Real-Time Image Guided Neurosurgery Using Distributed and Grid Computing

Nikos Chrisochoides, Andriy Fedorov, Andriy Kot*
Neculai Archip, Peter Black, Olivier Clatz, Alexandra Golby, Ron Kikinis, Simon K. Warfield†

Abstract

Neurosurgical resection is a therapeutic intervention in the treatment of brain tumors. Precision of the resection can be improved by utilizing Magnetic Resonance Imaging (MRI) as an aid in decision making during Image Guided Neurosurgery (IGNS). Image registration adjusts pre-operative data according to intra-operative tissue deformation. Some of the approaches increase the registration accuracy by tracking image landmarks through the whole brain volume. High computational cost used to render these techniques inappropriate for clinical applications.

In this paper we present a parallel implementation of a state of the art registration method, and a number of needed incremental improvements. Overall, we reduced the response time for registration of an average dataset from about an hour and for some cases more than an hour to less than seven minutes, which is within the time constraints imposed by neurosurgeons. For the first time in clinical practice we demonstrated, that with the help of distributed computing non-rigid MRI registration based on volume tracking can be computed intra-operatively.

Keywords: distributed computing, grid computing, medical imaging, image registration

*nikos@cs.wm.edu, fedorov@cs.wm.edu, kot@cs.wm.edu, Department of Computer Science College of William and Mary, Williamsburg, VA 23187, USA

†narchip@bwh.harvard.edu, pblack@partners.org, oclatz@bwh.harvard.edu, agolby@partners.org, kikinis@bwh.harvard.edu, warfield@bwh.harvard.edu, Brigham and Women's Hospital, Harvard Medical School, Boston, MA 02115, USA

SC2006 November 2006, Tampa, Florida, USA

1 Introduction

Cancer is one of the leading causes of death in USA and around the world. The American Cancer Society estimated that 12,700 people were expected to die from brain and nervous system cancer alone, in 2005 [American Cancer Society 2005]. Medical imaging, and Magnetic Resonance imaging (MRI) in particular, provide great help in diagnosing the disease, and, when necessary, facilitate minimally invasive brain tumor resection. The determination of the optimal limits during the surgical resection of intrinsic brain tumors is particularly challenging due to the difficulty of distinguishing tumor from essential brain by visual inspection. The introduction of intraoperative imaging aided by near real-time deformation, or *registration*, of preoperative datasets can allow the surgeon to base their intraoperative decisions on quantitative analysis in addition to his/her experience and intuition.

The primary need for the described research is motivated by intraoperative deformation of the brain tissue as the tumor resection progresses. Figure 1 shows that such deformation can be quite significant, thus preoperative imaging data becomes invalid. It has been shown that intra-operative Magnetic Resonance Imaging (MRI) can capture brain deformation and can be effectively used to guide the registration process [Ferrant 2001; Rexilius 2001; Warfield et al. 2002; Clatz et al. 2005]. Nowadays, Magnetic Resonance Therapy (MRT) systems are becoming more and more ubiquitous. In this paper we focus on the computational challenges of a displacement-based approach to registration of preoperative MRI to intraoperative data, as presented in [Clatz et al. 2005]. This method uses landmark tracking across the entire image volume, thus makes the non-rigid registration more accurate, but computationally expensive as compared to similar methods which are using surface tracking [Ferrant 2001]. Initial intra-operative experiments at Brigham and Women's Hospital (BWH) and at the College of William and Mary (W&M) using a 4-processor cluster identified two problems with the parallel implementation of the non-rigid registration method described in [Clatz et al. 2005]:

1. The execution time of registration is very high, and varies between 40 minutes to more than an hour (depending on the input), for a single intra-operative MR scan;
2. The scalability of the original code is poor due to computationally expensive sequential processing components.

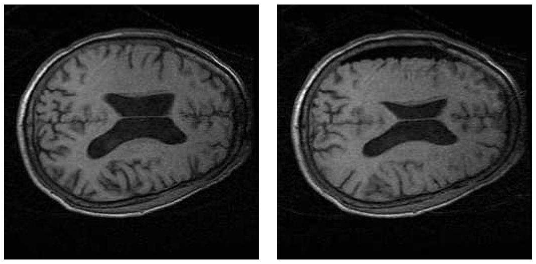
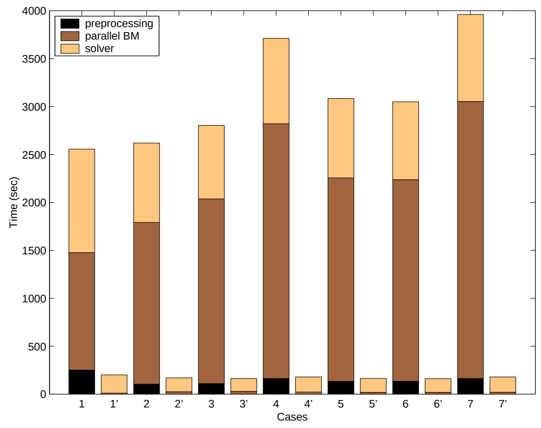


Figure 1: Same MRI volume slice before (left) and after (right) tumor resection.

The execution time of the registration application on high performance platform at Brigham and Women's Hospital (*BWH-HPC*, see Section 5) exceeds by far the time surgeon can wait to proceed with the tumor resection. During neurosurgery a whole brain MRI acquisition can be performed within 4 minutes. We would like the non-rigid registration of the preoperative data into the intra-operative configuration to take no more than an intra-operative MRI would take, and the less time it takes the better. There are two ways to achieve this objective: (1) develop equally accurate, but less computationally demanding models, or (2) use scalable and more efficient implementations of the methods we have. In this paper we explore the latter approach.

The registration method in [Clatz et al. 2005] formulates brain shift as a functional minimization problem using Partial Differential Equations (PDEs) whose solution is approximated by Finite Element Methods (FEM). In the specific model the functional can be decomposed into a similarity energy and a regularization energy. The similarity energy (i.e., boundary conditions of the PDE) is computed by a block (or landmark/feature) matching algorithm [Clatz et al. 2005]. This is the most computationally intensive step of the registration process. Figure 2 depicts the break-down of the registration time for seven different data sets on *W&M-HPC* (see Section 5) using the original PVM [Belguelin et al. 1993] implementation. In the same Figure, for comparison purposes, we present the total execution time of the new scalable and more portable MPI [Snir and Gropp 1998] code on multiple clusters (see Section 5) with a large ($>$ 100) number of single-core processors. The data for the new version do not include pre-processing costs, because the code is restructured to perform all the preprocessing as soon as it has all required input. This happens much before the arrival of the first intra-operative image and thus the pre-processing time is masked by the time it takes to prepare the patient for the surgery.

Other less computationally intensive steps are: (1) the discretization (or mesh generation) of the domain, which is part of the pre-processing (see Figure 9) and (2) the solution of the (non-)linear system of equations that result from the discretization of the PDE. In [Fedorov et al. 2006] we presented a sequential method which addresses the mesh generation problem within



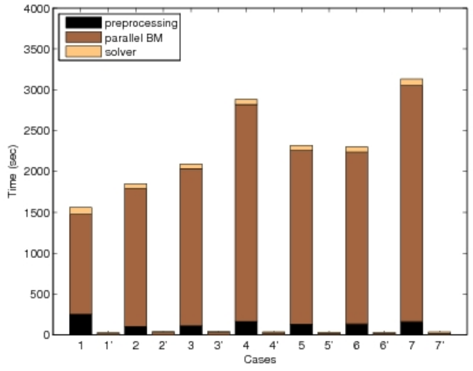


Figure 2: Breakdown of the execution time of: (1) the original parallel non-rigid registration application for cases $i = 1,7$ on W&M-HPC and (2) its new implementation for the same cases i', $i = 1,7$, but using eight clusters with 269 processors in two different administrative domains. *The data reported in the SC2006 paper (top); the same data collected with the code which does not save any unnecessary debugging information (bottom).*

the time constrains imposed by current Image Guided Neuro-Surgery (IGNS) procedures. A survey of parallel mesh generation methods that could address future needs of IGNS methods is presented in [Chrisochoides 2005]. Finally, for the the parallel solution of the (non-)linear system of equations, there is vast literature and some efficient publicly available software libraries like PETSc [Argonne National Laboratory 2005]. These two relevant topics (i.e., mesh generation and solution of systems of equations) are out of the scope of this paper.

In this paper we focus on two parallel and distributed computing issues pertinent to the productivity and effectiveness of neurosurgeons during IGNS procedures: (1) performance and (2) ease-of-use of the non-rigid registration procedure. The first issue introduces two well known and difficult problems: (a) load balancing and (b) fault-tolerance. The scientific computing community (including our group) has developed application-specific load balancing libraries and general purpose software runtime systems [Basermann et al. 2000; Yuan et al. 1997; Devine et al. 2000; Decker 2000; Friedman et al. 1997; Kalé and Krishnan 1993; Kohn and Baden 2001; Barker et al. 2004]. The issue of fault tolerance for parallel and distributed computing has also been addressed previously [Fagg and Dongarra 2000; Sankaran et al. 2005; Chakravorty and Kalé 2004; Elnozahy et al. 2002]. We do not use the existing software for load balancing and fault tolerance for the following three reasons: (1) the benefits these systems offer do not justify the additional complexity required to rewrite and maintain the code; (2) the smaller the number of third party libraries the registration code uses the easier its distribution becomes within the medical community; (3) the computation consists of separate phases, each of which requires a different approach to fault tolerance. We believe that customized application-specific load balancing and fault tolerance are most appropriate for this application.

Using a multi-level dynamic load balancing method we describe in Section 4.2 and fault-tolerance approach we describe in Section 4.3 we are able to safely run the registration application on multiple CoWs. As a result we have reduced the execution time of the parallel block matching by two orders of magnitude (i.e., less than 30 seconds for our last case) as compared to the execution time we get on BWH-HPC and W&M-HPC. In Section 4.4 we address the ease-of-use which help us to farther reduce the response time of the registration procedure by eliminating human intervention during the distributed non-rigid registration phase. We were able to reduce the involvement and mundane work of the research personnel of the hospital during an IGNS procedure and save (in addition to improvements from dynamic load balancing) up to 30 minutes. We present detailed performance evaluation data in Section 5.

2 Image guided neurosurgery

Neurosurgical resection is the primary therapeutic intervention in the treatment of cerebral gliomas [Black 1998]. The goal of tumor resection is maximal removal of neoplastic tissue from the patient brain while incurring minimal damage to healthy structures and regions. The resection procedure is greatly complicated by the inability of the human eye to distinguish abnormal from healthy tissue for certain kinds of tumors. At the same time, data collected with MRI techniques often permit good approximation of the tumor location and boundaries. The advantages of MRI include high resolution of images and absence of known negative side effects on patients health. MRI gives information not only about the location of tumors, but recent advances in MRI allow to identify some of the functions certain parts of the brain fulfill [Golby et al. 2002], and about the connectivity between different parts of the brain [Talos et al. 2002]. As an example, Figure 3 depicts different structural and functional data derived with MRI overlapped with the high-resolution greyscale image. Functional MRI (fMRI) studies allow to identify most important regions of brain (e.g., those responsible for motor or speech activity). Such areas should be carefully avoided during the surgery as much as possible. Diffusion Tensor MRI (DT-MRI) help to identify major fiber tracts, which connect cortex (grey matter) areas of brain. This is important, because partial disability of the patient can result not only from the damage of cortex, but also because of the disruption of the white matter connectivity. Depending on the complexity of the case, some or all of this information is usually collected before the surgery in order to identify as precise as possible the location of tumor, and to prepare the intervention strategy so that the most important healthy parts of the brain remain intact.

Unfortunately, as we have mentioned earlier, significant deformation of brain is possible during the course of the surgery. In order for pre-operative data to be useful, it has to be deformed to account for intra-operative changes. The forces which act on the brain matter during surgery as well as the biomechanical properties of the live tissue are hard to derive, which complicates direct modeling of tissue deformation. Most approaches are *displacement-based*. Sparse displacement field is derived with intraoperative imaging, and is used to guide the volume deformation, e.g., by using a patient-specific biomechanical model for such regularization. One of the approaches to acquire the sparse displacement field is to use open MR scanner, which allows to perform the surgery while the patient is being placed within the open magnet. One such scanner shown on Figure 3 in the inventory of Brigham and Women's Hospital (BWH) is used regularly for image guided prostate biopsy and neurosurgery cases [Warfield et al. 2005]. Images obtained with open scanner are not as high resolution as preoperative images, and may be noisy. Functional and Diffusion Tensor MRI [Golby et al. 2002; Talos et al. 2002] (see Figure 3, left) cannot be collected intra-operatively, because of procedure time and scanner resolution constraints. However, low resolution brain scans can be obtained relatively quickly and become handy for measuring the amount of residual tumor tissue during neurosurgical resection, which are completed within open MR scanner.

As the tumor is being resected progressively, and the surgeon is approaching the neighboring healthy tissue, more brain deformation occurs. Consequently, discrepancy between the pre-operative images and the reality is increasing. The research currently underway at BWH is attempting to use intra-operative MR data to track the brain shift and use this information for non-rigid registration of multi-modal preoperative [Ferrant 2001; Clatz et al. 2005] data. However, the registration results can be useful only if they are delivered to surgeons as soon as possible after the acquisition of an intra-operative scan: *delays must be minimal during the open skull surgery.* In the next section we concentrate on the computational

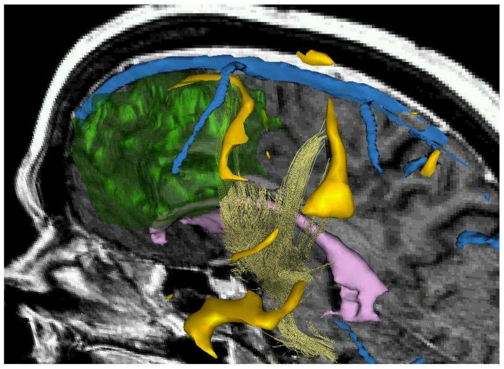
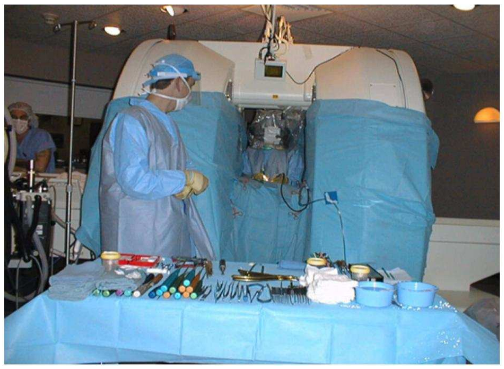



Figure 3: Left: MRI slice overlayed with fMRI activation regions, DT-MRI tractography, and segmentations of tumor and other intra-cranial areas; right: 0.5T open MR scanner (Signa SP, GE Medical System, Brigham and Women's Hospital).

challenges of the non-rigid registration approach [Clatz et al. 2005].

3 Computational aspects of non-rigid FEM registration

The approach to non-rigid registration which has been implemented, optimized and evaluated in this paper has previously been described in [Clatz et al. 2005; Archip et al. 2006]. The reader is encouraged to study those papers as they contain the detailed discussion and mathematical formulation of the implemented algorithm. In the present paper we focus only on computational challenges of the method, and give a brief outline of the algorithm for the sake of completeness. The registration algorithm consists of three main steps [Clatz et al. 2005].

1. The patient-specific model has to be generated. In the context of the application, the patient-specific model corresponds to the coarse mesh of segmented intra-cranial cavity of a patient. The segmentation [Archip et al. 2006] is prepared based on preoperative imaging data. Once the patient is placed in the open magnet, the position of his/her head is fixed and does not change during the course of the surgery. Rigid registration [Archip et al. 2006] is used to rigidly align the pre-operative data with the patient's position.

2. *Sparse* displacement field is estimated from the intra-operative scan using block matching. It is computed based on the minimization of the correlation coefficient between regions, or blocks, of the pre- and intra-operative images.

3. A finite element model of the intra-cranial cavity with linear elastic constitutive equation is initialized with the sparse displacement field. Approximation and interpolation formulations of the registration problem are combined in an iterative hybrid method which was described by Clatz et al. in [Clatz et al. 2005]. This combined approach is robust enough to tolerate and reject outliers in the beginning, and at the same time has been proven to converge toward the interpolation formulation.

Table 1: Execution time (sec) of the different phases for the non-rigid registration procedure (see Figure 9).

ID	Phase				
	A	B	C	C'	D
1	7.41	170.25	17.06	15.95	189.67
2	1.43	53.21	38.8	32.23	144.54
3	1.17	97.26	43.98	33.62	134.79
4	1.32	116.89	32.47	32.40	156.65
5	1.07	151.55	27.51	26.08	145.39
6	1.3	109.33	27.91	27.01	142.7
7	2.46	116.89	32.53	29.08	159.49

The result of this three step non-rigid registration is a displacement field, which for each voxel[1] of the floating image defines a displacement vector which maps it to the corresponding voxel of the fixed image.

While addressing the computational issues of the registration procedure, it is important to understand the separation of time-critical components of the computation. In the context of the application we define the *response time* as the time between the acquisition of the intra-operative scan of the deformed tissue and the final visualization of the registered preoperative data on the console in the surgery room. All of the steps, which may be solved before the fixed image is acquired, have to be separated and pre-computed, so that the response time is minimized. We present the timeline of a typical surgery scenario in Figure 9.

[1] *Voxel*, short for "volume pixel" is the smallest distinguishable component of a 3D image.

Intra-cranial cavity has to be segmented [Archip et al. 2006] from the high resolution MRI of the patient before the surgery. On the surgery day, once the first intra-operative scan is acquired (before the resection starts), the pre-operative data has to be rigidly registered. According to the transformation matrix obtained from the rigid registration [Archip et al. 2006], segmented intra-cranial cavity mask has to be updated, so that no intra-operative segmentation of the registered data is required. Next we generate a coarse tetrahedral mesh [Fedorov et al. 2006] model of the intra-cranial cavity. The block matching part of the algorithm is initialized and the relevant blocks are selected. As evident from Figure 9 and Table 1, initialization is a time-consuming part of the method. However, the time between the acquisition of the first intra-operative scan and the beginning of the resection is usually more than an hour, which is sufficient to complete the initialization (see Figure 9).

Once the intra-operative image which shows brain shift is acquired, we can proceed with the deformation computation. Note, that the rigid component of the deformation (translation and rotation) has already been eliminated by rigid registration of the preoperative data. We seek to find a deformation field, which for each voxel of the preoperative (*floating*) image identifies its mapping within the intraoperative (*fixed*) image. Block matching is a well-established method in computer vision [Bierling 1988]. This method is based on the assumption that deformations result in translation for some portions of the image, and that these translations are relatively small. We first subdivide the floating image into blocks of fixed size, as shown in Figure 4, left. Given such a block, we need to find the most similar block in its neighborhood. The search is performed within the fixed image, and is constrained by the *search window* of fixed size around the block (see Figure 4, right). The location of the block within its window which maximizes its cross-correlation with the corresponding window volume is selected as the *candidate displacement*.

The displacement resulting after block-matching require additional processing to reject outliers, because of the noise present in the image, and the reader is referred to [Clatz et al. 2005] for the details of iterative outlier rejection scheme. The final displacement can be applied to the preoperative fMRI and DT-MRI data.

Block matching calculation is the most time-consuming computational component of the registration procedure. It is also straight-forward to parallelize: matches for different blocks of the image can be computed independently in parallel provided the processing elements (PEs) are initialized with the fixed and floating images.

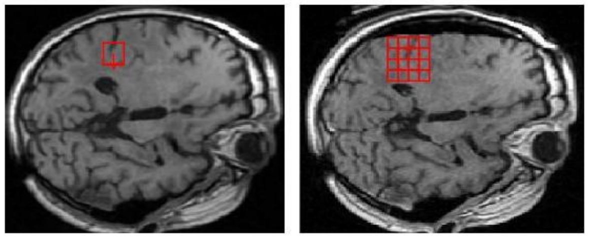


Figure 4: Highlighted is a block of the floating image on the left and the corresponding window of the fixed image on the right.

4 Real-world enabled non-rigid FEM registration

Neurosurgeries at BWH are performed regularly within open MR scanner at the discretion of the neurosurgeon. The scanner is used mostly for adjusting the predetermined resection path and tracking residual tumor tissue during the course of the intervention. Non-rigid registration of the pre-operative MRI is not a part of existing FDA approved protocols, but is indeed a very promising direction of future improvement of the procedure. It is hard to evaluate its usefulness without practical experience with near to real-time delivery of the results to neurosurgeons during the course of intervention, and this is exactly what we aim to provide at this point. The details of the IGNS procedure have been previously presented in [Archip et al. 2006]. In this section we address the challenges which we have to address in order to make this clinical study possible.

The evaluation of the implementation of the non-rigid registration method presented in [Clatz et al. 2005] on seven cases indicates that the execution time is dominated by the sparse displacement field computation. Figure 2 depicts the breakdown on the execution time of our original implementation on the dedicated *W&M-HPC* workstation which is similar to the *BWH-HPC* platform at BWH, where the high execution times were observed first. The execution time can be improved by utilizing parallel computational resources. However, by moving the application from a homogeneous and small number of dedicated computers to large scale heterogeneous and time-shared clusters, we have to address three practical problems: (1) load-balancing, (2) fault-tolerance, and (3) ease-of-use.

4.1 Multi-level distributed block matching

Our initial goal was to introduce minimal changes to the original implementation presented in [Clatz et al. 2005]. However, it was more important to solve the practical problems, otherwise the code could not be usable during IGNS. As we show in the subsequent sections, this was not possible to achieve with the simple approach we have tried, and major rethinking and restructuring of the initial code was required. In this section we give a high level overview of the new distributed block matching implementation. The benefits of our approach will be evident from the subsequent sections.

The processing required by the block matching is embarrassingly parallel. In order to find a match for a given block, we need the block center coordinates, and the areas of the fixed and floating images bounded by the block matching window [Clatz et al. 2005]. The fixed and floating images are loaded on each of the processors during the preprocessing step, as shown in Figure 9. The total workload is maintained in a *work-pool* data structure. Each item of the work-pool contains the three coordinates of the block center (total number of blocks for a typical dataset is around 100,000), and the best match found for that block (in case the block was processed; otherwise that field is empty). We use the master-worker computational model to distribute the work among the PEs.

However, because of the scarce resource availability we have to be able to deal with computational clusters which belong to different administrative domains. In order to handle this scenario, we use hierarchical multi-level organization of the computation with master-worker model within each cluster and work-stealing for inter-cluster load balancing. We select a separate master node within each cluster. Each master maintains a replica of the global work-pool, and is responsible for distributing the work according to the requests of the nodes within the assigned cluster, and communicating the progress of work completion to the other master(s). We use MPI for intra-cluster communication with the master node, and TCP sockets for inter-cluster work-pool synchronization (we assume there are open ports in the firewalls of the computing clusters).

Adjustable parameters of our model are (i) the amount of work (number of blocks) which is assigned by a master to a processor in response to a work request; (ii) frequency of the synchronization communication between masters to keep the work-pool replicas coherent. At one extreme, we can completely eliminate the communication by static assignment of work to all the nodes in the beginning of the computation, while on the other extreme we can request work one block at a time, and synchronize work-pools after completing each match calculation. These parameters were adjusted experimentally. In the next section we explore different approaches to work assignment, and elaborate on how the hierarchical model allows us to address the practical problems associated with the implementation.

4.2 Load balancing

Load balancing is one of the primary concerns during the parallel block matching phase of the algorithm. The work can be assigned to the PEs in the beginning by distributing block centers and the two input images. Communication is required only for the collection of the results in order to assemble the final solution. The computation is highly scalable. As evident from Table 2, we see a significant reduction in the runtime as we increase the number of processors involved. However, the same Table shows that the reduction is not proportional to the number of processors we use, because the scalability of our application is hindered by load imbalances.

Table 2: Execution time (sec) of sparse displacement field computation with static work distribution on WM-SciClone cluster.

ID	Number of processors					
	40	80	120	160	198	240
1	377.82	199.96	142.14	110.53	91.33	60.84
2	173.41	101.49	158.08	89.72	52.75	29.70
3	316.20	159.34	105.50	81.40	65.28	53.54
4	407.82	218.86	150.56	113.17	93.67	71.17
5	353.96	183.20	127.19	105.06	83.43	57.09
6	298.97	151.16	104.72	81.30	64.13	49.53
7	300.99	151.48	105.10	80.11	64.37	49.49

The imbalance of the processing time across different nodes involved in the computation is caused by our inability or difficulty to predict the time required per block of data on a given architecture. If we have that processing time information before we start block matching, we can perform balanced assignment and minimize the idle time. First, there are *data-dependent* sources of load imbalance. Block matching is performed only for those blocks, centers of which are located within the segmented mask. Computational costs of block matching are lower for windows which are partially outside the segmentation compared to those which are completely internal. However, the small fraction of blocks located near by the boundary has a small impact on the overall imbalance. The *platform-dependent* sources of imbalance are more challenging. They are caused by the heterogeneous nature of the PEs we use. Also, some of the resources may be time-shared by multiple users and applications, which affect the processing time in an unpredictable manner.

The static load balancing approach uses fixed data equidistribution of work among the processors of a single or multiple clusters. This approach is not applicable for heterogeneous and time-shared environments. A straightforward extension of the static equi-distribution assignment should take into account the difference in processing time for blocks on different platforms. Experimental data we have collected confirmed, that the CPU clock speed is a good predictor for the processing time of a single block in a heterogeneous environment. We implemented heterogeneous static load balancing, which assigns blocks proportionally to the CPU clock speed. Figure 5 depicts the execution of the non-rigid registration using heterogeneous assignment and (retrospectively) the intra-operative image from case 7.

However, static work assignment of any kind is effective only when we are able to predict the processing

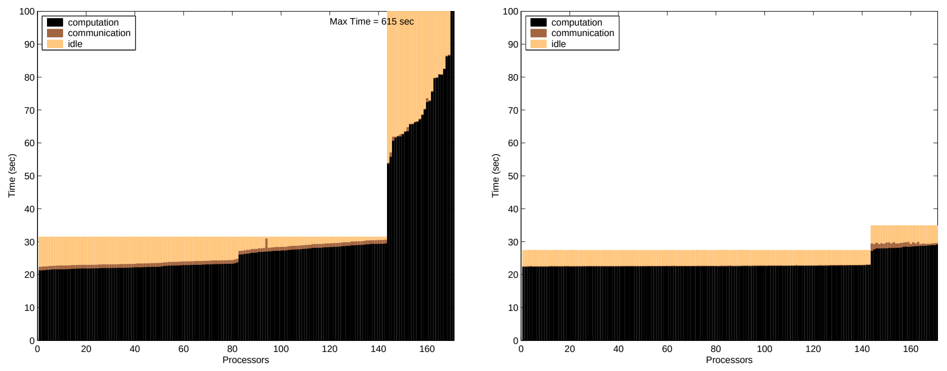


Figure 5: Processing time breakdown, 144 nodes of *WM-SciClone* and 29 nodes of *WM-CSLab*. Static weighted work assignment (left, 615 sec) is not effective in time-shared environment of *WM-CSLab*. Dynamic load balancing alleviates this problem (right, 35 sec).

time for a unit of work, which is difficult to accomplish if the computation is performed in a time-shared environment. The time-critical nature of the computation forces us to utilize all possible computational resources which might be available during the course of the surgery. At least a portion of the high end workstations available on time-shared clusters are usually underutilized. Also, we cannot rely on the availability of the dedicated cluster resources during the surgery given the statistics on wait times as shown on Figure 6. A dedicated cluster can only be made available for the sole use of our computation if we ask the system administrator to intervene, in which case from 2 to 5 days are required to drain the job queues. Such an advance notice is not always available. Furthermore, the resources are underutilized during the wait time, and the surgery may even be canceled in the last moment. This suggests, that in order to deliver the registration results in a timely fashion we need to employ all available dedicated and time-shared resources; their efficient utilization requires dynamic load balancing.

Multi-level dynamic load balancing method which we adopted for parallel block matching is different from the static multi-level graph partitioning methods presented in [Abou-Rjeili and Karypis 2006], and is closer to the hierarchical partitioning approaches described in [Teresco et al. 2005; Faik 2005]. We use initial rough estimation of the combined computational power of each cluster involved in the computation (based on CPU clock speed) for the weighted partitioning of the work-pool and initial assignment of work. However, this is a rough "guess" estimation, which is adjusted at runtime using a combination of master/worker and work-stealing [Blumofe et al. 1995; Wu 1993] methods. As we described earlier, each master has a copy of the global work-pool, which are identical in the beginning of the computation. The portion of the work-pool assigned to a specific cluster is partitioned in meta-blocks (a sequence of blocks), which are passed to the cluster nodes using the master-worker model. As soon as all the

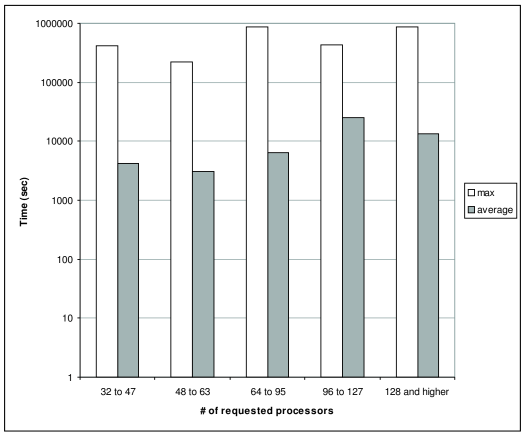


Figure 6: Average wait-in-queue time statistics, as observed over last 4.5 years on WM-SciClone cluster.

matches for a meta-block are computed, they are communicated back to the master, and a new meta-block is requested. In case the portion of the work-pool assigned to a master is processed, the master continues with the "remote" portions of work (i.e., those, initially assigned to other clusters). As soon as the processing of a "remote" meta-block is complete, it is communicated to all the other master nodes to prevent duplicated computation. As apparent from Figure 5, this approach works nicely even when we have clusters which operate in the time-shared mode, and gives good balance of work within a particular cluster and between the clusters.

4.3 Fault tolerance

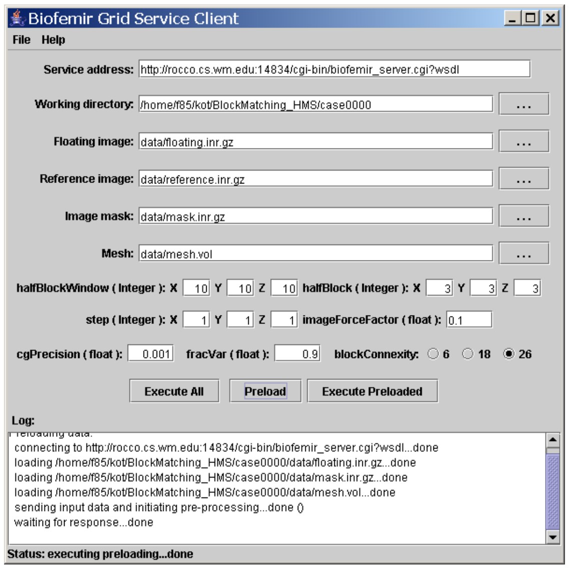
As we have shown in the previous sections, non-rigid registration and especially its block matching component, are computationally demanding. Our goal is to show that the results can be delivered intra-operatively. We think it is absolutely necessary to provide at least minimum fault tolerance in the code in order to minimize the possibility of any delay after the deformed intra-operative image is acquired.

Although our implementation is not restricted to two computational clusters, only W&M CSLab computers (further referred to as *WM-CSLab*) and the W&M SciClone cluster [College of William&Mary 2006] (further referred to as *WM-SciClone*) were available and used for the purposes of our study. We perform data pre-processing sequentially on the fastest machine at the *WM-CSLab*, parallel block matching uses combined computational facilities of the *WM-CSLab* and *WM-SciClone*, and the sequential FEM solver is executed sequentially on the fastest machine at *WM-CSLab* (the web server is executed on a separate machine, which is not the fastest machine available; this is happening because of the site-specific technical details). Our implementation is completely decoupled, which provides the first level of fault tolerance, i.e., if the failure takes place at any of the stages, we can seamlessly restart just the failed phase of the algorithm and recover the computation.

The communication between different clusters is handled via TCP sockets. In case of a hardware problem on one of the clusters, the rest of the unaffected sites can continue with the execution —a monolithic code which uses a conventional MPI implementation, would crash in case of a single node failure.

The second level of fault tolerance concerns with the parallel block matching phase. It is well-known that the vulnerability of parallel computations to hardware failures increases as we scale the size of the system [Fagg and Dongarra 2000]. We would like to have a robust system which in case of failure would be able to continue the parallel block matching without recomputing results obtained before the failure. This functionality is greatly facilitated by maintaining the previously described work-pool data-structure which is maintained on both *CSLab* and *WM-SciClone* master nodes.

The work-pool data-structure is redundantly replicated on the separate file-systems of these clusters, and has a tuple for each of the block centers. A tuple can be either empty, if the corresponding block has not been processed, or otherwise it contains the three components of the best match for a given block. The work-pool is synchronized periodically between the two clusters, and within each cluster it is updated by the PEs involved. Based on the statistical data collected over the last four years, most of the failures are caused by the power outages, which are happening mostly because of the inclement weather, and glitches related to air conditioning in machine rooms. Campus-wide power outage is a rare occurrence, and as long as one of the clusters remains operational, we are able to sustain the failure of the other computational site.



Service address: http://rocco.cs.wm.edu:14834/cgi-bin/biofemir_server.cgi?wsdl

Working directory: /home/f85/kot/BlockMatching_HMS/case0000

Floating image: data/floating.inr.gz

Reference image: data/reference.inr.gz

Image mask: data/mask.inr.gz

Mesh: data/mesh.vol

halfBlockWindow (Integer): X 10 Y 10 Z 10 halfBlock (Integer): X 3 Y 3 Z 3

step (Integer): X 1 Y 1 Z 1 imageForceFactor (float): 0.1

cgPrecision (float): 0.001 fracVar (float): 0.9 blockConnexity: 6 18 [✓ 26]

```
connecting to http://rocco.cs.wm.edu:14834/cgi-bin/biofemir_server.cgi?wsdl...done
loading /home/f85/kot/BlockMatching_HMS/case0000/data/floating.inr.gz...done
loading /home/f85/kot/BlockMatching_HMS/case0000/data/mask.inr.gz...done
loading /home/f85/kot/BlockMatching_HMS/case0000/data/mesh.vol...done
sending input data and initiating pre-processing...done ()
waiting for response...done
```

Status: executing preloading...done

Figure 7: The Graphical User Interface (GUI) of the web-service for the parallel and distributed non-rigid registration application.

Both the frequency of work-pool updates within each cluster and the intra-cluster synchronization is an adjustable trade-off between the communication time and a possibility of duplicate block processing (this may happen, if a block has been processed in one site, but the work-pool update has not yet been propagated). However, note that frequent synchronization of work-pools can be done by a separate background thread not involved in the computation, and the communication latencies associated with intra-cluster updates can be generally hidden by employing non-blocking communication feature of existing communication libraries. Overall, these side effects are negligible compared to the benefits we gain by our ability to save the computation state.

4.4 Ease of use

We developed a client web-service using Apache Axis implementation of SOAP [Apache Software Foundation 2006], Java for portability and a simple GUI for ease-of-use (see Figure 7). The client is responsible for gathering input information from the computational radiologist (e.g., input file and execution options), performing the remote invocation of service functions and delivering the results. In addition, it will perform sanity and compatibility checks of the input data to avoid errors at later time on the server-side.

5 Evaluation

The performance evaluation is based on non-rigid registration of seven image datasets acquired at BWH. Two of these seven registration computations were accomplished during the course of surgery, while the rest of the computations were done retrospectively. The surgery for case 6 was postponed in the last moment due to patient's health complications, and the analysis was performed retrospectively. All of the intra-operative computations utilized *WM-SciClone*, which was reserved in advance for the neurosurgery use.

Experimental Setup The 4-processor clusters where we run the original PVM code are: (1) at BWH, a 2-way Dell Precision Workstation 650, Intel(R) Xeon(TM) 3.06GHz and and a 2-way 3.20GHz processors both with 1MB Level 3 cache 512KB L2 cache and 3.5GB RAM (referred as *BWH-HPC* throughout the text) and (2) at W&M a Dell PowerEdge 6600 4 HT processors (8 hyperthreads) with Intel(R) XEON(TM) MP CPU 1.50GHz, 512KB L2 cache 16GB RDRAM (referred as *WM-HPC* throughout the text). All the experiments for the new implementation we present here were performed in eight clusters: (1) a cluster of 30 Dell Precision 360n single-core cpu @ 3 to 3.6GHz and 1GB SDRAM 333MHz which is used from CS students for class assignments and projects (*WM-CSLab*) and (2) WM-SciClone cluster of W&M [College of William&Mary 2006] (*WM-SciClone*). These clusters belong in different administrative domains.

SciClone is a heterogeneous cluster maintained at College of William and Mary. It is arranged as seven subclusters which can be used individually or together. The extensive details of hardware configuration and structure of *SciClone* are available elsewhere [College of William&Mary 2006]. We used all but *Typhoon* subclusters of *SciClone*.

The networks of College of William and Mary and BWH (subnet of Harvard University) are connected with a high performance *Internet2* backbone network [Internet2 consortium 2006] with the slowest link having bandwidth of 2.5 Gbps, which facilitates quick transfer of intra-operative data.

In all cases *gcc 3.3* was used for compilation.

Discussion The original PVM implementation (being the first prototype) was structured to prove the concept of the method [Clatz et al. 2005], rather than being used intraoperatively for non-rigid registration. Its utility during neurosurgery was hindered by the high execution time on our local resources at BWH. This problem has been addressed very successfully in this paper using a number of optimizations. The quantitative results are summarized in Tables 3 and 4.

Portability of the code has been improved. Original implementation used PVM [Belguelin et al. 1993] that is not widely supported nowadays. The new implementation is based on MPI [Snir and Gropp 1998]. From porting the code from BWH-HPC cluster at BWH to the dedicated 240 processor WM-SciClone at W&M we improved the performance of the application by 3.5 times (see first and second rows of the Table 3). One would

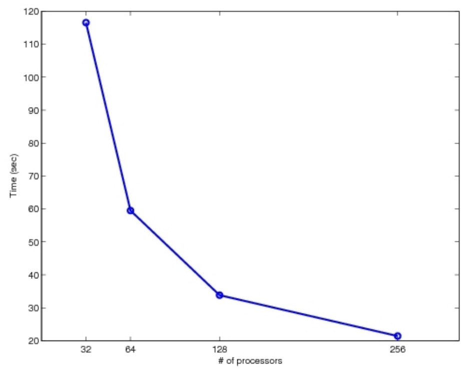


Figure 8: Parallel blockmatching scalability on the NCSA *TeraGrid* site.

expect a linear improvement (i.e., 60 times) due to the scalable nature of the algorithm. However, this was not feasible due to the following three reasons: (1) monolithic design of the original implementation, i.e., it was designed to run on the same static configuration from the beginning to the end; (2) the fastest of the 240 processors of *WM-SciClone* is much slower (in terms of CPU speed) than the nodes of *BWH-HPC* and *WM-CSLab*, and (3) presence of work load imbalances.

Decoupling of the pre-processing phase from intra-operative image registration. This allowed us to: (1) execute the sequential pre-processing phase on the fastest available workstation available and (2) *mask latencies* caused by MPI initialization of the parallel component of the implementation. As a result we were able to reduce the response time of the non-rigid registration on average by more than 12 minutes (see second and third rows of Table 3 and Figure 9). The maximum improvement we observed so far was more than 16 minutes (case 1).

Multi-level dynamic load balancing over multiple clusters reduced the execution time by more than 14 minutes when we used 240 processors and the decoupled implementation of the code (compare the second row with the fourth and fifth rows of Table 3). The flexibility we achieved from the portability of the code and these two optimizations (decoupling and dynamic load balancing) together reduced the absolute response time of the parallel block matching code alone by two orders of magnitude; on *WM-HPC* we can complete this step in 1225 seconds (case 1) and 2890 seconds (case 7) while with all the computational resources of multiple clusters we could possibly utilize, we complete it in less than 30 seconds (see Figure 5 and Table 4). These experimental data indicate, that we gain about 50% improvement in the performance of the application due to dynamic load balancing i.e., we can achieve the same performance with half of the resources; for hospitals where space is at premium this is a very important result.

Web-services became an option after the decoupling and implementing the functionality to handle platform-dependent aspects of heterogeneous and time-shared

Table 3: Execution time (sec) of the intra-surgery part of the implemented web-service at various stages of development. *Striked through items correspond to the data reported in the SC06 published paper. The code used originally has been saving debugging data which significantly hindered performance. The table reports new time measurements collected with optimized no-debug code.*

Setup	ID						
	1	2	3	4	5	6	7
WM-HPC, using origina PVM implementation	1558 ~~2556~~	1850 ~~2619~~	2090 ~~2803~~	2882 ~~3711~~	2317 ~~3083~~	2302 ~~3048~~	3130 ~~3961~~
WM-SciClone (240 procs), no load-balancing	745 ~~1361~~	639 ~~985~~	595 ~~949~~	617 ~~1112~~	570 ~~1011~~	550.4 ~~996~~	1153 ~~1126~~
Pre- and post-processing on single CS node, BM on WM-SciClone (240 procs), no load-balancing	226 ~~397~~	123 ~~252~~	182 ~~304~~	189 ~~331~~	217 ~~347~~	174 ~~302~~	189 ~~333~~
Pre- and post-processing on single CS node, pre-processing in advance, BM on WM-SciClone (240 procs), dynamic load-balancing	35 ~~206~~	53 ~~183~~	56 ~~178~~	47 ~~189~~	42 ~~172~~	42 ~~170~~	48 ~~192~~
Pre- and post-processing on single CS node, pre-processing in advance, BM on WM-SciClone (240 procs) and WM-CSLab(29 procs), dynamic 2-level load-balancing	**30** ~~200~~	**40** ~~169~~	**42** ~~163~~	**37** ~~179~~	**34** ~~163~~	**33** ~~161~~	**35** ~~179~~

clusters; the use of web-service and hence the elimination of the human factor from the loop reduced the response time by roughly 20 minutes. Originally, the initialization involved two teams, one at BWH and another at W&M and four different administrative domains, taking up to 30 minutes of time. With the use of GUI-interfaced web-services, the time to start the application at BWH and initiate its execution at W&M was reduced to 75 seconds [2], which is the bare-minimum cost for data transfer and I/O for the application. Computations for Case 7 (run retrospectively) took less than 7 minutes (as measured between the time the data was submitted for computation from BWH until the result was received back from W&M). *This is an improvement in the response time by 13 to 15 times.*

Fault-tolerance became very important attribute of the code, since by increasing the hardware components and the complexity of the distributed computing environment we use, we increase the chances for failures. Our implementation can safely run the registration procedure on several clusters in different locations at a modest overhead of 6% over the non-fault-tolerant version, and deliver results of the computation even in case of the failures of all but one cluster involved.

We have also evaluated our implementation on the *Mercury* nodes of the NCSA *TeraGrid* site [TeraGrid Project 2006]. The 64-bit homogeneous platform available at NCSA allows for high sustained computational power and improved scalability of the code (we attribute this to *gcc* optimization on this particular platform). We show the scalability results in Figure 8. We believe *TeraGrid* has a great potential for this application. However, we observed that reservation of large number of nodes may involve significant wait time (from hours to more than a week), which complicates the use of the *TeraGrid* resources for intra-operative computations: resources should be available on a short notice.

[2]57s to send the images from BWH to W&M and 18s to retrieve the output at BWH from W&M. These data vary due to network traffic, but consistently we observe them to be below 2 minutes total.

Table 4: Execution time (sec) of sparse displacement field computation with dynamic work distribution on WM-SciClone cluster.

ID	Number of processors					
	40	80	120	160	198	240
1	200.84	107.77	77.67	63.41	54.65	33.27
2	103.65	77.62	125.95	111.62	36.89	17.06
3	217.70	109.74	75.45	56.62	47.33	38.80
4	242.35	125.64	91.23	70.19	57.80	43.98
5	189.17	100.95	73.00	57.73	49.75	32.47
6	152.13	81.20	57.97	45.65	37.49	27.51
7	150.81	78.98	57.35	45.91	37.21	27.91

6 Conclusions

In this paper we use parallel and distributed computing as a utility to provide a faster and more effective decision making support for a time-critical application: Image Guided Neurosurgery. The work described in this paper showed, for the first time, the feasibility of near real-time image fusion for brain MRI using landmark tracking across the entire image volume. This became possible because we were able to achieve: (1)

the reduction of the execution time of the parallel block matching from 2890 seconds at BWH to less than 30 seconds at W&M (for the last intra-operative non-rigid registration case); (2) effective use of a large number of processors, with dynamic load balancing, we improved the performance of our original code for parallel block matching by 50%; (3) the reduction of the overheads associated with manual initialization and transfer of data from BWH from 20-30 minutes to about 60 seconds; (4) ease-of-use; and (5) the first fault-tolerant and web-service based non-rigid registration code using landmark tracking across the entire volume.

Acknowledgments

This investigation was supported in part by the NSF ACI-0312980, NSF EIA-0203974, NSF ITR 0426558, a research grant from CIMIT, grant RG 3478A2/2 from the NMSS, and by NIH grants R21 MH067054, R01 RR021885, P41 RR013218, and U41 RR019703. This work was performed in part using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund. We thank WM-SciClone administrator Tom Crockett for his continuous support and personal attention to this project. Steve Whalen at BWH provided a lot of help with image acquisition and coordination of the study.

References

Abou-Rjeili, A., and Karypis, G. 2006. Multi-level algorithms for partitioning power-law graphs. In *IEEE International Parallel and Distributed Processing Symposiu.*

American Cancer Society, 2005. Cancer facts and figures: 2005. `http://www.cancer.org/docroot/STT/stt_0_2005.asp?sitearea=STT\&level=1`, accessed 16 Feb 2006.

Apache Software Foundation, 2006. Apache axis project. `http://ws.apache.org/axis/`, accessed 23 April 2006.

Archip, N., Fedorov, A., Lloyd, B., Chrisochoides, N., Golby, A., Black, P. M., and Warfield, S. K. 2006. Integration of patient specific modeling and advanced image processing techniques for image guided neurosurgery. In *Proceedings of SPIE 6141 Medical Imaging 2006: Visualization, Image-Guided Procedures, and Display*, K. Cleary and R. G. Jr., Eds., vol. 61411E.

Argonne National Laboratory, 2005. PETSc: Portable, extensible toolkit for scientific computation. v.2.3.0, `http://www-unix.mcs.anl.gov/petsc/petsc-2/`, Accessed 16 Feb, 2006.

Barker, K., Chernikov, A., Chrisochoides, N., and Pingali, K. 2004. A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems 15*, 2 (Feb.), 183–192.

Basermann, A., Clinckemaillie, J., Coupez, T., Fingberg, J., Digonnet, H., Ducloux, R., Gratien, J., Hartmann, U., Lonsdale, G., Maerten, B., Roose, D., and Walshaw, C. 2000. Dynamic load-balancing of finite element applications with the drama library. *Applied Mathematical Modeling 25*, 83–98.

Belguelin, A., Dongarra, J., Geist, A., Manchek, R., Otto, S., and Walpore, J. 1993. Pvm: Experiences, current status, and future direction. In *Supercomputing '93 Proceedings*, 765–766.

Bierling, M. 1988. Displacement estimation by hierarchical block matching. In *Proceedings of SPIE Visual Communication and Image Processing*, vol. 1001, 942–951.

Black, P. 1998. Management of malignant glioma: role of surgery in relation to multimodality therapy. *Journal of Neurovirology 4*, 2, 227–236.

Blumofe, R., Joerg, C., Kuszmaul, B., Leiserson, C., Randall, K., and Zhou, Y. 1995. Cilk: An efficient multithreaded runtime system. In *Proceedings of the 5th Symposium on Principles and Practice of Parallel Programming*, 55–69.

Chakravorty, S., and Kalé, L. 2004. A fault tolerant protocol for massively parallel systems. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium.*

Chrisochoides, N. 2005. *Numerical Solution of Partial Differential Equations on Parallel Computers.* No. 51 in Lecture Notes in Computational Science and Enginering. Springer-Verlag, December, ch. Parallel Mesh Generation, 237–264.

Clatz, O., Delingette, H., Talos, I. F., Golby, A. J., Kikinis, R., Jolesz, F. A., Ayache, N., and Warfield, S. K. 2005. Robust non-rigid registration to capture brain shift from intra-operative MRI. *IEEE Transactions on Medical Imaging 24*, 11, 1417–1427.

College of William&Mary, 2006. SciClone Cluster Project. `http://www.compsci.wm.edu/SciClone/`, accessed 23 April 2006.

Decker, T. 2000. Virtual data space - load balancing for irregular applications. *Parallel Computing 26*, 1825–1860.

Devine, K., Hendrickson, B., Boman, E., John, M. S., and Vaughan, C. 2000. Design of dynamic load-balancing tools for parallel applications. In *Proc. of the Int. Conf. on Supercomputing.*

Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys 34*, 3, 375–408.

Fagg, G. E., and Dongarra, J. 2000. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer-Verlag, 346–353.

Faik, J. 2005. *A Model for Resource-Aware Load Balancing on Heterogeneous and Non-Dedicated Clusters*. PhD thesis, Rensselaer Polytechnic Institute, Troy.

Fedorov, A., Chrisochoides, N., Kikinis, R., and Warfield, S. K. 2006. An evaluation of three approaches to tetrahedral mesh generation for deformable registration of MR images. In *Proceedings of IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2006)*, 658–661.

Ferrant, M. 2001. *Physics-based Deformable Modeling of Volumes and Surfaces for Medical Image Registration, Segmentation and Visualization.* PhD thesis, Universite Catholique de Louvain.

Friedman, R., Goldin, M., Itzkovitz, A., and Schuster, A. 1997. Millipede: Easy parallel programming in available distributed environments. *Software-Practice and Experience 27*, 8 (August), 929–965.

Golby, A. J., Poldrack, R. A., Illes, J., Chen, D., Desmond, J. E., and Gabrieli, J. D. 2002. Memory lateralization in medial temporal lobe epilepsy assessed by functional MRI. *Epilepsia 43*, 8, 855–863.

Internet2 consortium, 2006. Internet2 home page. `http://www.internet2.edu/`, accessed 23 April 2006.

Kalé, L., and Krishnan, S. 1993. CHARM++: A portable concurrent object oriented system based on C++. In *Proceedings of OOPSLA '93*, 91–108.

Kohn, S., and Baden, S. 2001. Parallel software abstractions for structured adaptive mesh methods. *Journal of Par. and Dist. Comp. 61*, 6, 713–736.

Rexilius, J., 2001. Physics-based nonrigid registration for medical image analysis. MS Thesis.

Sankaran, S., Squyres, J. M., Barrett, B., Lumsdaine, A., Duell, J., Hargrove, P., and Roman, E. 2005. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. *International Journal of High Performance Computing Applications 19*, 4, 479–493.

Snir, M., and Gropp, W. 1998. *MPI: The Complete Reference.* The MIT Press.

Talos, I. F., Walker, D., Zou, K., Kikinis, R., Jolesz, F., and Black, P. 2002. Factors affecting resectability of adult hemispheric low-grade gliomas under intraoperative MRI-guidance. *European Radiology 12*.

TeraGrid Project, 2006. TeraGrid Home page. `http://teragrid.org/`, accessed 23 April 2006.

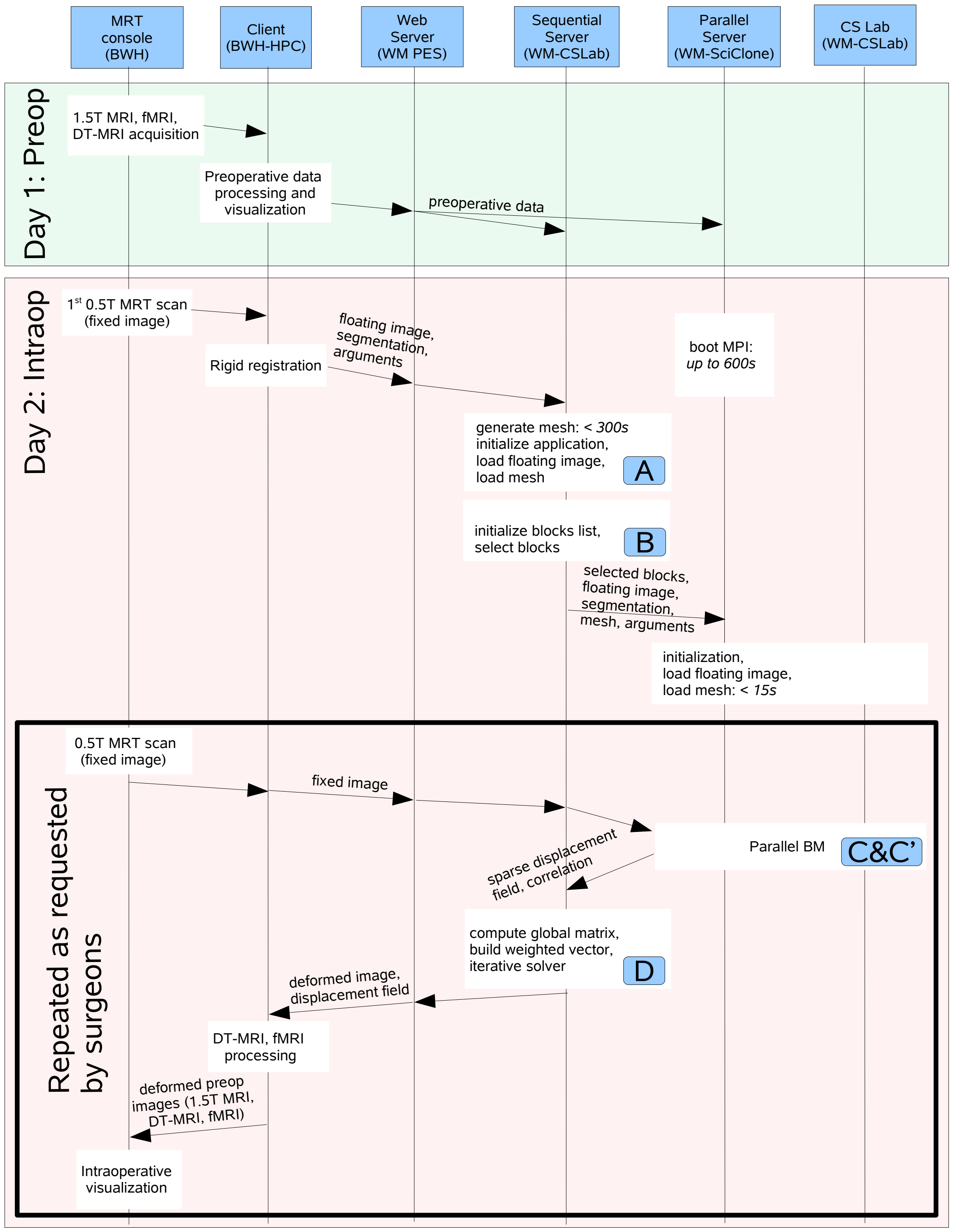
Teresco, J. D., Faik, J., and Flaherty, J. E. 2005. Resource-aware scientific computation on a heterogeneous cluster. *Computing in Science & Engineering 7*, 2, 40–50.

Warfield, S. K., Talos, F., Tei, A., Bharatha, A., Nabavi, A., Ferrant, M., Black, P., Jolesz, F. A., and Kikinis, R. 2002. Real-time registration of volumetric brain MRI by biomechanical simulation of deformation during image guided neurosurgery. *Computing and Visualization in Science 5*, 1, 3–11.

Warfield, S. K., Haker, S. J., Talos, I. F., Kemper, C. A., Weisenfeld, N., Mewes, A. U. J., Goldberg-Zimring, D., Zou, K. H., Westin, C. F., and Wells, W. M. 2005. Capturing intraoperative deformations: research experience at Brigham and Women's Hospital. *Medical Image Analysis 9*, 2, 145–162.

Wu, I. 1993. *Multilist Scheduling: A New Parallel Programming Model.* PhD thesis, School of Comp. Sci., Carnegie Mellon University, Pittsburg, PA 15213.

Yuan, X., Salisbury, C., Balsara, D., and Melhem, R. 1997. Load balancing package on distributed memory systems and its application particle-particle and particle-mesh (P3M) methods. *Parallel Computing 23*, 10, 1525–1544.



MRT console (BWH)
Client (BWH-HPC)
Web Server (WM PES)
Sequential Server (WM-CSLab)
Parallel Server (WM-SciClone)
CS Lab (WM-CSLab)
Day 1: Preop
1.5T MRI, fMRI, DT-MRI acquisition
Preoperative data processing and visualization
preoperative data
Day 2: Intraop
1st 0.5T MRT scan (fixed image)
Rigid registration
floating image, segmentation, arguments
boot MPI: up to 600s
generate mesh: < 300s initialize application, load floating image, load mesh
A
initialize blocks list, select blocks
B
selected blocks, floating image, segmentation, mesh, arguments
initialization, load floating image, load mesh: < 15s
Repeated as requested by surgeons
0.5T MRT scan (fixed image)
fixed image
Parallel BM
C&C'
sparse displacement field, correlation
compute global matrix, build weighted vector, iterative solver
D
deformed image, displacement field
DT-MRI, fMRI processing
deformed preop images (1.5T MRI, DT-MRI, fMRI)
Intraoperative visualization

Figure 9: Timeline of the image processing steps during IGNS using the new non-rigid registration application (see Table 1 for time spent in each phase).