pre- and post-operative MR images. We should note that there are some important deformations between the images due to the surgical operation and artifact (distortion due to air-brain interfaces, presence of electrodes, ...).

	Data set 1	Data set 2	Data set 3
Image dimension	$256^2 \times 124$ voxels	$256^2 \times 124$	$256^2 \times 128$
Voxel size	$0.94^2 \times 1.3$ mm	$0.90^2 \times 1.0$ mm	$0.98^2 \times 1.4$ mm
$RMS(T_3)$	3.54 mm	10.7 mm	8.96 mm
$RMS(T_2)$	2.80 mm	1.08 mm	2.60 mm
$RMS(T_1)$	**0.92** mm	**0.68** mm	**0.71** mm
$RMS(T_0)$	reference	reference	reference

In all cases, the relative precision of the transformation at level 1 is still below the voxel size. Moreover, we computed that it corresponds to a relative precision of 0.2 mm at the center of the brain. Thus, an optimal trade-off between precision and computation time seems to be obtained by stopping the algorithm at pyramid level 1. We display in Figure 10 the full and "level 1" resolution times for different numbers of bi-processor machines: we obtain a one minute registration with only two bi-processor machines. With low resolution images (the Vanderbilt database), we obtain even more impressive computation times: from 21 seconds for 1 (bi-pro) PC to 7 seconds for 5 (bi-pro) PCs.

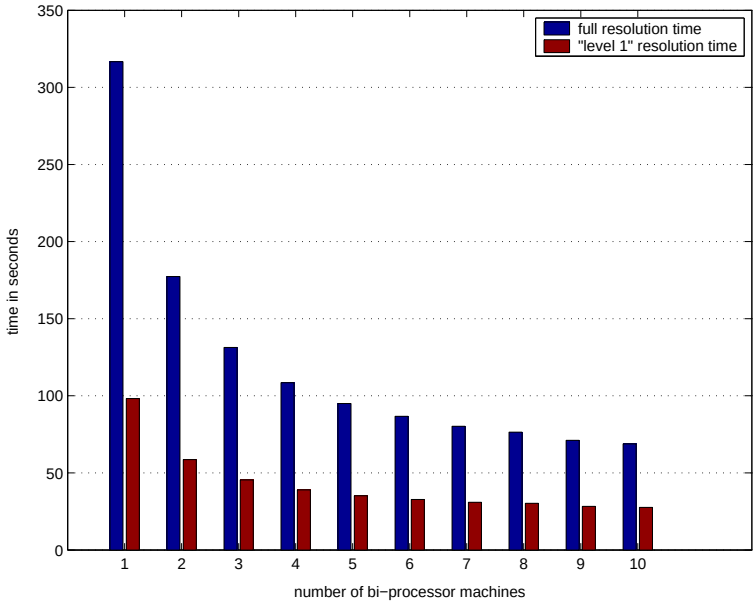

Figure 10: *Computation times for the full resolution (T_0) in blue and the "level 1" resolution (T_1) in red for 1 to 10 bi-processor workstations*

6 Discussion and conclusion

The registration algorithm we chose to parallelise computes at each step a sparse vector field by block matching, and uses that vector field to estimate a robust parametric (rigid to affine) transformation using Least-Trimmed-Squares. This estimation is embedded in a multi-scale framework. A profiling of the program showed that 93% of the time was spent in the vector field computation, the other 7% being spent in image resampling, LTS minimisation, and I/O operations. Thus, we proposed parallel MPI and OpenMP implementations of the vector field computation, with a resulting acceleration of approximately 48% on two processors. In practice, using OpenMP instead of MPI did not significantly reduce the computation time, but decreased the amount of necessary memory. We also proposed an OpenMP implementation of the resampling that provides a speedup of a bit less (40%), mainly due to memory access limitations.

The speedup results on more than two processors closely follow the theoretical bound given by Amdahl's law. For the same number of processors, the parallelisation

appears to be slightly more efficient for bi-processor machines (thanks to memory sharing in OpenMP). Since they are also relatively cheaper (roughly 4 bi-pro, i.e. 8 processors, for 5 mono-pro), we can conclude that a small cluster of bi-processor PCs is an optimal choice.

The computation times themselves are excellent on small images (19 seconds for 5 bi-processors on Vanderbilt images), and still very good on typical intra-operative and high resolution MR images: respectively 45 secondes and 1mn35 for 5 bi-pro, 33 and 70 seconds for 10 bi-pro workstations. For this last configuration, the computation time of the vector field equates the computation time of the sequential part. This means that we cannot gain much more by using additional machines.

The analysis of the multi-scale transformation estimation shows that one way to further accelerate the algorithm (or reduce the number of machines) is to slightly decrease its accuracy. Experiments showed that the relative precision of the transformation at level 1 is still below the voxel size, for only 27% of the total computation time. The registration of high resolution MRI now takes only 1 mn on a cluster of only 2 bi-processor PCs. With lower resolution images (the Vanderbilt database), we even obtain a registration time of 15 seconds on the same cluster configuration.

Another way to improve the computation times without reducing the precision of the algorithm is to modify the parallel algorithmic scheme. As stated in Section 5.2, the theoretical lower bound on the execution time is due to the sequential sections of the algorithm (in our case the resampling of the entire floating image). To reduce the computation time of this part, the main idea would be to resample the image only at necessary voxels within each process. As each process computes the matches for only a few blocks, it should only resample the part of the floating image which is within these blocks. However, this means that the selection of relevant blocks (see Section 3) cannot be done at each iteration as before, since some blocks may no longer overlap with the reference image. To solve this problem, the idea is to pre-compute an approximate variance of each bloc (since the variance depends on the block size, it has to be done at each pyramid level). Then, at each iteration, select the relevant blocs that entirely overlap the reference image, and resample the floating image only on these blocs, "on the fly". With such an approach, one could expect to resample only $20\%/Num_{proc}$ of the image per processor at the highest pyramid level, which could lead to a cumulative resampling time of less than one second on 5 bi-processor machines instead of 17 seconds for a high resolution image.

The results obtained in this article, both in terms of computation time for registration and methodology for parallelisation, open research avenues for the massive treatment of medical image databases, such as the quantification of disease evolution

on a large number of patients, or information retrieval and exploration in large image collections.

Acknowledgements

The authors would like to thank Janet Bertot for proofreading.

References

[AVV90] L.M. Auer and V. Van Velthoven. *Intraoperative Ultrasound Imaging in Neurosurgery*. Springer-Verlag, 1990.

[BYT+97] R.D. Bucholz, D. Yah, J. Trobaugh, L. McDurmont, C. Sturm, C. Baumann, J.M. Henderson, A. Levy, and P. Kessman. The correction of stereotactic inaccuracy caused by brain shift using an intraoperative ultrasound device. In *Proc of CVRMed-MRCAS'97*, volume 1205 of *LNCS*, pages 459–466, Grenoble, France, 1997.

[CJ99] R.W. Cox and A. Jesmanowicz. Real-time 3d image registration for functional mri. *Magnetic Resonance in Medecine*, 42:1014–1018, 1999.

[CRM96] G. E. Christensen, R. D. Rabbitt, and M. I. Miller. Deformable templates using large deformation kinematics. *IEEE Transactions on Medical Imaging*, 5(10):1435–1447, October 1996.

[Ger99] D. Gering. System for surgical planning and guidance using image fusion and interventional mr. Master's thesis, MIT, 1999. http://www.slicer.org.

[GNK+99] D.T. Gering, A. Nabavi, R. Kikinis, E.L. Grimson, N. Hata, P. Everett, F.A. Jolesz, and W.M. Wells. An integrated visualization system for surgical planning and guidance using image fusion and an interventional imaging. In *Proc. MICCAI'99*, volume 1679 of *LNCS*, pages 809–819, Cambridge (UK), October 1999.

[GNK+01] D.T. Gering, A. Nabavi, R. Kikinis, N. Hata, L.J. O'Donnell, E.L. Grimson, F.A. Jolesz, P.M. Black, and W.M. Wells. An integrated visualization system for surgical planning and guidance using image fusion and an open mr. *Journal of Magnetic Resonnance Imaging*, 13:967–975, 2001.

[GSS+96] D.H. Gronemeyer, R.M. Seibel, A. Schmidt, A. Melzer, and M. Dell. Two- and three-dimensional imaging for interventional mri and ct guidance. *Stud Health Technol Inform*, 29:62–76, 1996.

[Jol97] F. Jolesz. Image-guided procedures and the operating room of the future. *Radiology*, 204:601–612, 1997.

[Kik00a] R. Kikinis. Hpc in image-guided surgery. Technical Report 193, Surgical Planmig Laboratory, 2000. Oral Presentation at PDC Annual Conference, Center for Parallel Computers (PDC), Royal Institute of Technology (KTH), Stockholm, Sweden. December 14-15, 2000.

[Kik00b] R. Kikinis. Igt: Today and tomorrow. Technical Report 192, Surgical Planmig Laboratory, 2000. Oral Presentation at the Washington CAS Group Meeting.

[KMM+01] D.F. Kasher, S.E. Maier, H. Mamata, Y. Mamata, A. Nabavi, and F.A. Jolesz. Motion robust imaging for continuous intraoperative mri. *Journal of Magnetic Resonnance Imaging*, 13:158–161, 2001.

[LPA84] L.D. Lunsford, R. Parrish, and L. Albright. Intraoperative imaging with a therapeutic computed tomographic scanner. *Neurosurgery*, 15:559–561, 1984.

[MPI95] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, May 1995. http://www.mpi-forum.org/.

[NVMW01] T. Netsch, A. Van Muiswinkel, and J. Weese. Towards real-time multimodality 3-d medical image registration. In *Eight International Conference on Computer vision*, volume 1, pages 718–725, July 9–12 2001.

[Ope98] OpenMP Architecure Review Board. *OpenMP C and C++ Application Program Interface Version 1.0*, October 1998.

[ORPA00] S. Ourselin, A. Roche, S. Prima, and N. Ayache. Block Matching: A General Gramework to Improve Robustness of Rigid Registration of Medical Images. In A.M. DiGioia and S. Delp, editors, *Third International Conference on Medical Image Computing And Computer-Assisted Intervention (MICCAI'00)*, pages 557–566, Pittsburgh, Pennsylvania USA, October 11-14 2000.

[ORS+01] S. Ourselin, A. Roche, G. Subsol, X. Pennec, and N. Ayache. Reconstructing a 3D Structure from Serial Histological Sections. *Image and Vision Computing*, 19(1-2):25–31, January 2001.

[PCA01] X. Pennec, P. Cachier, and N. Ayache. Tracking brain deformations in time sequences of 3D US images. In M.I. Insana and R.M. Leahy, editors, *Proc. of IPMI'01*, number 2082 in LNCS, pages 169–175, Davis, CA, USA, June 2001. Springer Verlag.

[POA00] S. Prima, S. Ourselin, and N. Ayache. Computation of the Mid-Sagittal Plane in 3D Images of the Brain. In D. Vernon, editor, *Sixth European Conference on Computer Vision, ECCV'2000*, volume 1842-3 of *Lecture Notes in Computer Science*, pages 685–701, Dublin, Ireland, June/July 2000. Springer. Electronic version: http://www.inria.fr/RRRT/RR-3841.html.

[RL87] Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection*. Wiley Series in Probability and Mathematical Statistics, first edition, 1987.

[SJR95] J. Schenk, F. Jolesz, and P. Roemer. Superconducting open-configuration mr imaging system for image-guided therapy. *Radiology*, 195:805–814, 1995.

[Wa97] J. West and al. Comparison and evaluation of retrospective intermodality brain image registration techniques. *Journal of Comp. Assist. Tomography*, 21:554–566, 1997.

[WFG+00] S.K. Warfield, M. Ferrant, X. Gallez, A. Nabavi, F.A. Jolesz, and R. Kikinis. Real-time biomechanical simulation of volumetric brain deformation for image guided neurosurgery. In *SC 2000: High Performance Networking and Computing Conference*, volume 230, pages 1–16, Dallas, USA, Nov 4–10 2000. SPL Technical Report 188.

[WRN+99] J. Weese, P. Rösch, T. Netsch, T. Blaffert, and M. Quist. Gray-value based registration of ct and mr images by maximization of local correlation. In *Second International Conference on Medical Image Computing And Computer-Assisted Intervention (MICCAI'99)*, volume 1679 of *LNCS*, pages 656–663, Cambridge (UK), October 1999.

[WVA+96] M.W. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis. Multi-modal volume registration by maximization of mutual information. *Medical Image Analysis*, 1(1):35–51, 1996.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399