In Proceedings of the IEEE Biomedical Circuits and Systems Conference,
pages 53-56, Baltimore, Maryland, November 2008.

Towards Systematic Exploration of Tradeoffs for Medical Image Registration on Heterogeneous Platforms

William Plishker[1,2], Omkar Dandekar[1,2], Shuvra S. Bhattacharyya[2], and Raj Shekhar[1,2],
[1]Radiology Department,
University of Maryland Medical College,
Baltimore, Maryland, USA
plishker@umd.edu

[2]Department of Electrical and Computer Engineering
University of Maryland, College Park
College Park, Maryland, USA

***Abstract*—For the past decade, improving the performance and accuracy of medical image registration has been a driving force of innovation in medical imaging. The ultimate goal of accurate, robust, real-time image registration will enhance diagnoses of patients and enable new image-guided intervention techniques. With such a computationally intensive and multifaceted problem, improvements have been found in high performance platforms such as graphics processors (GPUs) and general purpose clusters, but there has yet to be a solution fast enough and effective enough to gain widespread clinical use. In this study, we examine the differences in accuracy and speed of implementations of the same image registration algorithm on a general purpose uniprocessor, a GPU, and a cluster of GPUs. We utilize a novel domain specific framework that allows us to simultaneously exploit parallelism on a heterogeneous platform. Using a set of representative images, we examine implementations with speedups of up to two orders of magnitude and accuracy varying from sub-millimeter to 2.6 millimeters of average error.**

I. Introduction

Advances in medical imaging technologies have enabled medical diagnoses and procedures simply not possible a decade ago. The accelerating speed of acquisition and the increasing resolution of images have given doctors more information, less invasively about their patients. However, because of the multitude of imaging modalities (e.g. computed tomography (CT), positron emission tomography (PET), magnetic resonance (MRI), ultrasound (US), etc.) and the sheer volume of data being acquired, utilizing this new data effectively has become problematic. To tap into the potential of this raw data, these images can be merged into one integrated view through a procedure called *image registration*.

Image registration is the process of combining two images such that the features in one image are aligned with the features in another. The first phase of most registration techniques is correcting for whole image misalignment, called *rigid registration*. Nonunifrom correction may follow where non-linear local deformation from breathing or changes over time is aligned using *non-rigid registration.* While automatic registration algorithms exist that are robust, they tend to be computationally intensive, typically taking minutes to solve on high end processors for rigid registration and hours for non-rigid. Such complexity has inhibited the adoption of registration technology in the clinical workflow. While specific requirements vary from application to application, real-time registration must be on the order of seconds to be viable in most image-guided intervention scenarios.

To bring more accurate and more robust image registration algorithms into the clinical setting, a significant body of research has been dedicated to acceleration techniques for image registration. Graphics processors (GPUs) have been utilized in various image registration algorithms [1,2,3]. Clusters and other multiprocessor systems are often the target of higher level parallelism [4,5]. The Cell processor has been used to accelerate rigid registration using mutual information with additional speedup from just processing a subset of the total voxels [6]. Another important platform to consider is custom hardware (such as a field programmable gate array (FPGA)), which is suitable for exploiting much of the lowest levels of parallelism [7,8].

While many of these works have shown important performance results, no medical image registration work has systematically assessed the performance and accuracy tradeoffs for a set of platforms, especially heterogeneous ones. Since no single technique has accelerated registration sufficiently for all clinical applications, we believe real-time image registration will require a combination of targets that can each accelerate different parts of the application. In this work, we make a side-by-side comparison of platforms of interest, including GPUs, a general purpose processor (CPU), and a cluster of GPUs. In order to make a fair comparison between platforms, we implement the same algorithm on each and carefully tailor each implementation to the target platform. While other works target parallelism on heterogeneous platforms exist [9], to our knowledge only domain and target specific approaches have been used for medical image registration acceleration.

To realize implementations that utilize parallelism in different ways, we pay special attention throughout the design process on exposing and organizing application parallelism in a methodical way. To this end, our implementations are created in the context of a novel framework based on a previously published image registration taxonomy [10]. We employ this framework on a commonly used rigid and a non-rigid registration algorithm. The remainder of this paper covers background related to image registration acceleration, describes our implementations, present our experiments, and discusses what is needed to achieve of real-time registration.

978-1-4244-2879-3/08/$25.00 ©2008 IEEE

II. High Speed Image Registration

Intensity based image registration algorithms rely on correlations between voxel (3D pixel) intensities and are known to be robust and computationally intensive. A transformation is often described as a deformation field, in which all parts of the image to be deformed (the *floating image*) have a specific deformation such that they align with the other image (the *reference image*). Construction of the deformation field can start from a just few parameters in the case of rigid registration or from a set of *control points* which capture the non-uniformity of non-rigid registration. The final transformation contains the information necessary to deform all of the voxels in the floating image. Once a transformation is constructed, it is applied to the floating image. This transformed image can be compared to the reference image using a variety of similarity metrics, such as mean squared difference (MSD) or mutual information. For iterative approaches, the similarity value is returned so that it may guide the optimization engine towards successively better solutions. Problem parameters may change during run-time to improve speed and accuracy.

While image registration is a computationally intensive problem, it can be readily accelerated by exploiting parallelism. Researchers have applied a variety of innovative approaches at different levels of the problem. Enhancements that focus on acceleration through parallelism can be binned into levels based on the basic unit of computation considered. These include optimization level parallelism (for those techniques that parallelize the application at the iteration level), volume level parallelism (for independent volume calculations), subvolume level parallelism (for parallel work done on subvolumes), voxel level parallelism (for parallelism between voxels), and operation level parallelism (for any arbitrary operation that may occur during processing a voxel).

III. Heterogeneous Acceleration Implementations

A. Image Registration Implementation Framework

To properly conduct a study of accuracy versus speed trade-offs with such diversity of parallelism available in the application, we construct a novel design framework customized to image registration. If we were just dealing with individual platforms, we could simply implement each acceleration technique into the code base as necessary. But since we are experimenting with combined approaches (estimated to have high performance synergy [11]), we start by expressing the different types of parallelism in a structured fashion (see Fig. 1). After exposing and categorizing application parallelism, we map these to architectural primitives as presented by the programming models of our respective targets. For example, a message passing interface (MPI) implemented on a cluster is appropriate to support subvolume parallelism, while GPUs are a natural fit for subvolume and voxel level parallelism.

After identifying this mapping, we employ the tools and design principles specific to the target platform component. For instance, a GPU's array of pixel processing elements may be abstracted by the programming model as a set of threads as with the Compute Unified Device Architecture (CUDA) [12]. In Fig. 1, the *Linear Transformation* box in the *Rigid* application pictorially represents an assignment of a rigid registration's linear transformations to CUDA threads. Similar to the denotation of the mapping of computation, we represent the mapping of application communication to an architectural primitive. For a non-rigid registration algorithm, the gradient computation for a *free-form deformation* (FFD) grid can be readily accelerated using a general purpose cluster [4] and on a GPU. We show our combination of these two approaches on the non-rigid side of Fig. 1.

B. Implementations on Multiple Platforms

Based on the structure from the framework and insights of the previous section, we implement the same image registration algorithm on a uniprocessor, a GPU, and a GPU cluster. For rigid registration, the optimization method is based on simplex [13] and the similarity metric is mean squared difference (MSD) with nearest-neighbor interpolation. The non-rigid algorithm is based on a gradient descent method [14] with an FFD grid utilizing B-spline interpolation between control points and trilinear interpolation for voxels.

For our rigid algorithm, parallelism comes from the independence of the mean squared difference calculation performed on separate subvolumes. Large subvolumes are a good match to the granularity of MPI nodes and small volumes map naturally to CUDA *blocks* (an abstraction of GPU pixel multiprocessors), so both the GPU and the cluster can exploit the similarity calculation parallelism. Each can be used for single platform acceleration but also combined by making the MPI subvolumes large enough to be divided into smaller subvolumes used by the GPU.

For our non-rigid implementation, we utilize the subvolume level parallelism of the gradient calculation for each control point in MPI. The gradient calculation using finite difference is made up of multiple similarity calculations with addition of B-spline interpolation between control points to determine local deformation for a given floating image voxel. A GPU can effectively accelerate this calculation by utilizing cooperative multithreading: mapping plane interpolation to a set of threads, row interpolation to a subset

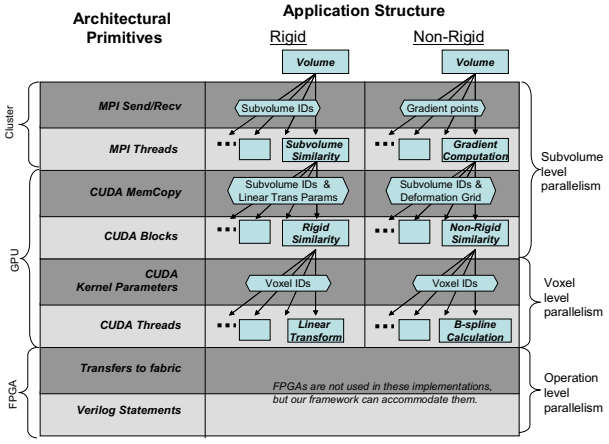
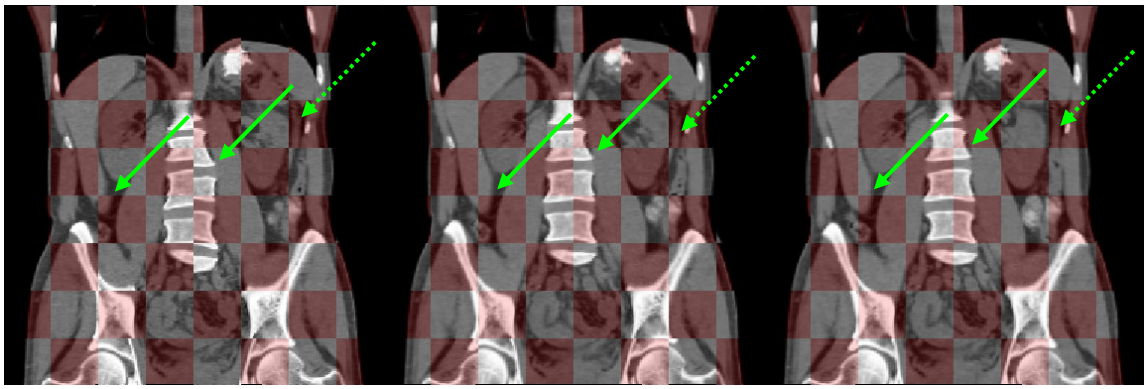


Fig. 1. A rigid and a non-rigid registration algorithm described in our framework targeting a cluster of graphics processors.



(a) (b) (c)

Fig. 2. Example registration (Case 1) of an image and a non-rigidly deformed version of it fused with a checkerboard pattern: uncorrected (a), GPU(s) corrected (b), and CPU corrected (c)

of the same threads, and finally the point interpolation to a single thread. As with rigid, the separation of these two parallelism constructs inside of a structured framework allows us to utilize them on individual platforms as well as on a combined heterogeneous platform.

IV. Experimental Setup

Using a single code base, we incorporated acceleration techniques wrapped by preprocessing directives. At compile time the software could be targeted for a specific parallel platform: a general purpose CPU, a GPU, or the GPU cluster. The CPUs used were 3GHz Intel Xeons, each with 4GB of RAM. The GPUs tested were NVIDIA 8800 GTXs targeted with CUDA SDK 2.0. The cluster was made up of 6 nodes, each with an 8800 and connected via Gigabit Ethernet. For rigid registration, the implementations were profiled with five pairs of CT images of the torso where the translation and rotation vectors were known. Each image was 256 × 256 × 256 with 8 bits representing voxel intensity. The rotation ranged between -25° and 25° on each axis and between -25 mm to 25 mm in each dimension.

With non-rigid, five new pairs were created by deforming a torso with a grid of size 5 × 5 × 5 overlaid. Each control point was randomly moved in each dimension by up to 2 cm in either direction. The grid used to correct this was of the same size. The image voxel size was 1.38 × 1.38 × 1.5 mm. The simplex and gradient descent parameters (starting position, initial step size, stopping criterion, etc.) were held constant across all cases and implementations. We constructed MPI subvolumes equal to the number of nodes in the cluster which made them large enough to be divided into GPU subvolumes to match the GPU blocks. The MPI reference image blocks were maximized to promote GPU utilization, divided into blocks of 8 × 8 × 4 for rigid registration on the GPU and 4 × 4 × 4 for non-rigid registration. The non-rigid blocks were smaller because more resources were used for the non-rigid registration similarity calculation, limiting the number of threads that could be assigned to one processing element of the GPU.

V. Results

While rigid registration results were of similar quality across platforms evaluated, non-rigid registration results varied. A typical case of one such variation is the cross-section shown in Fig. 2. Solid green arrows highlight how both the GPU and the CPU improve on the uncorrected fusion. Our GPU implementation currently uses single precision arithmetic optimized for graphics, while the CPU implementation utilizes double precision. As a result, the GPU is unable to produce the same quality of solutions in non-rigid registration. Dashed green arrows show misalignment in the uncorrected and GPU corrected images, but that is correctly aligned in the CPU corrected image. Since images were artificially deformed with a known deformation grid, we calculate the average and maximum distance between the known and recovered deformation over all voxels. The single GPU and the multiple GPU solution produced equivalent registration accuracy as MPI does not change the behavior of our implementation.

In an effort to produce a clearer picture of how well the kernels are accelerated, the timing results for rigid and non-rigid registration are reported in Table 1 and Table 2, respectively. They do not include the time for initialization, file IO, nor the one time image loads into GPU memory. Even on the most accelerated instances, the kernels examined consume the majority of the execution time, as it takes under 2 seconds for such overhead on each platform. The rigid registration speedup from a single GPU is almost 50x over a CPU as the core transformation matrix application to the image maps well to the GPU architecture.

The non-rigid single GPU attains comparable 20x average speedup over the CPU implementation due to our parallelization of the B-spline calculation that models the non-linear deformation. Unlike rigid registration, this acceleration comes at the expense of accuracy, producing an inferior registration result. In the GPU implementation, fewer steps are taken before the stopping criterion is met. The inability for a GPU to closely approach the same CPU solution is due to a difference in precision. When a control point is varied for its finite difference calculation, it makes only a minor change in the similarity that directs the subsequent step to be taken.

TABLE I. SPEED AND ACCURACY RESULTS OF RIGID REGISTRATION

	Case					Avg
	1	**2**	**3**	**4**	**5**	
CPU Accuracy: avg	0.17	0.10	0.15	0.09	0.12	**0.13**
(mm) max	0.24	0.14	0.18	0.13	0.15	**0.17**
GPU Accuracy: avg	0.04	0.10	0.16	0.09	0.12	**0.10**
(mm) max	0.05	0.14	0.21	0.13	0.15	**0.14**
CPU iterations	267	264	299	259	206	**259**
total time (s)	1,320	1,147	1,944	1,538	931	**1376**
1 GPU iterations	270	254	328	334	376	**312**
total time (s)	28.2	25.4	33.2	33.7	38.2	**31.7**
6 GPU time (s)	5.3	4.8	6.2	6.3	7.1	**5.9**
GPU step speedup relative to CPU	47	43	64	59	44	**52**
6 GPU step speedup relative to 1 GPU	5.3	5.3	5.4	5.3	5.4	**5.3**
6 GPU step speedup relative to CPU	**252**	**230**	**344**	**315**	**239**	**276**

Even though the GPU precisely calculates most of precision finite difference calculations, some subset of them is poorly estimated on some steps. Since all points advance simultaneously after the gradient is calculated, even a few wayward control points can significantly skew the similarity between the reference and floating image, inhibiting the overall convergence. An individual non-rigid step is still 20x faster on a single GPU compared to a CPU, but this is considerably less than the 50x of rigid registration. This is due to the increased thread coordination and resource usage of the non-rigid similarity calculation.

VI. CONCLUSION

We have presented our study of accuracy and speed of image registration as part of our ongoing effort to find better platforms and acceleration techniques for medical image registration. Utilizing a heterogeneous cluster of GPUs with a novel domain specific framework, we have measured over two orders of magnitude application speedup over a CPU implementation on representative registration cases. There is negligible loss in accuracy for rigid registration, but a CPU implementation does provide more accuracy for a commonly used gradient descent approach studied here. Towards our goal of robust real-time registration, we believe that the advantages of each implementation could be reaped by running different phases of image registration on different platforms (e.g. GPU first for a fast course solution, and CPU second for more accuracy, as the imaging scenario would permit). Alternatively, a different algorithm could be employed for the GPU that would be less sensitive to precision effects. Also the recent introduction of double precision data type into CUDA may also provide a relevant design point. We believe the structured approach presented here will enable our continued exploration into these and other implementations.

TABLE II. SPEED AND ACCURACY RESULTS OF NON-RIGID REGISTRATION

	Case					Avg
	1	**2**	**3**	**4**	**5**	
CPU accuracy: avg	0.64	0.64	0.53	0.66	0.72	**0.64**
(mm) max	4.91	2.91	2.74	3.57	3.84	**3.59**
GPU accuracy: avg	2.80	2.78	2.41	2.60	2.40	**2.60**
(mm) max	9.47	10.30	8.49	8.48	8.28	**9.00**
CPU iterations	46	45	34	39	34	**40**
total time (s)	47,923	54,455	37,149	43,444	55,662	**47,762**
1 GPU iterations	13	21	8	10	13	**13**
total time (s)	782	1,303	489	727	723	**805**
6 GPU time (s)	213	326	125	269	195	**206**
GPU step speedup relative to CPU	17.3	19.5	17.9	15.3	29.4	**20**
6 GPU step speedup relative to 1 GPU	3.7	4.0	3.9	2.7	3.7	**3.8**
6 GPU step speedup relative to CPU	**64**	**78**	**70**	**41**	**109**	**72**

ACKNOWLEDGEMENTS

This work was supported by the Department of Defense grant DAMD17-99-1-9034. We would like to express our thanks to the Imaging Technologies Laboratory and the DSPCAD group.

REFERENCES

[1] R. Strzodka, M. Droske, and M. Rumpf, "Fast image registration in dx9 graphics hardware. journal of medical informatics and technologies," vol. 6. pp. 43-49, 2003.

[2] A. Köhn, J. Drexl, et al., "GPU accelerated image registration in two and three dimensions," in Bildverarbeitung für die Medizin 2006, H. Handels, et al., Eds. 2006, Springer Berlin Heidelberg. pp. 261-265.

[3] D. Ruijters, B. Romeny, and P. Suetens, "Efficient GPU-accelerated elastic image registration," In Proc. Sixth IASTED International Conference on BIOMEDICAL ENGINEERING (BioMed), Innsbruck, Austria, pp. 419-424, February 2008.

[4] F. Ino, K. Ooyama, and K. Hagihara, "A data distributed parallel algorithm for nonrigid image registration," Parallel Computing, vol. 31(1), pp. 19-43, 2005.

[5] R. Stefanescu, X. Pennec, and N. Ayache, "Parallel non-rigid registration on a cluster of workstations," Proc. of HealthGrid'03, 2003.

[6] M. Ohara, H. Yeo, et al. "Real-time mutual-information-based linear registration on the cell broadband engine processor," In 4th IEEE ISBI, Arlington, VA, pp. 33-36, 2007.

[7] C.R. Castro-Pareja, J.M. Jagadeesh, and R. Shekhar, "FAIR: a hardware architecture for real-time 3-D image registration," Information Technology in Biomedicine, vol. 7(4), pp. 426, 2003.

[8] R. Shekhar, V. Zagrodsky, et al. "High-speed registration of three- and four-dimensional medical images by using voxel similarity," Radiographics, vol. 23(6), pp. 1673-81, 2003.

[9] C. Haubelt, J. Falk, J. Keinert, et al. "A SystemC-Based Design Methodology for Digital Signal Processing Systems," EURASIP J. on Embedded Systems, vol. 2007(1), pp. 15, 2007.

[10] W. Plishker, O. Dandekar, S.S. Bhattacharyya, and R. Shekhar, "A taxonomy for medical image registration acceleration techniques," in Proceedings of the IEEE-NIH Life Science Systems and Applications, Bethesda, MD, pp. 215-218, 2007.

[11] W. Plishker, O. Dandekar, S.S. Bhattacharyya, and R. Shekhar, "Towards a heterogeneous medical image registration acceleration platform," in Proceedings of the IEEE BioCAS, Montreal, Canada, pp. 231-234, 2007.

[12] NVIDIA, "NVIDIA CUDA, Compute Unified Device Architecture, Programming Guide," 2007.

[13] J.A. Nelder and R. Mead, "A simplex method for function minimization," The Computer Journal, vol. 7, pp. 308-313, 1964.

[14] D. Rueckert, L.I. Sonoda, et al. "Nonrigid registration using free-form deformations: application tobreast MR images," IEEE Transactions on Medical Imaging, vol. 18(8), pp. 712-721, 1999.