Distribution of Image Registration Algorithms

Thorben Flämig

July 31, 2005

Abstract

Image Registration is an important field in nuclear medicine and other areas. It is an active field in research in universities as well as in clinics. Automatic registration can help in diagnosis of multiple diseases and be an important part in aftercare. By distributing a voxel property based registration algorithm fast and flexible automatic image registration can be achieved. To implement such a distributed algorithm is the aim of this work.

1 Introduction

Image Registration plays a more and more important role in medicine. Devices for medical imaging are developing fast and the combination of different image modalities like Positron Emission Tomography (PET) and Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) are spreading in research and medical everyday life.
Also the comparison of images taken with the same device at different states of a disease pattern is used more often to control progress in the treatment of patients. This is the motivation for this work.
First of all principles of Image Registration are discussed. Then the specific algorithm that was used in this work is described more precisely. Using examples from the implementation, the distribution of processes using the message passing interface will be explained. Then the results of testing the implementation are shown. In the conclusion finally the achieved results are discussed and the future prospects are presented.

2 Image Registration

Medical imaging plays an important role not only in clinical diagnosis but also in planning, execution and evaluation of surgical and radiotherapeutical treatment. There are two classes of image modalities, anatomical and functional imaging.

Examples for anatomical imaging are MRI and CT. In functional imaging methods like Single Photon Emission Computed Tomography (SPECT) or Positron Emission Tomography (PET) are used. As there is such a diversity of imaging techniques and uses there exist many different approaches to image registration. The registration process is defined as bringing the images into spatial alignment. It can be either applied on two images of the same modality, the so-called monomodal registration, or multimodal registration which can register two images of different modalities like PET and CT.
In both monomodal and multimodal registration there are multiple application areas.
Different approaches to image registration are possible. They differ in the nature of the registration basis, the nature of transformation and many other factors. Concerning the nature of the registration basis a first categorisation can be made in extrinsic, intrinsic and non-image based registration.
Extrinsic registration is based on information added to the imaged space, recognisable objects that can be easily recognised in the picture. Intrinsic registration is based on patient generated image content only. Non-image based registration is a rather inconvinient strategy as the registration relies on the calibration of the coordinate systems of the two imaging devices. In these categories there are further subdivisions.

Extrinsic registration can be divided in the invasive and non-invasive methods. These again can be split, the invasive methods are parted by the use of a stereotactic frame or fiducials (screw markers), the non-invasive are either using a mould, frame, dental adapter or something comparable, or fiducials (skin markers).

Intrinsic approaches are landmark-based, segmentation based or voxel property based. The landmarks can be anatomical landmarks or geometrical landmarks. The segmentation can be performed using rigid models like points, curves or surfaces, or by using deformable models like snakes or nets.
The voxel property based registration is divided in two categories. The reduction to scalars/vectors (moments, principal axes) can restrict the complexity. By using the full image content the costs for the computation can be very high but the results can be very reliable. This approach will be followed in this work.

The definitions and categorisations are emanate from "A Survey of Medical Image Registration" by J. B. Antoine Maintz and Max A. Viergever [1].

The chair for computer-aided medical procedures at the Technische Universität München is doing research in this field. Wolfgang Wein has implemented a sequential version of an image registration for different modalities [2]. Pixel/voxel intensities are used to compute a similarity measure. With the best-neighbor optimization strategy we run through the images whereas the first image is fixed and the other is moving in its six degrees of freedom. This is done until the similarity measure reaches an optimum. This stable position should be the correct registration of the two images. The workflow is illustrated in Fig. 1.

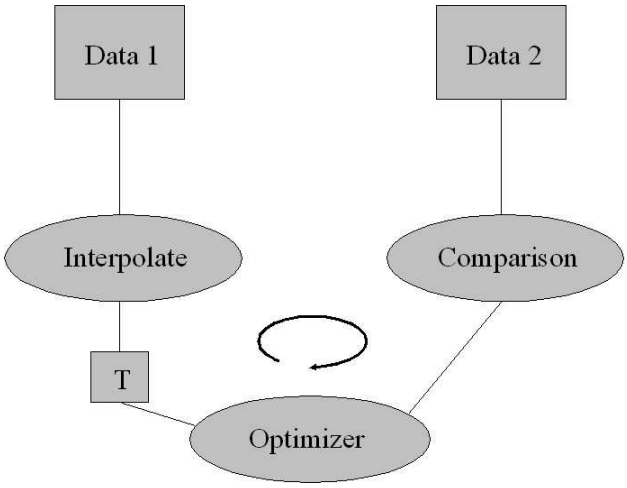

Figure 1: Data 1 is the moving image, Data 2 the fixed image. Starting with an initial transformation T the moving image is interpolated and the two images are compared. These values are used in the optimizer to refine the transformation T. This loop of interpolation, comparison and optimization runs until the values fall below a threshold and the registration terminates.

Another approach could be a method using Particle Swarm Optimization [4], which is a very interesting approach for global registration. Starting with a first initalisation given by the user for a set of positions, the particle swarm, a similarity measure is computed. Then the next position for each particle is chosen by adding a velocity vector. This vector is computed using the best value the particle has found itself and the best value of the whole swarm. So the swarm tends to gather around the optimum. It reduces the risk of getting stuck in local optima because not just the direct neighborhood of the particle is used to find the next position but also the best position of all others. The grade of parallelisation is also not restricted because the number of particles used is not limited by any means. But the results show that the improvements over the evolutionary strategy are not yet persuasive.

In nonlinear image registration for example one approach in many is using a Finite Element Method [3]. The Finite Element Method is exact and fast for monomodal images and similar images. With highly different images like multimodal images usually are it loses accuracy. Thus it would not be so interesting in this field of application.

As the sequential implementation of Wolfgang Wein is producing stable registrations with different image modalities and also different image dimensions we are following this approach. But the registration of two huge volumes generated by medical imaging devices needs a lot of computational work. To make it practical in clinical use the question if parallel computation could shorten this process is raised.

3 Parallel Computation Approaches

There are different approaches of parallel computing. The most common are shared memory and message passing.

3.1 Shared Memory

In shared memory the program is divided into several processes. These run on the same multi-processor machine and share one adress space. So they can work on the same data. This saves memory space because you do not have to load the whole data for each process. But on the other hand conflicts may emerge because two processes try to modify the same piece of data. Synchronization can handle this so the critical areas are restricted and can be entered by just one process at a time. Disadvantage of synchronization is that these parts are sequential again and the speedup of parallel processing can not be reached in synchronized areas.

3.2 Message Passing

Message passing can be used either with multi-processor machines or a cluster of several independent computers. Different parts of the program are computed on different processors or on another computer. They communicate with each other and exchange results or working data by sending messages to each other. So it is the more general approach to distributed computation as no special features of a computer architecture are specifically used and the implementation can be platform independent. Usually the overhead with message passing is bigger but with own adress spaces for the different processes some memory conflicts can be avoided.

3.3 Decision to use MPI

In this work we chose message passing because the data that has to be exchanged is very small in comparison to the amount of work that has to be computed for each position. So we are not limited to multi-processor machines or a special computer architecture. And with the image data available for every client no reading conflicts should arise.

This could be also very useful because probably clusters of workstations or personal computers will be available in clinical use while multi-processor machines with up to twelve processors are not so widespread. Using the MPI standard makes the program code as platform-independent as possible. Without major

modifications it should compile and run on every architecture that provides an implementation of MPI.

4 MPI Message Passing Interface

The Message Passing Interface MPI is a standard for message passing programs defined by the mpi-forum [5]. It is a library specification for message passing. A number of functions is specified, their behavior and the way they have to be called. As stated it is just a specification and there are multiple implementations. In this project the implementation used is MVAPICH [6].

MVAPICH, spoken *em-vah-pich*, MPI for InfiniBand on VAPI Layer, where VAPI stands for Verbs Level Interface, is an implementation from Ohio State University, which is a patch for MVICH, MPI for Virtual Interface Architecture, from National Energy Research Supercomputer Center NERSC, that is based on MPICH 1.2.5 from Argonne National Laboratory.

This is the MPI implementation that is provided on the infiniband cluster of the LRR, Fig. 2. It provides the user with a number of functions and status information to communicate between processes [7].

Figure 2: Picture of the Infiniband Cluster of the LRR Chair, Technische Universität München

All processes run with the same program code. To use this interface the header file *mpi++.h* needs to be included. To get started with the distributed computation the *Init* function is called. *MPI::COMM_WORLD* is the so-called standard communicator. Now the user can get information about the status of the available nodes and the rank of the process which is currently working through the code. With this information the user is able to separate the work on different processes. Communication is possible with multiple send- and receive-commands, blocking and non-blocking [8].

Example for the communication:

```
int size = MPI::COMM_WORLD.Get_size();
int rank = MPI::COMM_WORLD.Get_rank();
```

The procedure *Get_size* returns the number of processes that are launched and are available for the computation.
Get_rank returns the so-called rank of the process. This is a number between zero and size-1 and enables the programmer to distinguish between the simultaneously executed processes. Using this flag he can decide if a process shall execute a certain part of the program code.

```
if (rank == 0) {
// sending termination messages to all clients
      for (int s = 1; s < size; s++){
        float pose[6];
        int to = s;
        int count = 6;
        int tag = 1024;
        for (int j = 0; j < 6; j++) pose[j] = -1;
             MPI::COMM_WORLD.Send(pose, count, MPI::FLOAT, to, tag);
        }
}
```

This part of the program shall be executed by the process with *rank* zero which will be the server during the computation.
In the variable *to* the destination is saved. By using a loop running from one to *size* every client gets this message.
Setting *tag* to 1024 is the termination flag. Thus the client can recognise that the computation has reached the result and terminate.
The standard format of the send command can be found in [8]:
MPI::COMM_WORLD.Send(buf, count, datatype, dest, tag). Here *buf* is a pointer to the data that has to be sent. The *count* is an Integer that gives the length of the buffer that is sent. In *datatype* as it says the datatype that shall be sent is declared. *dest* means the rank of the process that shall receive the data. And *tag* is again an Integer that can be used to give some identification to the message.

```
if (rank != 0) {
        tag = MPI::ANY_TAG;
        count = 6;
        from = 0;
        MPI::Status status;
```

```
float pose[6];
MPI::COMM_WORLD.Recv(pose, count, MPI::FLOAT, from, tag, status);
int st_tag = status.Get_tag();
if(st_tag == 1024){
       return;
}
```

All clients shall execute this part of the program. The condition *rank!=0* excludes the server with rank zero which shall not execute the clients' code but makes sure that all clients can enter this part.
As the client does not know the value that will be used as tag the term *MPI::ANY_TAG* accepts any possible message tag.
In *count* the expected length of the data is defined. In *from* the expected sender of the message that will be received is defined as the server. In *status* information about the received message will be saved. In the floating point array the received values will be stored. The tag of the message is stored in *st_tag* and will be used to identify the result.
If the received message tag is *1024* the message is recognised as termination message and the client exits. Otherwise the computation with the received data will start.

Thus it is very comfortable for the developer to use several processors or computers for his programs.
When starting a program with the *mpirun* command the user decides which and how many processes he wants to use. This is done by giving the total number of processes used and the names of the nodes to the program in the console. Nodes with multiple processors can be repeatedly referred to. So it does not make a difference to the implementation if several different computers are used to run the program or a multi-processor machine executes the whole program. They can even be mixed without problems.
By secure shell the nodes are acessed so theoretically they do not have to be directly connected by ethernet or infiniband like it is the case here, they can even be connected just over the internet. If this makes sense in reality has to be tested, but if the part of the communication is small and the problem needs a lot of computational power to solve the subproblems it may be an interesting opportunity to occupy unused resources. With the information that MPI gives the programmer about the number of processes used he can easily adapt to the really available processes.

5 The Optimization Algorithm

As optimization algorithm the Best-Neighbor Optimizer is used, as already stated. The principle is very easy, and it works very stable so far. A visualization can

be found in Fig. 3. Picture A shows the first step of the algorithm, starting at the pixel in the middle all 2n neigbors are evaluated. If one value is better than the current value this pixel is set as the next location that has to be evaluated. Here this is illustrated by the grey line from the starting location to the best neighbor. This can also be seen in Picture B. The yellow pixels are the neighbors that are evaluated. Again the grey line illustrates the way the optimizer takes to the next location. If no better value is found, as shown with the red pixels in Picture C, the stepsize is halved. Then the neighborhood is evaluated again. The blue pixels are the neighborhood that is evaluated after halving the stepsize. The algorithm terminates when either the stepsize or the difference in the cost functions falls below a threshold.

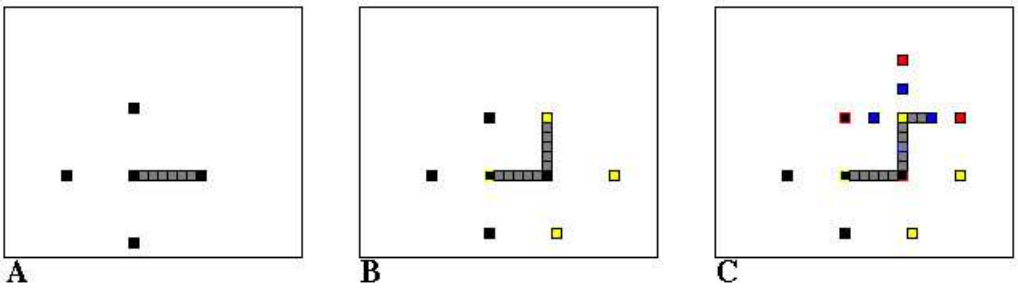

Figure 3: A visualisation of the Optimization Best Neighbor Algorithm.

This optimization algorithm has been chosen for this work because of its stability and the intuitive possibility to distribute the computation. For each iteration twelve positions have to be tested regarding six degrees of freedom . The twelve positions that have to be tested do not depend on each other and can be computed in parallel.

The approach utilising Particle Swarm Optimization [4] would be more powerful as a global registration method and is not limited in its degree of parallelism. So there surely is a field to be researched in the future.

6 The Parallel Registration

6.1 Distribution Strategy

In Fig. 4 A the point where the distribution steps in can be seen. The loop of interpolation, comparison and optimization taken from Fig. 1 can be distributed as for each transformation T there are 2n independent positions with n being the degrees of freedom. Thus these values can be evaluated in parallel as indicated in Fig. 4 B.

For the parallel evaluation the work must be divided. The server sends the actual poses, each an array of six floating points containing the coordinates of the

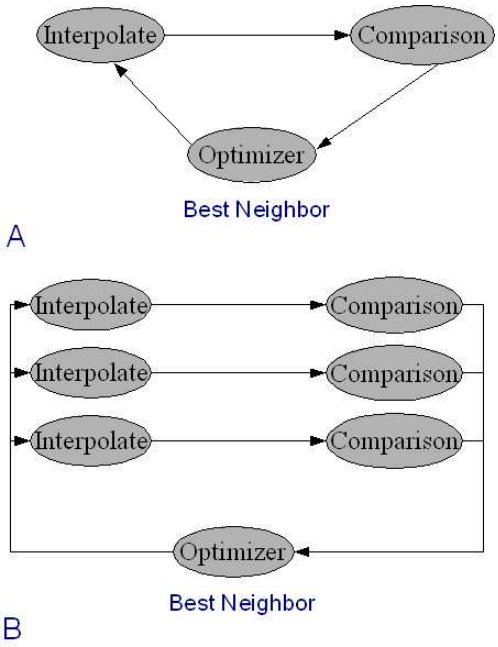

Figure 4: In A the point of attack is shown. This loop of interpolation, comparison and optimization taken from Fig. 1 is divided into independent parts as indicated in B. For each transformation T there are 2n independent positions with n being the degrees of freedom. These positions can be evaluated in parallel.

position to be evaluated to the clients which are waiting to receive their values to evaluate. Additionally in the message tag an Integer is sent that either identifies the result with the value *1024* as termination flag or is the position in the array of result values in all other cases. The clients start computing the similarity measure and send their result back to the server which can evaluate one value on its own during the waiting period.
Communication is done by the blocking send and receive command to make sure that each value reaches its worker.

6.2 Implementation

As shown in Fig. 5 the class *RegistrationMPI* is included into the class diagram as subclass of *Registration* in a decorator pattern. Like this the implementation is very flexible, as the instance of the class *Registration* that is used as *m_reg* can be of all subclasses of *Registration* like *Registration2D2D*, *Registration2D3D* or *Registration3D3D*. So the field of application is wide and the user can register images in different dimensions without adapting to another program.

Like described in the section about MPI the program has to be started with the *mpirun* command. The implementation is structured that in the main pro-

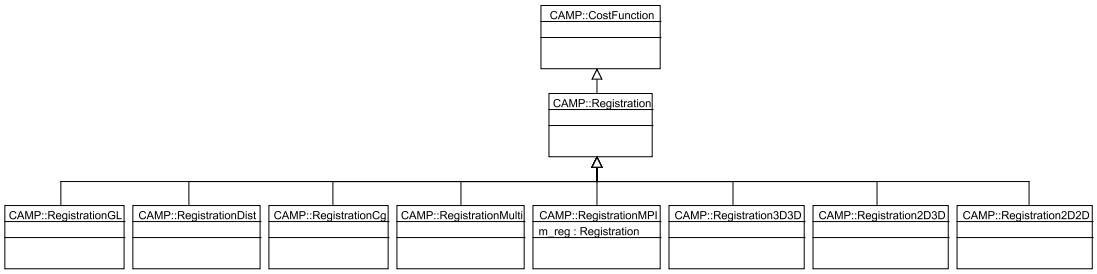

Figure 5: The class diagram in which the distributed registration is included.

gram the *init* function is called. Then every process instantiates an instance of the class *RegistrationMPI*. Now the *run* method is called on this instance. So far all processes are doing the same. In the *run* method the case distinction between server and clients is made. This is done with an *if* statement that makes the process with rank zero to be the server and all others will execute the client code. The clients start a loop waiting for data to compute or the termination signal. The evaluation of the position that is sent to a client is done by calling the *evaluate* method which evaluates one value. It is the same method that is called several times in the sequential registration. The only difference now is that the values are computed at the same time by different clients.
The server calls the *run* method of the superclass *Registration*. This starts the optimization. When the method *evaluateParallel* instead of the sequential *evaluate* requests n results in each call, this enables the parallel computation of n values at a time.

If less than 12 processes are available the implementation adapts to the situation and distributes the work as good as possible. The clients get their jobs according to the round-robin method. As the jobs should be computed in the same time this brings a satisfying spreading of the work in every case.
For the server a case distinction decides how many values are computed on their own. With more than twelve nodes all values are sent to clients and the server confines itself to communication and checking the results. With less clients it computes more and more values on its own to keep the workload properly partitioned. Thus the available working power is used optimally.

After each of these steps the server compares the results. If one value is better than the actual position the position is changed to this better value and the next positions are sent out. Else the stepsize is halved and again the next positions are sent out.
This procedure goes on until the stepsize falls below a threshold, as seen in Fig. 3. Then the server sends out a termination message to the clients and the program completes its work by writing the final position and some additional information in a file.

7 Results

Extensive tests have been run on the Infiniband Cluster of the LRR chair of the Technische Universität München. On four Itanium2 nodes the testing was done. Each of the nodes has four Intel Itanium2 processors at 1.3 GHz with 8 GB main memory on every node. The processor system bus: 128 bit, 400 MHz data rate and the memory bus: 256 bit, 200 MHz data rate.

The volumes that were used in the tests are a CT scan of a male thorax with resolution of 512x512x449 and a CT scan of the heart of the same male, the resolution is 512x512x229. A view of the volumes can be seen in Fig. 6.

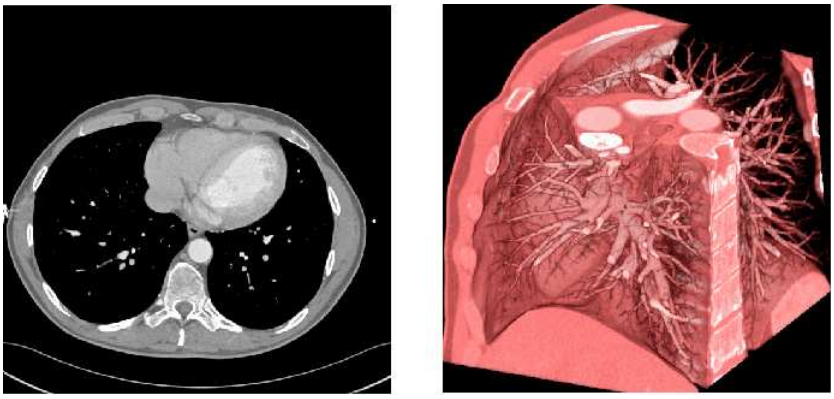

Figure 6: The images with which the tests were done, a CT scan of a male thorax and a CT scan of the heart of the same patient.

The performance is convincing. The values are exactly the same as with the sequential registration, which should be obvious as the same values are computed. The speedup is as good as we hoped, with twelve nodes the computation needs 567.3 seconds to register the two volumes. The sequential registration computes 6528.0 seconds. So the distributed registration needs just about 8.7% of the time the sequential registration needs as you can see in Fig. 7. The runtime with the two example volumes is illustrated, time is in seconds. For comparison the runtime of the sequential registration is shown first. By adding processes the runtime decreases clearly. The fact that it does not change between for example four and five processes is due to the condition that the twelve positions to be evaluated cannot be divided better and so there is always one process waiting for the others to complete their work. The same problem is from six to eleven processes where some processes have to evaluate two positions, and others just one. With twelve positions the workload can be distributed properly so each process evaluates one position. So the maximum speedup can be reached by running the distributed registration with twelve processes. With twelve and thirteen processes the runtime is nearly the same, there is no measurable improvement if the server does nothing but communication.

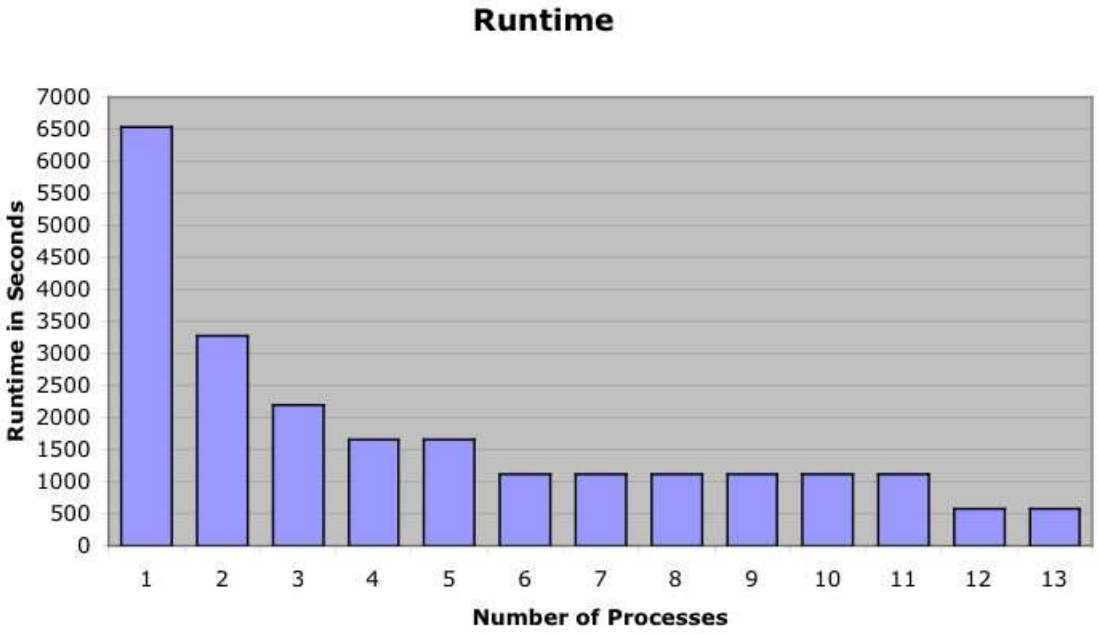

Figure 7: The runtime of the tested registration on the Itanium nodes.

As shown in Fig. 8 the spreading of processes on different nodes or running as many processes as possible on one of the Itanium2 nodes does not make a great difference in runtime. One process on each of the four Itanium2 nodes with four processors has nearly the same runtime as running four processes on one node. Just the arrangement of two processes on two nodes each has a small advantage to the others. This may be due to incalculable operating system interference. The capacity of the memory bus is big enough to handle four processes computing on the data on one node.
So in clinical use it would be recommendable to acquire a small cluster with multi-processor computers so space and costs can be kept as small as possible but the maximum in speed can be achieved.

The time consumed by communication is negligible as can be seen in Fig. 9. One process computing all the values on its own without any communication needs pretty much the same time, even a few seconds more, as two processes of which just the client computes all similarity measures and the server is just doing communication. The difference between the values is due to interferences by the operating system.

The Infiniband Cluster has been expanded and now has another 36 AMD Opteron nodes with four processors each. With a clock rate of 2.4 GHz they are a lot faster than the Itanium nodes on which most of the testing has been done. In first benchmarks the distributed registration was 2.9 times faster than on the Itanium nodes. With twelve active processes the registration of the example images terminated after 196 seconds. This shows that an acceptable registration

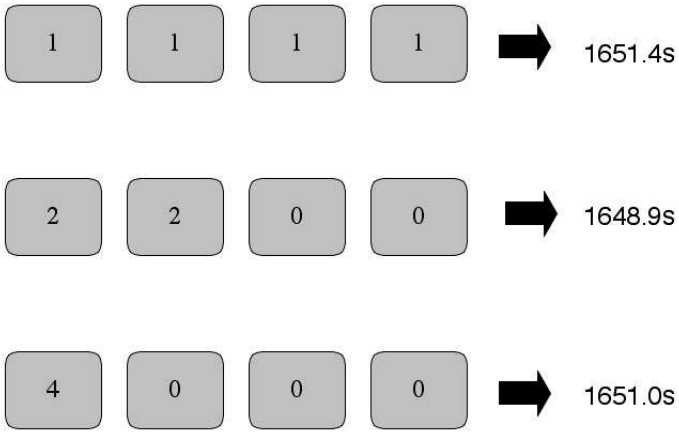

Figure 8: Different spreading of processes does not influence the runtime.

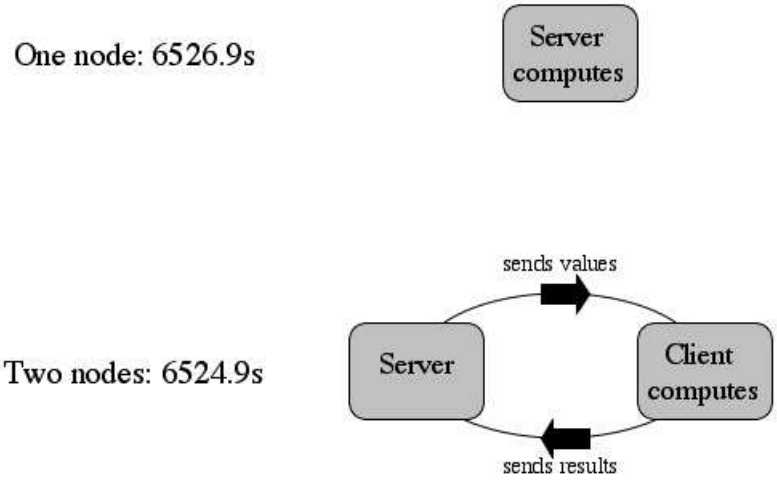

Figure 9: The time lost because of process communication is negligible.

time is reachable with this implementation.

8 Conclusion

First of all the goal of distributing the sequential image registration algorithm is reached.
As the implementation is very flexible it can be used for the registration of many possible imaging problems like 2D/3D and 3D/3D. The image modality can also be very different because with the pixel/voxel intensity based similarity measure computed tomography, magnet resonance tomography, positrone emission tomography and all other medical image data produce comparable results. Even

non-medical data can be registered.
There is no further information about the images needed. The implementation can be used by medical laities without any knowledge what the images show and how they were produced.

By using the message passing interface MPI the implementation can be easily introduced in clinics. Platform independent as it is it can be used on clusters of workstations, on Linux Clusters or multi-processor machines and also personal computers using Microsoft Windows or Apple OS can be used for the computation. Only a running implementation of MPI is needed. This is freely downloadable on the internet [1].

So with the tremendous speedup that is reached with the distribution of the already used sequential implementation it should be possible to spread the technique of automatic image registration in the clinics. The easy handling and the quick results hopefully convince more and more doctors of the usefulness of automatic image registration in diagnosis and aftercare.

The degree of twelve parallel computations with the actual algorithm is the maximum that is reachable. For more nodes and thus a greater speedup the algorithm has to be changed. Possibilities therefore could be to increase the stepsize and the number of poses that are tested for each step.

[1] *MVAPICH Download of the Ohio State University*,
http://nowlab.cis.ohio-state.edu/projects/mpi-iba/download.html

References

[1] *A Survey of Medical Image Registration*, J. B. Antoine Maintz and Max A. Viergever, Image Sciences Institute, Utrecht University Hospital, Utrecht, the Netherlands, 1997

[2] *Intensity Based Rigid 2D-3D Registration Algorithms for Radiation Therapy*, diploma thesis, Wolfgang Wein, Technische Universität München, 2003

[3] *Non-Linear Registration using a Finite Element Method*, M. Valdiviesso-Casique and S. Arridge, Department of Computer Science University College London, London, 1999

[4] *An Approach to Multimodal Biomedical Image Registration Utilizing Particle Swarm Optimization*, Wachowiak, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 8, NO. 3, JUNE 2004

[5] *The MPI Forum*, http://www.mpi-forum.org/

[6] *MPI over InfiniBand Project, Ohio State university*, http://nowlab.cis.ohio-state.edu/projects/mpi-iba/

[7] *An index of the generally available functions provided by the Message Passing Interface*, http://www.mpi-forum.org/docs/mpi-11-html/node182.html

[8] *MPI Tutorial*, University of Notre Dame, 1998, http://www.lam-mpi.org/tutorials/nd/