Non-Rigid Registration for brain MRI: faster and cheaper

Yixun Liu*

Department of Computer Science,
College of William and Mary,
Williamsburg, VA 23185, USA
E-mail: yixun@ieee.org

Andriy Fedorov and Ron Kikinis

Surgical Planning Laboratory,
Brigham and Women's Hospital,
Boston, MA 02115, USA
E-mail: fedorov@bwh.harvard.edu
E-mail: kikinis@bwh.harvard.edu

Nikos Chrisochoides*

Department of Computer Science,
College of William and Mary,
Williamsburg, VA 23185, USA
E-mail: nikos@cs.wm.edu
*Corresponding authors

Abstract: We study the problem of Non-Rigid Registration (NRR) for intra-operative recovery of brain shift during image-guided neurosurgery. Time-critical nature of the tumour resection procedure presents a major obstacle to the routine clinical use of many available NRR approaches. In this paper, we utilise the resources of a single multicore workstation with an advanced graphics card to parallelise and evaluate an end-to-end implementation of a clinically validated NRR method. The results on clinical brain MRI data show the parallel NRR can reach real-time clinical requirement.

Keywords: non-rigid registration; GPU; multicore; real-time; brain shift.

Reference to this paper should be made as follows: Liu, Y., Fedorov, A., Kikinis, R. and Chrisochoides, N. (2010) 'Non-Rigid Registration for brain MRI: faster and cheaper', *Int. J. Functional Informatics and Personalised Medicine*, Vol. 3, No. 1, pp.48–57.

Biographical notes: Yixun Liu is a PhD Candidate. Currently, he is doing his PhD in the Center for Real-time Computing of Computer Science Department at William and Mary. His areas of interest are medical image computing, brain shift in image-guided neurosurgery, and parallel computing.

Andriy Fedorov received his PhD Degree in Computer Science from The College of William and Mary in 2009. He is currently a Research Fellow within Surgical Planning Laboratory at Brigham and Women's Hospital and

Harvard Medical School. His research interests are in medical image computing with the focus on the development of practical methods and software tools for clinical research applications.

Ron Kikinis, MD, is the founding Director of the Surgical Planning Laboratory, Department of Radiology, Brigham and Women's Hospital, Harvard Medical School, and a Professor of Radiology at Harvard Medical School. He is the Principal Investigator of the National Alliance for Medical Image Computing and of the Neuroimage Analysis Center. He is also the Research Director of the National Center for Image Guided Therapy. His activities include technological research (segmentation, registration, visualisation, high performance computing), software system development (most recently the 3D Slicer software package), and biomedical research in a variety of biomedical specialties.

Nikos Chrisochoides is Full Professor, John Simon Guggenheim Fellow in Medicine and Health, Founder and Director of the Center for Real-Time Computing, and Co-founder of the Medical Imaging Software Technologies LLC. His research interests are in medical image computing and parallel scientific computing, specifically, parallel mesh generation on both theoretical and implementation aspects. His research is application-driven. Currently, he is working on real-time mesh generation for biomedical applications like non-rigid registration for Image Guided Neurosurgery.

1 Introduction

Local deformations of soft tissue during image-guided clinical interventions (e.g., tumour resection, ablation and biopsy) compromise the accuracy of targeting in the cases when the decisions are made based solely on the preoperative data. NRR is an image-processing technique that allows accounting for such deformation by utilising some form of intraoperative imaging. Estimation of the transformations that recover these deformations is a computationally intensive task, which is often prohibitive for its intraoperative application.

Several groups made an attempt to use GPU to accelerate NRR. Levin et al. (2004) implemented a high-performance Thin Plate Spline (TPS) volume-warping algorithm by combining hardware-accelerated 3D textures, vertex shaders and trilinear interpolation. Ruiz et al. (2008) used polynomial mapping as non-rigid transformation and achieved a factor of 4.11 speedup with a single GPU and 6.68 with a GPU pair over CPU-based NRR. A more complex physics-based registration method, previously developed and evaluated in Clatz et al. (2005), Chrisochoides et al. (2006) and Archip et al. (2007), has been parallelised by Chrisochoides et al. (2006) within 5 minutes on a distributed computing infrastructure with hundreds of computing nodes. However, the use of such distributed computing resources is often difficult or not feasible in clinical environment owing to network firewalls, lack of available resources, or lack of network connectivity in the Operating Room (OR).

In this paper, we present a new approach to parallelise NRR method in Chrisochoides et al. (2006). Specifically, in order to improve the overall performance we

- Use new fine-grain parallel approach for the efficient execution of the Block Matching (BM) component on GPU
- Introduce a parallel finite element discretisation and indexing scheme (Chrisochoides et al., 1994) that can minimise expensive gather/scatter operations in the linear iterative solver
- Employ an efficient public domain parallel linear solver, PETSc (http://www.mcs.anl.gov/petsc/petsc-as/).

Our work makes the following contributions:

- Reduce the end-to-end response time of an accurate and clinically tested NRR method by more than three times to better meet real-time clinical requirements for image-guided neurosurgery
- Use 10 times less-expensive computing platform (compared with traditional CoWs) by taking advantage of disruptive and inexpensive technologies like GPUs with limited if not zero systems administration support
- Increase reliability and flexibility by bringing, for the first time, the solution closer to the OR.

Our overall goal is acceleration of transitioning research results in NRR into the clinical use.

2 Methods

2.1 Non-Rigid Registration approach

The objective of NRR is to find a transformation of the preoperative (floating) image so that it is aligned with the intraoperative (reference) image. The approach is based on the concept of energy minimisation. A sparse set of registration points within the preoperative brain MRI is identified. The displacement between the pre- and intraoperative images is estimated using BM (Bierling, 1988) at each such point. On the basis of these displacements, the deformation field defined at mesh nodes is estimated under the constraint of a biomechanical model. Registration is formulated as a minimisation problem of the energy:

$$W = (\mathbf{HU} - \mathbf{D})^T \mathbf{S}(\mathbf{HU} - \mathbf{D}) + \mathbf{U}^T \mathbf{KU} \tag{1}$$

where **K** is the stiffness matrix, **H** is the linear interpolation matrix from the displacements recovered by BM and those at the mesh vertices, **S** is the BM weight matrix, **D** contains BM displacements, and **U** is the unknown displacement vector at the mesh vertices.

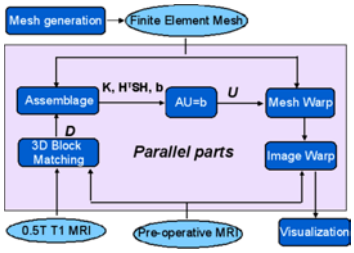
Regularisation of the solution using the biomechanical energy unavoidably contains approximation error (Clatz et al., 2005). We address this problem by iterative estimation of the displacement

$$\begin{aligned} &\mathbf{F}_0 = 0, \quad \mathbf{F}_{i-1} = \mathbf{KU}_{i-1} \\ &[\mathbf{K} + \mathbf{H}^T \mathbf{SH}]\mathbf{U}_i = \mathbf{H}^T \mathbf{SD} + \mathbf{F}_{i-1}. \end{aligned} \tag{2}$$

This iterative method reduces the approximation error at each iteration, while rejecting outlier registration points. In the remainder of the paper, we call equation (2) incremental solver owing to its incremental improvement of the accuracy. Such formulation is robust to outliers and allows to reduce the approximation error at the expense of longer execution time.

The NRR method contains two computationally intensive components: BM and the finite element solver. Our parallel NRR framework is shown in Figure 1. We use GPUs, which are designed to perform bulk computations of a kernel code on different input data, for BM. The finite element solver operates on irregular data structures requiring synchronisation and communication. Such computations cannot fully benefit from the GPU architecture (http://www.intel.com/research/), therefore we develop a multicore implementation of the solver. Both GPU and multicore processors can be found on current and emerging high-performance computing platforms.

Figure 1 Parallel NRR framework (see online version for colours)



2.2 GPU implementation of 3D Block Matching

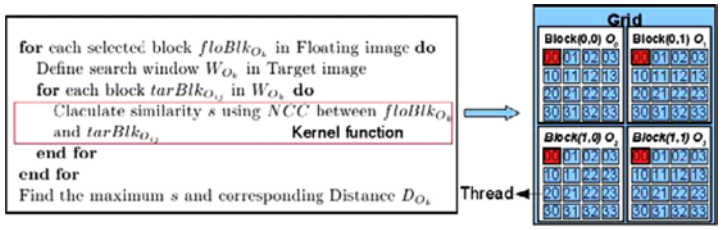
The NVIDIA Compute Unified Device Architecture (CUDA) (NVIDIA, 2008) abstracts GPU as a general-purpose multithreaded SIMD (single instruction, multiple data) architectural model, offering a C-like interface supported by a compiler and a runtime system for GPU programming. CUDA programming model organises threads in a grid, which is an array of blocks and each block is an array of threads, see Figure 2. All blocks in a grid have the same number of threads. Each thread block can be uniquely identified using two-dimensional coordinates given by the CUDA-specific keywords: *blockIdx.x*, *blockIdx.y*. Each thread block is in turn organised as a three-dimensional array of threads. The coordinates of threads in a block are uniquely defined by three thread indices: *threadIdx.x*, *threadIdx.y* and *threadIdx.z*.

BM is a well-known technique used originally for recovering motion from images (Bierling, 1988). BM is based on the assumption that a complex non-rigid transformation can be approximated by point-wise translations of small image regions. Such a translation can be recovered at a point of the floating image by selecting a block of voxels $B(O_k)$ centred around a point O_k, and searching for such a position of the block that maximises some similarity metric $M(B(O_a), B(O_b))$ with respect to the corresponding part of the target image window W_k. The similarity metric depends on the application, with Normalised Cross Correlation (NCC) suitable for registering mono-modal data.

The key to efficient GPU implementation is in the mapping of a sequential code to CUDA programming model, which is demonstrated in Figure 2. For each registration point and the corresponding block in the floating image, we assign a separate CUDA thread to calculate its similarity with a different portion of the search window. The size of

the portion depends on the size of the thread block and the search window. For instance, if the thread block is $4 \times 4 \times 4$ and search window is $8 \times 8 \times 8$, each thread will be responsible for the calculation of the similarity within a $2 \times 2 \times 2$ portion. The maximum similarity can be evaluated by parallel reduction of the computed similarity values.

Figure 2 Mapping of sequential BM to GPU programming model. Left: Sequential BM and Right: GPU BM (see online version for colours)



Once the mapping of the sequential algorithm to the CUDA architecture is identified, the challenge is in the selection of the implementation parameters. The two parameters in our GPU implementation are the image block size and thread block size. The former affects the precision of BM, whereas the latter, also known as GPU execution configuration, can significantly impact the performance of the code. As an example, 6 out of 7 benchmarks from NVIDIA SDK gain speedup ranging from 1.5 to 6.6 times when execution configuration is optimised (Liu et al., 2009).

The size of the execution configuration search space is so large that it is not practical to find the optimal parameter by manual tries. The details on tuning these parameters were described in Liu et al. (2009). We summarise the optimal settings for thread block given the sizes of the image block and search window in Figure 3.

Figure 3 Optimal GPU execution configuration for <*imageblocksize, windowsize*>

Block \ Window	11×11×19	13×13×13	15×15×15	17×17×17	19×19×19
7×7×7	<4×4×4>	<8×8×2>	<8×8×2>	<4×4×4>	<4×4×4>
9×9×9	<4×4×4>	<8×8×2>	<8×8×2>	<4×4×4>	<4×4×4>
11×11×11	<4×4×4>	<8×8×2>	<8×8×2>	<4×4×4>	<4×4×4>
13×13×13	<4×4×4>	<8×8×2>	<8×8×2>	<4×4×4>	<4×4×4>

2.3 Multicore implementation of incremental solver

The matrices used by the formulation of the incremental solver in equation (3) are derived from the finite element mesh and registration points. The objective of the parallelisation is to distribute the mesh and registration points among cores, and derives the solution to the linear system of equations in parallel.

We employ the ParMETIS library (http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview) to implement balanced parallel partitioning of the tetrahedral mesh among the processing cores. This procedure is illustrated in Figure 5. By following this partitioning strategy, we minimise the number of the interface elements to reduce communication, which is done via MPI. ParMETIS starts with an initial partitioning of the input mesh as shown in Figure 5(a). In this example, there are two cores, with the assigned mesh elements marked by green and red colours. Initially, 'green' core holds elements 0, 1 and 2 and 'red' core holds element 3, 4 and 5.

On the basis of the initial partitioning, ParMETIS generates a mapping between the elements and the cores to minimise the number of the interface elements. In Figure 5, this mapping assigns elements 0, 1 and 3 to the 'green' core, and elements 2, 4 and 5 to the 'red' core. On the basis of the produced mapping, mesh elements along with the corresponding registration points will be reassigned to the corresponding cores, as shown in the final partitioning. An example of the 4-way partitioned mesh (the number of cores on a typical workstation) is given in Figure 4.

Figure 4 4-way partitioned mesh and registration points (see online version for colours)

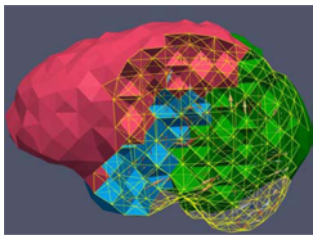
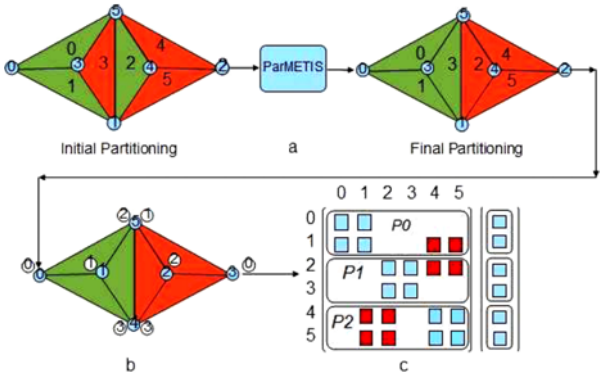


Figure 5 Partitioning and renumbering: (a) partitioning; (b) renumbering and (c) arrowhead pattern matrix and its distribution across cores: P0, P1 and P2 (see online version for colours)



Following the partitioning, vertices in each sub-mesh must be renumbered contiguously. We use local numbering strategy and let each core keep a mapping table to relate the local numbering with the global numbering. In Figure 5(b), the numbers in blue circles define global numbering, and the numbers in white circles correspond to local numbering. The advantage of this approach is that the sub-mesh can be considered as a separate mesh on each of the cores, while the communication with the non-local sub-meshes is facilitated by the mapping table. After partitioning and renumbering, we can construct a desirable arrowhead matrix, as shown in Figure 5(c), which is efficient in evaluating matrix-vector products – a major computation component of CG solver (Saad, 2003). The reason is that most computations can be performed locally, and only the submatrix marked with red colour need to communicate with the other cores (Chrisochoides, 1995).

On the basis of equation (3), we need to assemble the matrices $\mathbf{K}$ and $\mathbf{H}^{\mathrm{T}}\mathbf{SH}$. Construction of the stiffness matrix $\mathbf{K}$ has been well documented elsewhere (Bathe, 1996). To improve the performance of assembling stiffness matrix $\mathbf{H}^{\mathrm{T}}\mathbf{SH}$, we directly set

the values at its corresponding entries instead of assembling **H** and multiplying its transpose with **S** and **H**. Each registration point k (i.e., one block centre O_k) contained in tetrahedron with vertex (v_0, v_1, v_2, v_3) will contribute to $\mathbf{H}^{\mathrm{T}}\mathbf{SH}$ at position (v_i, v_j), $i, j \in [0:3]$ with the submatrices $\mathbf{H}_{vi}^{T}\,\mathbf{S}_k\mathbf{H}_{v_j} = h_i \times \mathrm{S}_k \times h_j$, in which $\mathbf{S}_k$ are 3×3 confidence submatrix and $h_j, j \in [0:3]$ are linear interpolation factors (Clatz et al., 2005).

After assembling the matrices, we have a linear system $AU = b$, where A, U, b are distributed across cores as shown in Figure 5(c). A is a semi-positive definite matrix, and we use Conjugate Gradient (CG) solver to find the solution. This component is also computed in parallel, facilitated by the PETSc implementation of the CG solver (http://www.mcs.anl.gov/petsc/petsc-as/).

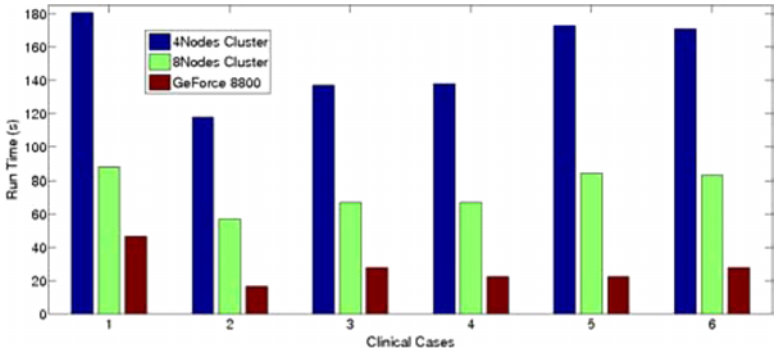
3 Results

3.1 Block Matching results

We compare the performance of the BM on a typical modern workstation equipped with NVIDIA GeForce 8800 GT GPU with its MPI implementation running on an 8-node cluster (each node is Dell PowerEdge SC1435, 2 x dual-core Opteron 2218, 2.6 GHz CPU). The results were collected for computations on 6 retrospective brain tumour resection cases.[1]

Figure 6 shows the comparison of performance for the considered implementations. Compared with the 4-node cluster (16 cores), the minimum speedup is 3.9 (case 1) and the maximum speedup is 7.7 (case 5). Compared with the 8-node cluster (32 CPUs), the minimum speedup is 1.9 (case 1) and the maximum speedup is 3.8 (case 5).

Figure 6 Performance evaluation using six existing retrospective data from Brigham and Women's hospital. Image block: $9 \times 9 \times 9$, search window: $11 \times 11 \times 19$, optimal thread block (obtained from Figure 3): $4 \times 4 \times 4$ (see online version for colours)



3.2 Incremental solver results

The processor used for the experiment is $2 \times$ Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16 GHz. The runtime of the solver depends on the size of the mesh. Three different sizes of the mesh ranging from small, middle to large were generated based on case 6 using sequential mesh generator RGM (Fedorov et al., 2005) developed in our group. A thoroughly evaluated biconjugate gradient solver implemented within Gmm++ library

(http://home.gna.org/getfem/gmm_intro), which was used in Clatz et al. (2005), Chrisochoides et al. (2006) and Archip et al. (2007), is used for comparison with our parallel incremental solver. The runtime required for partitioning, matrix and vector assembling, and incremental solver is listed in Table 1.

Table 1 Performance comparison between sequential and parallel solver

Mesh		*Sequential* (time: second)		*Parallel* (time: second)			
Vertices	*Tetras*	*Assemblage*	*Solver*	*Partition*	*Assemblage*	*Solver*	*Speedup*
1607	7272	6.780	23.560	0.040	0.450	2.500	10.15
3526	17137	7.500	31.280	0.10	0.43	3.10	10.68
6737	33931	8.040	43.520	0.12	0.49	4.66	9.78

Compared with the sequential solver, our parallel solver needs additional partitioning time, but the overall gain in performance is significant.

The above-mentioned results demonstrate the desirable speedup brought by this parallel solver, but they cannot truly reveal the performance of the parallel solver. Obviously, simply comparing with the number of the cores is not reasonable too. Salomon et al. (2005) proposed a reasonable method to evaluate the performance by comparing the relative speedup with its upper bound given by Amdahl's law. The sequential parts in FE solver are mesh loading, ParMETIS initiation (including partitioning initiation and element graph creation), which accounts for about 9% of the total computation. It can be calculated from Table 2. The upper bound for 4 cores can be calculated as: 100/9+91/4 = 3.2. The relative speedup is about 2.2 for large mesh, which is obtained by calculating the ratio between the execution time on one core and four cores from the data in Table 2. As we can see, the relative speedup is close to its upper bound.

Table 2 Performance evaluation for parallel solver on large mesh. Mesh loading and ParMETIS initiation are performed using one core

No. of Cores	*Mesh load*	*InitMetis*	*Partition*	*Assemble*	*Solve*	*Total time*
1 Core	1.02	0.04	0.14	1.05	11.25	13.5
4 Cores	1.00	0.03	0.09	0.49	4.66	6.27

4 Discussion

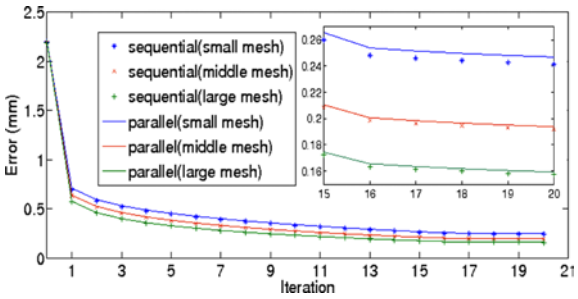
In this paper, a parallel NRR technique is developed by making full use of the computation power of a single multicore server or desktop. As confirmed by our experimental evaluation on real data, by using the multicore and GPU of a conventional, highly affordable workstation, we can deliver the result of the computation within 36 s (case 6). In comparison, the time required for NRR of the same data on an 8-node cluster with 32 cores is about 135 s.

The important consideration in developing efficient GPU implementation is the optimal execution configuration. Our initial selection of the thread block $8 \times 8 \times 2$ proved to be four times slower than the optimal $4 \times 4 \times 4$ thread block. The speedup of the parallel solver tends to decrease with larger meshes. This is due to the increasing overhead of the mesh partitioning. This overhead can be eliminated (in the future) by

fully parallelising the finite element mesh generation phase (Chrisochoides, 2005). Large finite element meshes are required to satisfy the accuracy requirements in the clinic.

We note that because we use GPU for BM and multicore for solver, the numerical solution is not identical to the one derived with the original code. The differences are due to the lack of real support for double precision in the available GPUs (only new G280 supports double precision in hardware), and due to very small round-off errors introduced when computations in the parallel linear solver (due to concurrency) are performed in a different order. As we show in Figure 7, these differences are negligible.

Figure 7 Precision evaluation for different mesh. Middle figure shows the zoom in of iteration from 15 to 20 (see online version for colours)



Overall, as confirmed by our study, complex physics-based NRR methods are now ready for wide application in the OR without requiring distributed or cluster computing resources. However, to achieve this performance, careful design of GPU mapping and optimisation of the GPU execution environment is required. In our future work, we can further improve the performance of the solver by achieving better load balancing and apply transformation that will completely eliminate expensive gather/scatter operations by augmenting the linear system without increasing redundant computations. In addition, the current and future improvements in the execution time of the finite element solver will allow for dynamic mesh refinement, which will adjust the mesh based on the rejection of outliers.

Acknowledgement

This work is supported in part by NSF grants: CCF-0916526, CCF-0833081, and CSI-719929 and by the John Simon Guggenheim Foundation. Andriy Fedorov was partially supported by NIH grant R01AA016748. Ron Kikinis was partially supported by RR13218, EB005149.

References

Archip, N., Clatz, O., Whalen, S., Kacher, D., Fedorov, A., Kot, A., Chrisochoides, N., Jolesz, F., Golby, A., Black, P. and Warfield, S. (2007) 'Non-rigid alignment of preoperative MRI, fMRI, and DT-MRI with intra-operative MRI for enhanced visualization and navigation in image-guided neurosurgery', *Neuroimage*, Vol. 35, No. 2, pp.609–624.

Bathe, K. (1996) *Finite Element Procedure*, Prentice-Hall, New Jersey.

Bierling, M. (1988) 'Displacement estimation by hierarchical block matching', *Proc. SPIE Vis. Comm. and Image Proc.*, Vol. 1001, pp.942–951.

Chrisochoides, N. (1995) *Multithreaded Model for Dynamic Load Balancing Parallel Adaptive PDE Computations*, Technical Report.

Chrisochoides, N. (2005) 'Parallel mesh generation', in Bruaset, M., Bjorstad, P. and Tveito, A. (Eds.): *Numerical Solution of Partial Differential Equations on Parallel Computers*, Vol. 51, pp.237–259.

Chrisochoides, N., Elias, H. and John, R. (1994) 'Mapping algorithms and software environment for data parallel PDE iterative solvers', *J. Parallel Distrib. Comput.*, Vol. 21, No. 1, pp.75–95.

Chrisochoides, N., Fedorov, A., Kot, A., Archip, N., Black, P., Clatz, O., Golby, A., Kikinis, R. and Warfield, S. (2006) 'Toward real-time image guided neurosurgery using distributed and Grid computing', *Proc. of IEEE/ACM SC06*, Tampa, Florida.

Clatz, O., Delingette, H., Talos, I.F., Golby, A., Kikinis, R., Jolesz, F., Ayache, N. and Warfield, S. (2005) 'Robust non-rigid registration to capture brain shift from intra-operative MRI', *IEEE Trans. Med. Imag.*, Vol. 24, No. 11, pp.1417–1427.

Fedorov, A., Chrisochoides, N., Kikinis, R. and Warfield, S.K. (2005) 'Tetrahedral mesh generation for medical imaging', *8th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2005)*, Palm Springs, California, USA.

Levin, D., Dey, D. and Slomka, P. (2004) 'Acceleration of 3d, nonlinear warping using standard video graphics hardware: implementation and initial validation', *Comput. Med. Imaging Graph*, Vol. 28, pp.471–483.

Liu, Y., Zhang, E.Z. and Shen, X. (2009) 'A cross-input adaptive framework for GPU programs optimization', *23rd IEEE IPDPS*, Rome, Italy, pp.1–10.

NVIDIA (2008) *Cuda Programming Guide 2.0*.

Ruiz, A., Ujaldon, M., Cooper, L. and Huang, K. (2008) 'Non-rigid registration for large sets of microscopic images on graphics processors', *J. Sign Process Syst.*, Vol. 55, Nos. 1–3, April, pp.229–250.

Saad, Y. (2003) *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Salomon, M., Heitz, F., Perrin, G.R. and Armspach, J.P. (2005) 'A massively parallel approach to deformable matching of 3d medical images via stochastic differential equations', *Parallel Computing*, Vol. 31, No. 1, pp.45–71.

Note

[1]http://www.spl.harvard.edu/publications/item/view/541

Websites

Gmm++: http://home.gna.org/getfem/gmm_intro

Intel: Ct: A flexible parallel programming model for tera-scale architectures http://www.intel.com/research/

ParMETIS: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

PETSc: http://www.mcs.anl.gov/petsc/petsc-as/