Grid-Enabled Software Environment for Enhanced Dynamic Data-Driven Visualization and Navigation during Image-Guided Neurosurgery *

Nikos Chrisochoides[1], Andriy Fedorov[1], Andriy Kot[1], Neculai Archip[2], Daniel Goldberg-Zimring[2], Dan Kacher[2], Stephen Whalen[2], Ron Kikinis[2], Ferenc Jolesz[2], Olivier Clatz[3], Simon K. Warfield[3], Peter M. Black[4], and Alexandra Golby[4]

[1] College of William & Mary, Williamsburg, VA, USA
[2] Department of Radiology, Brigham and Women's Hospital, Boston, MA, USA
[3] Department of Radiology, Children's Hospital, Boston, MA, USA
[4] Department of Neurosurgery, Brigham and Women's Hospital, Boston, MA, USA

Abstract. In this paper we present our experience with an Image Guided Neurosurgery Grid-enabled Software Environment (IGNS-GSE) which integrates real-time acquisition of intraoperative Magnetic Resonance Imaging (IMRI) with the preoperative MRI, fMRI, and DT-MRI data. We describe our distributed implementation of a non-rigid image registration method which can be executed over the Grid. Previously, non-rigid registration algorithms which use landmark tracking across the entire brain volume were considered not practical because of the high computational demands. The IGNS-GSE, for the first time ever in clinical practice, alleviated this restriction. We show that we can compute and present enhanced MR images to neurosurgeons during the tumor resection within minutes after IMRI acquisition. For the last 12 months this software system is used routinely (on average once a month) for clinical studies at Brigham and Women's Hospital in Boston, MA. Based on the analysis of the registration results, we also present future directions which will take advantage of the vast resources of the Grid to improve the accuracy of the method in places of the brain where precision is critical for the neurosurgeons.

Key words: Image registration, Distributed Processing, Grid, Data-Driven Visualization

1 Introduction

Cancer is one of the top causes of death in the USA and around the world. Medical imaging, and Magnetic Resonance Imaging (MRI) in particular, provide great help in diagnosing the disease. In brain cancer cases, MRI provides

* This research was supported in part by NSF NGS-0203974, NSF ACI-0312980, NSF ITR-0426558, NSF EIA-9972853.

extensive information which can help to locate the tumor and plan the resection strategy. However, deformation and shift of brain structures is unavoidable during open brain surgery. This creates discrepancies as compared to the preoperative imaging during the operation.

It is possible to detect the brain shift during the surgery. One of the means to do this is IMRI. IMRI provides sparse dynamic measurements, which can be used to align (register) the preoperative data accordingly. In this way, high-quality, multimodal preoperative imaging can be used during the surgery.

However, registration is a computationally-intensive task, and it cannot be initiated before IMRI becomes available. Local computing resource available at a particular hospital may not allow to perform this computation in time. The goal of our research is to use geographically distributed computing resources to expedite the completion of this computation. In the on-going collaboration between Brigham and Women's Hospital (BWH) in Boston, MA, and College of William and Mary (CWM) in Williamsburg, VA, we are studying how widely available commodity clusters and Grid resources can facilitate the timely delivery of registration results. We leverage our work from the state-of-the art registration method [1], and extensive experience with distributed processing and dynamic load balancing [2, 3].

We have designed a robust distributed implementation of the registration method, which meets the following requirements concerning (1) execution speed, (2) reliability, (3) ease-of-use and (4) portability of the registration code. We evaluate this prototype implementation in a geographically-distributed environment, and outline how IGNS computations can benefit from large-scale computing resources like *TeraGrid*.

2 Near-real-time Non-Rigid Registration

2.1 Registration Algorithm

The registration method was first presented in [1], and subsequently evaluated in [4]. The computation consists of preoperative and intraoperative components. Intraoperative processing starts with the acquisition of the first intraoperative scan. However, the *time-critical* part of the intraoperative computation is initiated when a scan showing shift of the brain is available. The high-level timeline of this process is shown in Fig. 1.

Here we briefly describe the three main steps of the algorithm:

1. the patient-specific tetrahedral mesh model is generated from the segmented intra-cranial cavity (ICC). The ICC segmentation [4] is prepared based on pre-operative imaging data. As the first intraoperative scan is available, preoperative data is rigidly (i.e., using translation and rotation) aligned with the patient's head position;
2. with the acquisition of the intraoperative image showing brain deformation, sparse displacement field is estimated from the intra-operative scan using blockmatching [1]. Its computation is based on the minimization of

the correlation coefficient between regions, or blocks, of the pre-operative (aka floating) image and the real-time intra-operative (aka fixed) image;
3. the FEM model of the intra-cranial cavity with linear elastic constitutive equation is initialized with the mesh and sparse displacement field as the initial condition. An iterative hybrid method is used to discard the outlier matches.

Steps 2 and 3 are time critical and should be performed as the surgeons are waiting. 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 operating room. These steps performed intraoperatively form the Dynamic Data-Driven Application System (DDDAS [1]) steered by the IMRI-acquired data. Our broad objective is to minimize the perceived (end-to-end) response time of the DDDAS component.

2.2 Implementation Objectives

We have completed an evaluation of the initial PVM implementation [1] of the described registration approach. The evaluation was done on retrospective datasets obtained during past image-guided neurosurgeries. We identified the following problems:

1. The execution time of the original non-rigid registration code is data-dependent and varies between 30 and 50 minutes, when computed on a high-end 4 CPU workstation. The scalability of the code is very poor due to work-load imbalances.
2. The code is designed as a single monolithic step (since it was not evaluated in the intraoperative mode) to run and a single failure at any point requires to restart the registration from the beginning.
3. The original code is not intuitive to use, a number of implementation-specific parameters are required to be set in the command line. This makes it cumbersome and error-prone to use during neurosurgery. The possible critical delays are exacerbated in the case the code has to run remotely on larger Clusters of Workstations (CoWs).
4. The original code is implemented in PVM which is not supported by many sites due to widespread use of MPI standard for message passing.

Based on the evaluation of the original code, the following implementation objectives were identified:

High-performance Develop an efficient and portable software environment for parallel and distributed implementation of real-time non-rigid registration method for both small scale parallel machines and large scale geographically distributed CoWs. The implementation should be able to work on both dedicated, and time-shared resources.

[1] The notion of DDDAS was first coined and advocated by Dr.Darema, see http://dddas.org.

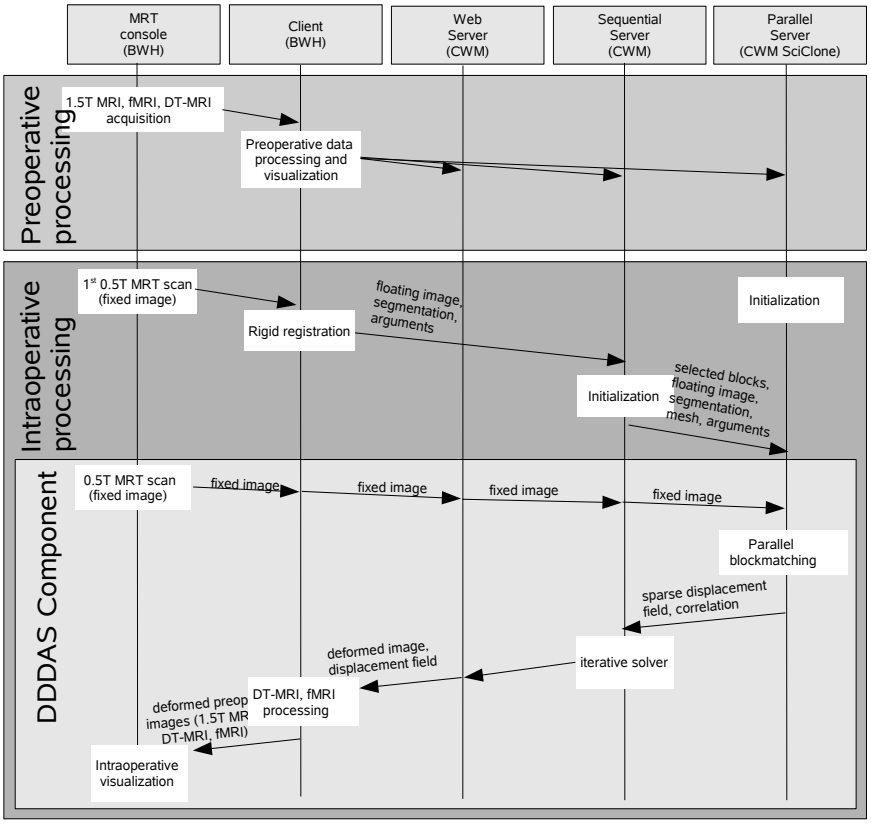

Fig. 1. Timeline of the image processing steps during IGNS (the client is running at BWH, and the server is using multiple clusters at CWM, for fault-tolerance purposes).

Quality-of-service (QoS) Provide functionality not only to sustain failure but also to dynamically replace/reallocate faulty resources with new ones during the real-time data acquisition and computation.

Ease-of-use Develop a GUI which automatically will handle exceptions (e.g., faults, resource management, and network outages).

We have developed an implementation which addresses the aforementioned objectives [5]. Next we briefly highlight some of the implementation details.

2.3 Implementation Details

Multi-level distributed block matching 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 [1]. The fixed and floating images are loaded on each of the processors during the initialization step, as shown in

Fig. 1. 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 processors.

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. We use 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 execution progress to the other master(s).

Multi-level Dynamic Load Balancing 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. The main sources of load imbalance are *platform-dependent*. These are caused by the heterogeneous nature of the PEs we use. More importantly, some of the resources may be time-shared by multiple users and applications, which affect the processing time in an unpredictable manner. The (weighted-) static work assignment of any kind is not effective when some of the resources operate in the time-shared mode.

We have implemented a multi-level hierarchical dynamic load balancing scheme for parallel block matching. 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 [6, 7] methods. 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 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.

Multi-Level Fault Tolerance 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 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. 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 by the master nodes.

The work-pool data-structure is 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. As long as one of the clusters involved in the computation remains operational, we will be able to sustain the failure of the other computational side and deliver the registration result.

Ease-of-use The implementation consists of the client and server components. The client is running at the hospital site, and is based on a Web-service, which makes it highly portable and easy to deploy. On the server side, the input data and arguments are transferred to the participating sites. Currently, we have a single server responsible for this task. The computation proceeds using the participating available remote sites to provide the necessary performance and fault-tolerance.

Table 1. Execution time (sec) of the intra-surgery part of the implemented web-service at various stages of development.

Setup	ID						
	1	2	3	4	5	6	7
High-end workstation, using original PVM implementation	1558	1850	2090	2882	2317	2302	3130
SciClone (240 procs), no load-balancing	745	639	595	617	570	550.4	1153
SciClone (240 procs) and CS lab(29 procs), dynamic 2-level load-balancing and fault-tolerance	**30**	**40**	**42**	**37**	**34**	**33**	**35**

3 Initial Evaluation Results

Our preliminary results use seven image datasets acquired at BWH. The computations for two of these seven registration computations were accomplished during the course of surgery (at the College of William and Mary), while the rest of the computations were done retrospectively. All of the intra-operative computations utilized *SciClone* (a heterogeneous cluster of workstations located at CWM, reserved in advance for the registration computation) and the workstations of the student lab (time-shared mode). The details of the hardware

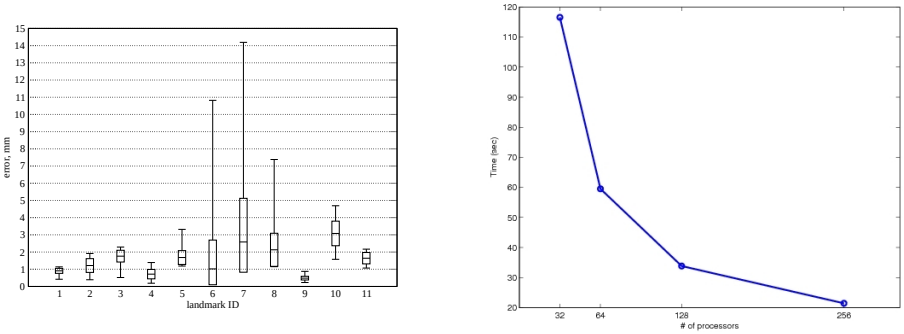

Fig. 2. Registration accuracy dependence on proper parameter selection (left). Excellent scalability of the code on the NCSA site of *TeraGrid* (right) enables intraoperative search for optimal parameters.

configuration can be found in [5]. Data transfer between the networks of CWM and BWH (subnet of Harvard University) are facilitated by the Internet2 backbone network with the slowest link having bandwidth of 2.5 Gbps.

The initial evaluation results are summarized in Table 1. We were able to reduce the total response time to 2 minutes (4 minutes, including the time to transfer the data). We showed, that dynamic load balancing is highly effective in time-shared environment. Modular structure of the implemented code greatly assisted in the overall usability and reliability of the code. The fault-tolerance mechanisms implemented are absolutely essential and introduce a mere 5-10% increase in the execution time. We have also evaluated our implementation on the *Mercury* nodes of the NCSA *TeraGrid* site [8]. The 64-bit homogeneous platform available at NCSA allows for high sustained computational power and improved scalability of the code (see Fig. 2).

4 Discussion

The registration algorithm we implemented has a number of parameters whose values can potentially affect the accuracy of the results. The evaluation of all parameters is computationally demanding (the parameter space of the algorithm has high dimensionality), which requires vast computational resources that are available only over the Grid. Based on the preliminary analysis, registration accuracy is dependent on the parameter selection. Fig. 2 shows the spread of registration precision at expert-identified anatomical landmarks. Given the distributed resources available within *TeraGrid*, we should be able to compute in parallel registration results which use different parameter settings. As the multiple registrations become available, the surgeon will specify the area of interest within the brain, and the registration image which gives the best *effective* accuracy in that particular region will be selected.

The implemented framework proved to be very effective during on-going clinical study on nonrigid registration. However, more work needs to be done to make the framework portable and easy to deploy on an arbitrary platform. Once this is complete, the registration can be provided as a ubiquitously-available Web service. The concerns about resource allocation and scheduling on a shared resource like *TeraGrid* are of high importance. The presented research utilized time-shared resources together with a large cluster operating in dedicated mode. However, we are currently investigating other opportunities, e.g., SPRUCE and urgent computing [9] on *TeraGrid*.

Acknowledgments. 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 SciClone administrator Tom Crockett for his continuous support and personal attention to this project. We acknowledge support from a research grant from CIMIT, grant RG 3478A2/2 from the NMSS, and by NIH grants R21 MH067054, R01 RR021885, P41 RR013218, U41 RR019703, R03 EB006515 and P01 CA067165.

References

1. Clatz, O., Delingette, O., Talos, I.F., Golby, A., Kikinis, R., Jolesz, F., Ayache, N., Warfield, S.K.: Robust non-rigid registration to capture brain shift from intra-operative MRI. IEEE Trans. Med. Imag. **24**(11) (2005) 1417–1427
2. Barker, K., Chernikov, A., Chrisochoides, N., Pingali, K.: A load balancing framework for adaptive and asynchronous applications. IEEE TPDS **15**(2) (February 2004) 183–192
3. Fedorov, A., Chrisochoides, N.: Location management in object-based distributed computing. In: Proc. of IEEE Cluster'04. (2004) 299–308
4. Archip, N., Clatz, O., Whalen, S., Kacher, D., Fedorov, A., Kot, A., Chrisochoides, N., Jolesz, F., Golby, A., Black, P.M., Warfield, S.K.: Non-rigid alignment of preoperative MRI, fMRI, and DT-MRI with intra-operative MRI for enhanced visualization and navigation in image-guided neurosurgery. NeuroImage (2007) (in press).
5. Chrisochoides, N., Fedorov, A., Kot, A., Archip, N., Black, P., Clatz, O., Golby, A., Kikinis, R., Warfield, S.K.: Toward real-time image guided neurosurgery using distributed and Grid computing. In: Proc. of IEEE/ACM SC06. (2006)
6. Blumofe, R., Joerg, C., Kuszmaul, B., Leiserson, C., Randall, K., Zhou, Y.: Cilk: An efficient multithreaded runtime system. In: Proceedings of the 5th Symposium on Principles and Practice of Parallel Programming. (1995) 55–69
7. Wu, I.: Multilist Scheduling: A New Parallel Programming Model. PhD thesis, School of Comp. Sci., Carnegie Mellon University, Pittsburg, PA 15213 (July 1993)
8. TeraGrid Project: TeraGrid Home page (2006) http://teragrid.org/, accessed 23 April 2006.
9. Beckman, P., Nadella, S., Trebon, N., Beschastnikh, I.: SPRUCE: A system for supporting urgent high-performance computing. In: Proc. of WoCo9: Grid-based Problem Solving Environments. (2006)