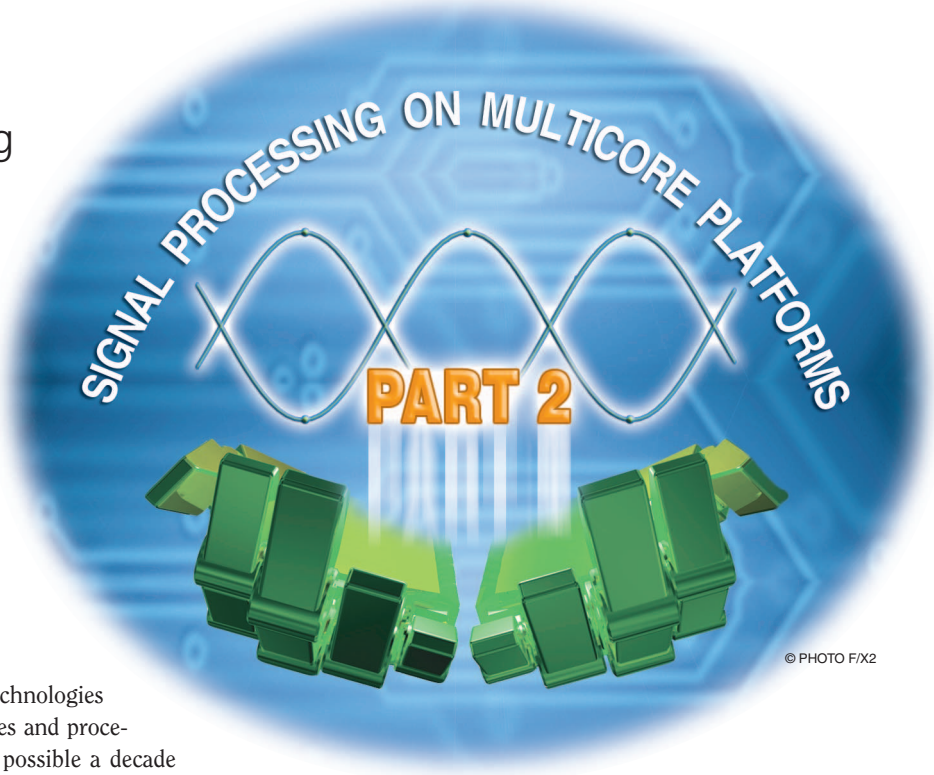
William Plishker, Omkar Dandekar, Shuvra S. Bhattacharyya, and Raj Shekhar

Utilizing Hierarchical Multiprocessing for Medical Image Registration

The possibilities and challenges of combining acceleration approaches that utilize complementary types of parallelism



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

Digital Object Identifier 10.1109/MSP.2009.935419

COMPUTE-ENABLED MEDICINE

For the past decade, improving performance and accuracy has been a driving force of innovation in automated medical image registration. 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 processing units (GPUs) and general-purpose multicore systems, but there has yet to be a solution fast enough and effective enough to gain widespread clinical use.

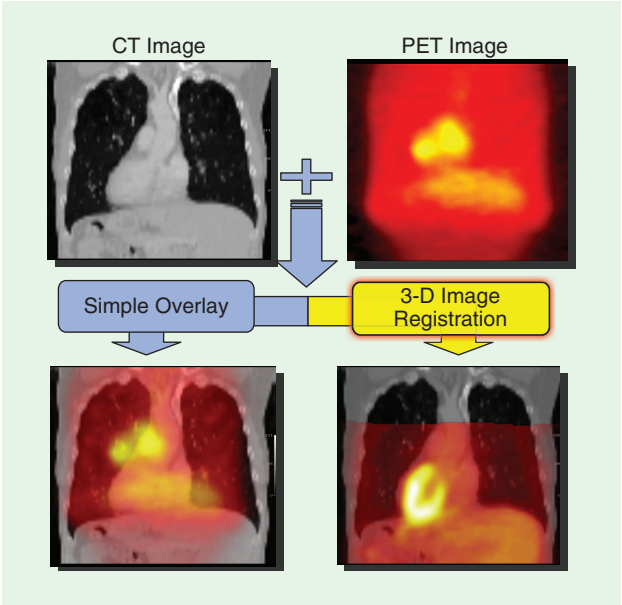
1053-5888/10/$26.00©2010IEEE

To achieve the necessary speed, we believe that a synergy of approaches will be needed, requiring many cores organized at different levels of granularity. We call such processing hierarchical multiprocessing, as it requires the use of multiple styles of parallelism to be properly utilized. To accelerate medical image registration, we explore some of the key issues of hierarchical multiprocessing by leveraging a novel domain-specific framework to design and implement an image-registration algorithm on a GPU and on a cluster of GPUs, and compare them in terms of speed and accuracy. Using a set of representative images, we achieve execution times as low as 2.5 s and accuracy varying from submillimeter to 2.4 mm of average error.

IMAGE REGISTRATION IS THE PROCESS OF COMBINING IMAGES SUCH THAT THE FEATURES IN AN IMAGE ARE ALIGNED WITH THE FEATURES OF ONE OR MORE OTHER IMAGES.

INTRODUCTION

Image registration is the process of combining images such that the features in an image are aligned with the features of one or more other images. An example pairing is shown in Figure 1. The first phase of most registration techniques is correcting for whole image misalignment, called rigid registration. Nonrigid registration often follows such that nonlinear local deformation from breathing or changes over time is corrected. While automatic and robust registration algorithms exist, they tend to be computationally intensive, often taking minutes to execute on high-end general-purpose processors for rigid registration and hours for nonrigid registration. 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.



[FIG1] A typical medical image registration case. The CT is the fixed image and the PET is the moving image. A simple overlay of these two gives a clinician little information about the case, but an accurately registered result overlays the metabolic information of PET on the structural information of CT.

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. A thorough discussion of this appears in the article "A Survey of Medical Image Registration on Multicore and the GPU," by R. Shams, et al. in this issue of *IEEE Signal Processing Magazine*. Many multicore platforms already in use for this purpose are GPUs, clusters of general-purpose processors, the Cell, and even custom hardware from implementations on field programmable gate arrays (FPGAs). While many of these works have shown performance improvements, no single technique has accelerated registration sufficiently for all clinical applications. We believe real-time image registration will require a combination of parallelism styles that can be used to accelerate different aspects of the application. Proper utilization of hierarchical platforms can lead to multiplicative effects from complementary acceleration techniques.

Utilizing parallelism for any single platform may be a challenging and time consuming implementation task. It carries the normal tasks of programming (e.g., creating a software architecture, developing algorithms, testing, debugging, and performance tuning), and the added difficulties of managing parallel threads (e.g., managing communication, debugging race conditions, and load balancing). Utilizing parallelism on a hierarchical multiprocessing platform is even worse, often involving multiple programming models and environments, which designers must decide how to use before beginning to write code. To facilitate a design process in which the structure of an application is considered when mapping to these diverse programming models, we leverage a novel framework based on an image registration specific taxonomy. To demonstrate the utility of this approach, we employ this framework on a commonly used a rigid and a nonrigid registration algorithm.

To explore the challenges and potential benefits of targeting hierarchical multiprocessing platforms for image registration, we study two in particular: a single GPU and a cluster of GPUs. Using these platforms, we took our single-threaded code base on a general-purpose processor and, for the most time-consuming kernels, added acceleration tailored to the target platform. In particular, we focus on utilizing parallelism described in our taxonomy. Based on these results, we discuss the possibilities and the challenges of combining acceleration approaches that utilize complementary types of parallelism. This article expands on preliminary work on this subject, which is presented in [1] and [2].

INTENSITY-BASED REGISTRATION

Intensity-based image registration algorithms rely on similarities between voxel [three-dimensional (3-D pixel)] intensities. These algorithms are known to be robust but tend to be computationally intensive. In an intensity-based image-registration

algorithm, a transformation is often described as a deformation field, in which all parts of the image to be deformed (the moving image) have a specific deformation such that they align with the other image (the fixed image). Construction of the deformation field can start from just a few parameters in the case of rigid registration or from a set of "control points" that capture the nonuniformity of nonrigid registration. The final transformation contains the information necessary to deform all of the voxels in the moving image. Once a transformation is constructed, it is applied to the moving image. This transformed image can be compared to the fixed 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 registration algorithm towards a solution. A variety of iterative optimization algorithms have been developed for medical image registration with different convergence properties and computational efficiency. Problem parameters may also change during run time to improve speed and accuracy including sampling rates, interpolation strategies, and varying grid resolutions.

While image registration is a computationally intensive problem, it can be readily accelerated by exploiting parallelism. Enhancements that focus on acceleration through parallelism can be binned into levels based on the basic unit of computation considered. Each of these must use a parallel platform as the target to exploit the exposed parallelism. These platforms support a standard parallel processing approach, a few of which are covered in the next section.

MULTICORE PLATFORMS

Many multicore systems are viable acceleration platforms for medical image registration. They vary in a variety of dimensions including number of processing elements, size and hierarchy of memory, the bandwidth and topology of on-chip interconnect, single-chip versus multichip, and specialized instructions or coprocessors [3]. Perhaps most importantly for this work, these high-performance multicore platforms expose different programming models, which exhibit different threading models, memory models, and even different language constructs for utilizing platform intrinsics. In this section, we discuss a few commonly used options that are applicable to image registration.

Message passing interface (MPI) is a popular standard for explicitly parallelizing code on multiprocessor systems. Threads have local memory that can be readily implemented on distributed memory platforms such as clusters. Threads then exchange data across the cluster with explicitly defined communication links. Because communication may be over relatively long latency links, threads tend to be more loosely coupled, which lends MPI to being utilized at the highest levels of parallelism. As an example, to employ MPI for medical image registration, gradient computation of the similarity measure can be distributed equally across nodes in a cluster [4]. This distribution is possible by virtue of the fact that each finite difference calculation for each control point is independent and requires only the neighboring voxels and control points to calculate.

A GPU is an array of processing elements customized for pixel processing. The increasing programmability of GPUs have made them excellent candidates for many other applications including image registration [5], [6], [17]. High-level languages are emerging to aid the task of programming GPUs such as NVIDIA's Compute Unified Device Architecture (CUDA) [7]. The GPU programming models export the architecture as a large number of lightweight threads. With CUDA, threads are grouped into blocks that may coordinate on one tightly clustered set of processing elements that are arrayed on NVIDIA GPUs. Some memory is shared while others are distributed, but each programming approach has explicit constructs to ensure high-speed input/output (I/O). As an independent streaming operation, the task could be efficiently distributed across the processing elements of the GPU.

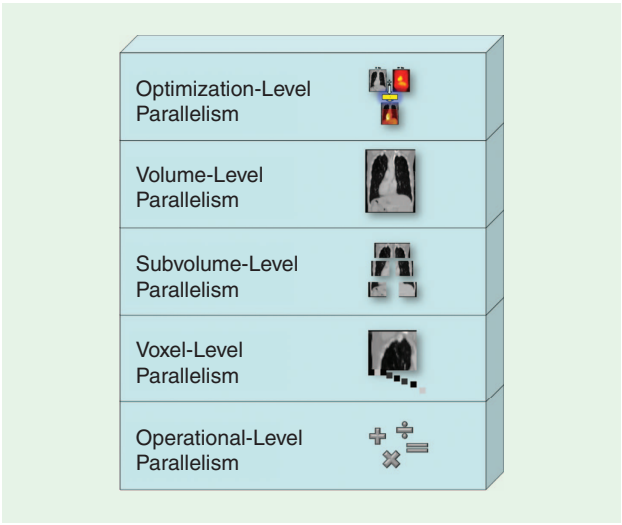
For the lowest level of parallelism, hardware description languages (HDLs) are often deployed. With HDLs, the final implementation is not destined for a processor, so designers lay out their application structurally, exposing interfaces and cycle-by-cycle control. A significant departure from traditional software programming languages, HDLs have no threading model and completely distributed memory structure. FPGAs can accelerate the voxel processing of transformation application and the similarity measure calculation in medical image registration [8]. The independence of tasks allows for many memory accesses, operations, and I/O to be performed in the same clock cycle.

Since many of these acceleration techniques are independent of each other and implemented on different types of platforms, a heterogeneous computational platform that supported all of these approaches would create a powerful new image registration engine. Orthogonal acceleration techniques such as CUDA and MPI techniques could provide multiplicative speedup effects. But to properly utilize such a heterogeneous multicore system, parallelism in the application domain must be identified and properly mapped to the target architecture.

IMAGE-REGISTRATION SPECIFIC TAXONOMY OF PARALLELISM

While acceleration techniques, in general, may modify functionality, we focus on the categorization of techniques that rely on parallelism for performance improvements. As with classical general-purpose categorizations, we classify acceleration techniques into "levels," which are depicted in Figure 2. Like the classical levels of parallelism (bit, data, instruction, task/thread, and process level), higher levels are specializations (or restricted forms) of lower levels. Our taxonomy can be mapped to classical levels in different ways (e.g., voxel-level parallelism could be implemented with data-level parallelism or task-level parallelism), but some mappings are impractical (e.g., optimization-level parallelism cannot be implemented with bit-level parallelism).

Conversely, it tends to be easier to reap the rewards of higher-level parallelism than lower. Many architectural and application factors affect this tendency both positively and negatively



[FIG2] Our domain-specific organization of parallelism.

(e.g., communication patterns, memory sizes, topology, and compiler performance), but for typical applications, the higher the level of parallelism expressed, the more readily applications can be accelerated.

OPTIMIZATION-LEVEL PARALLELISM

Optimization-level parallelism represents those parts of an algorithm that can run in parallel given the basic unit is an iteration of the image registration routine. Ino et al. [9] use this idea (which they call "speculative parallelism") to promote faster convergence in their time-critical registration application. Since the best optimization parameters are difficult to identify a priori, multiple instances of the same algorithm are launched with different parameters. Ultimately, after a specified time period, the best solution from these instances is selected. The multiple optimization instances require minimal communication and coordination and therefore are easy to execute in parallel.

Butz and Thiran [10] perform registration by utilizing a genetic algorithm in which fitness (or the metric of survival) is determined by how well the transformed image matches the fixed image. They implement this approach with an existing genetic solver parallelized using the MPI on a ten-node cluster. This optimization-level parallelism is naturally utilized because evaluating the population of solutions is an inherently independent act. With this class of parallelism, utilizing it is straightforward and can be efficiently implemented, but an individual optimization instance is not accelerated. For this, application designers must tap into opportunities at lower levels of parallelism.

VOLUME-LEVEL PARALLELISM

Volume-level parallelism is a generalization of optimization-level parallelism where the computational units operate on entire volumes. For example, an optimization iteration could be pipelined (applying one trial transform to the moving image while generating another candidate transform). Ino et al. [9] discuss the potential of "task parallelism" in accelerating the gradient computation of a rigid registration algorithm. This is possible, since independent finite difference calculations are done using the entire volume. While volume-level parallelism is simple to capture, its use is limited. For many algorithms, the number of independent, entire-volume calculations is small. Furthermore, distributing volumes to processing elements can suffer from high communication overhead. Lower levels of parallelism have tended to offer more opportunities for acceleration.

SUBVOLUME-LEVEL PARALLELISM

In medical image registration, subvolume-level parallelism is perhaps the most popular. In this approach, the computation is performed on subvolumes of image. Often designers can divide volume-level work into smaller subvolumes that are later recombined to produce the final solution. While this creates many opportunities for parallelism, it comes at the price of additional overhead such as coordinating how volumes will be split, managing overlap regions, and consolidating results.

Rohlfing and Maurer [11] employ subvolume-level parallelism for accelerating the similarity calculation. The volume is broken into equally sized sections such that a thread computes its local mutual histogram for mutual information (MI) and then merges its result into the global one. Ourselin et al. [12] use a block matching approach to find the deformation field. Inspired by video compression, the block matching technique compares "blocks" of one image against blocks of the other. These calculations are distributed across processors using MPI. In the same implementation, the authors accelerate image resampling with OpenMP. By distributing computation on individual multiprocessor machines, processes can share image memory and reduce the communication overhead incurred by transmitting images. They simultaneously utilize two programming paradigms to improve performance results.

Ino et al. [9] use "data parallelism" by distributing "small parts" of the image to subtasks that are assigned to different processors. Ino, et al. leverage this same level of parallelism in [4] by distributing the gradient computation of the similarity measure for control points across a distributed memory system. Such a distribution not only load balances the computation, but also reduces the memory requirements on an individual node.

Stefanescu et al. [13] parallelize the demons algorithm [14], which is based on optical flow, onto a 15-node cluster. The authors split the image into subvolumes to perform matching and filtering. Stenfanescu et al. [15] perform similar parallelization using a different registration technique. Subvolumes are assigned to different processors and communication is regularized over them. Hardware-based approaches can also utilize subvolume-level parallelism. Dandekar et al. [16] create an architecture in an FPGA that solves the registration problem recursively on subvolumes. Since each subvolume is an independent local registration problem, datapaths can be replicated for additional performance.

Greater effort has been applied to this level of parallelism to achieve speedups in medical image registration. This form is the most general flavor of parallelism and can be readily exploited by the most commonly used parallel platform clusters. While clusters

are not optimally suited to lower levels of parallelism, researchers have been finding opportunities for parallelism at lower levels using different platforms.

VOXEL-LEVEL PARALLELISM

Voxel-level parallelism describes parallelism in terms of single voxels. In this case, the regional benefit of using subvolumes is not present, so application designers find parallelism in independent voxel computations. Strzodka et al. [17] implement a gradient flow algorithm optimized for a GPU. This algorithm maps well to a GPU as images are stored into texture memory and the operations used are supported by the GPU hardware. Warfield et al. [18] utilized a "workpile" of threads to process a voxel independent classification method. They used threads on a shared memory platform to accelerate the task. Voxel-level parallelism that cannot be modeled as subvolume parallelism turns out to be rare. But with the rise in popularity of GPUs, efforts that utilize this parallelism are likely to increase. The last level explored by designers is the parallelism present in processing an individual voxel.

OPERATION-LEVEL PARALLELISM

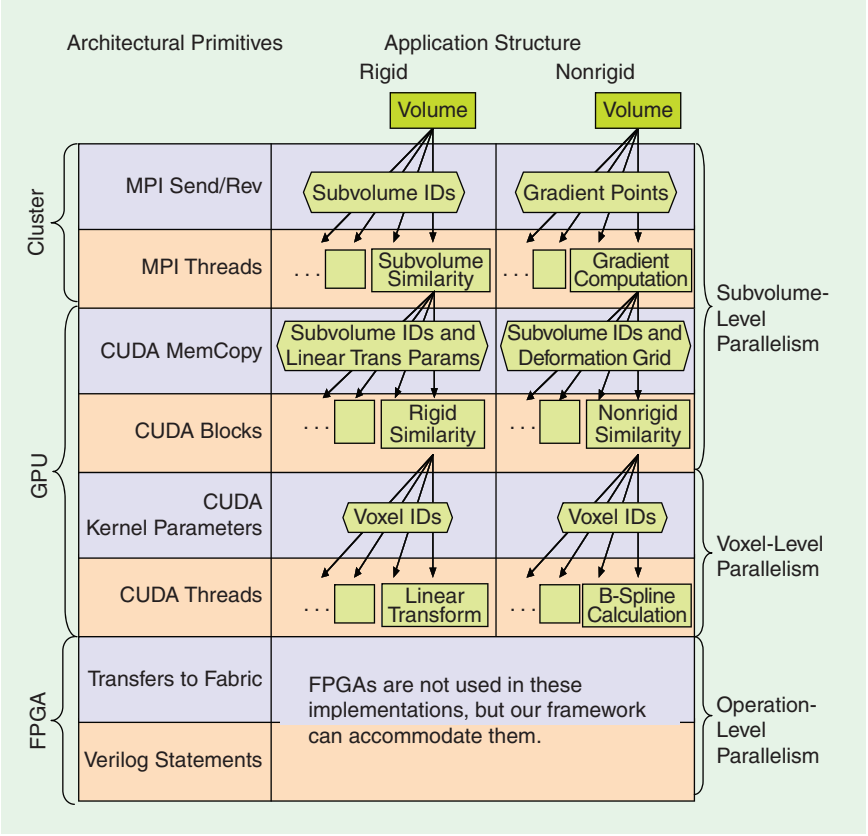
Operational-level parallelism is the lowest, most general form of parallelism. At this level, parallelism can be explored in many ways as the basic computational unit is no longer defined. Image-registration application designers have found parallel activities to accelerate when processing a single voxel. Castro-Pareja and Shekhar [8] construct an architecture that parallelizes the computation of transforming, interpolating, and computing the MI of voxels in an image in milliseconds. Designed in a HDL, it can perform MI-based rigid registration in about one minute. Beyond taking advantage of the instruction level parallelism transparently on a modern processor, operation level parallelism is the most difficult to utilize. Only custom hardware platforms are suitable to effectively exploit this level of parallelism.

STRUCTURED PARALLELISM IMPLEMENTATION

To evaluate the potential performance benefits of utilizing different levels of parallelism, we construct a design framework based on the image registration specific design taxonomy. If we were just dealing with individual platforms, we could simply implement each acceleration technique into the code base as necessary. But since we want to experiment with combined approaches, we start by expressing different types of parallelism in a structured fashion. After exposing and categorizing application parallelism, we map these to architectural primitives as presented by the programming models of our respective targets.

Mapping of parallelism to architectural resources requires insight about the application and architecture. Future work would assist in automating this procedure, but for now we rely on designer guidance. Once mapped, we employ the tools and design principles specific to the target platform component. For instance, a GPU's array of pixel-processing elements is often abstracted by the programming model as a set of threads with a language like CUDA. By using the target-specific programming environment, we exploit an efficient compilation path to the target with direct access to platform intrinsic crucial for performance. The development experience is also enriched through debuggers, visualization engines, and simulation environments.

For example, Figure 3 depicts both rigid and nonrigid applications each represented by a tree. Each registration algorithm is mapped to a hierarchical multiprocessing platform: a set of hosts networked together with MPI where each host has a GPU acceleration based on the CUDA programming model. The location of each application module indicates which computational resource it is mapped to. For instance, 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 computational mapping, we represent the system mapping of application communication to an architectural primitive. For a nonrigid registration algorithm, the gradient



[FIG3] A rigid and a nonrigid registration algorithm described in our framework targeting a hierarchical multiprocessing platform with graphics processors in a cluster.

computation for a free-form deformation (FFD) grid can be readily accelerated using a general-purpose cluster as well as a GPU.

To adhere to our hierarchical multiprocessing approach, the code was written to maintain the interfaces described by the structured mapping of parallelism. For example in Figure 3, subvolumes are sized to ensure that they are properly divisible inside an MPI thread. These interfaces allow for methodical changes of both the platform and the user interface.

EVALUATION

To evaluate our implementation framework, we choose a representative algorithm and high-performance multicore implementation vehicles.

ALGORITHM

Based on the structure of the framework and insights of the previous section, we implement the same image-registration algorithm on a single GPU and a GPU cluster. For rigid registration, the optimization method is based on downhill simplex [19] and the similarity measure is MSD with nearest-neighbor interpolation. The nonrigid algorithm is based on Rueckert et al.'s method [20] with an FFD grid utilizing B-spline interpolation between control points and trilinear interpolation between voxels.

For our rigid algorithm, parallelism comes from the independence of the MSD 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 the GPU pixel multiprocessors), so both the GPU and the cluster can exploit the similarity measure calculation parallelism. Each could be used for a single acceleration platform, but we combine them by constructing the MPI suvolumes to be large enough to be divided into smaller subvolumes used by the GPU.

For our nonrigid implementation, we utilize the subvolume-level parallelism of the gradient calculation for each control point in MPI. The gradient calculation using finite difference requires multiple similarity measure calculations with the addition of B-spline interpolation of control points to determine local deformation. A GPU can effectively accelerate this calculation by utilizing cooperative multithreading: mapping plane interpolation to a set of threads, row interpolation to a subset of the same threads, and finally the point interpolation to a single thread. As with rigid registration, the separation of these two parallelism constructs inside a structured framework allows us to utilize them on individual platforms as well as on a combined hierarchical multiprocessing platform.

EXPERIMENTAL SETUP

We based our experimental implementations on a single-threaded code base utilizing double precision floating point computations. Using this single-code base, we incorporated acceleration techniques wrapped by preprocessing directives. At compile time the software could be targeted for a specific parallel platform. The considered parallel platforms are as follows:

- a single GPU-NVIDIA GeForceGTX 285 with 1 GB of RAM targeted with CUDA SDK 2.2
- a GPU cluster: Four GPUs in separate PCs connected via gigabit Ethernet with the structure described in Figure 3.

The GPU implementations utilized single precision floating point calculations to optimize performance on the platform. 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 $256 \times 256 \times 256$ with 8 b representing voxel intensity. The deformation parameters were determined at random for each case and the rigid and nonrigid registration cases were separate so that we could study both scenarios individually. There was no rigid misalignment in the nonrigid registration cases and no nonrigid misalignment in the rigid registration cases. The rotation ranged between $-25°$ and $25°$ on each axis and between -25 mm to 25 mm in each dimension. With nonrigid registration, five new pairs were created by deforming a torso with a grid of size $5 \times 5 \times 5$ 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 $1.38 \times 1.38 \times 1.5$ mm. The algorithm stopped when a minimum step size was reached at which there was no improvement. The downhill simplex and gradient descent parameters (such as starting position, initial step size, and stopping criterion) were held constant across all cases and implementations.

We constructed MPI subvolumes equal to the number of nodes in the cluster that made them large enough to be divided into GPU subvolumes to match the GPU blocks. The GPU block dimensions were $8 \times 8 \times 4$ for rigid registration and $4 \times 4 \times 4$ for nonrigid registration. In general, larger blocks are more beneficial to performance since more threads are available to keep GPU utilization high, but in our case, nonrigid blocks were smaller because more resources are used for the nonrigid registration similarity measure calculation. This limits the number of threads that can be assigned to one processing element of the GPU.

RESULTS

Rigid registration results were of high quality for the GPU accelerated implementation, while nonrigid registration results vary as shown visually by one case in Figure 4, in which the moving and fixed images are tiled together in a checkerboard fashion. A perfectly registered result should show no misalignment at the tile boundaries. By observing the alignment of the checkerboard at the spine, one can see the improvement of the GPU accelerated result and the original result. Both greatly improve on the initially nonrigidly unaligned image, but the original result is superior to the GPU accelerated result. This is due to the fact that our GPU implementation currently uses single precision floating point arithmetic optimized for graphics, while the original implementation utilizes full double precision math. As a result, the GPU accelerated implementation is unable to arrive at the same quality of solutions in nonrigid registration

A summary of the test results is shown in Table 1. Since the images were artificially deformed with a known deformation

field, we calculated the average distance between the known and recovered deformation over all voxels. As a point of comparison, the original unaccelerated implementation achieved an average accuracy of 0.1 mm and 0.6 mm for the rigid and nonrigid cases, respectively. Note that 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 perform on the GPU, the timing results reported in Table 1 do not include the time for initialization, file I/O, or the one time image loads into GPU memory. The total time for this overhead takes under two seconds, most of which could be amortized by new images streaming into the GPU during the registration of prior images. Considering acquisition and reconstruction time of intraprocedural medical images, both rigid registration GPU implementations are feasible in a clinical setting by producing a new aligned image in just a few seconds. Clinicians using this platform during a procedure would see aligned preprocudural images refresh every few seconds based on the changes captured by the intraprocedural images, which would provide meaningful, timely guidance for a variety of procedures.

[FIG4] Example registration (Case 1) of an image and its nonrigidly deformed version fused with a checkerboard pattern: (a) uncorrected, corrected with the original, (b) unaccelerated CPU implementation, and (c) corrected with the implementation with GPU acceleration.

DISCUSSION

The single GPU significantly improve the performance of the original code base, in both the rigid and nonrigid registration cases. In each of these implementations, the performance improvement is derived from structuring the application so that the code can be methodically targeted to hierarchical multiprocessing platforms. We observed performance derived from this GPU implementation comes at the expense of inaccuracy over the original implementation comes from the limited floating point precision present in the GPU. When a control point is varied for its finite difference calculation, it makes only a minor change in the similarity measure. Even though the GPU approximates well most of double precision finite difference calculations, some subset of them is poorly estimated during each step. Since all points advance simultaneously after the gradient is calculated, even a few wayward control points can significantly skew the similarity measure between the fixed and moving image, inhibiting the overall convergence.

SUMMARY

Hierarchical multiprocessing offers the potential of significant performance improvement to some compute intensive applications, but it is accompanied by new design challenges including finding and exploiting parallelism. In this work, we discussed our approach to utilizing hierarchical multiprocessing in the context of medical image registration. By first organizing application parallelism into a domain-specific taxonomy, we structured an algorithm to target a set of multicore platforms. We demonstrated the approach on a cluster of GPUs requiring the use of two parallel programming environments to achieve fast execution times. There is negligible loss in accuracy for rigid registration when employing GPU acceleration, but it does adversely effect our nonrigid registration implementation due to our usage of a gradient descent approach.

Towards our goal of robust real-time registration, we believe that the advantages of GPU and multi-GPU acceleration could be reaped by running different phases of image registration on different platforms (e.g., using GPU acceleration first for a fast, coarse solution, and then not using it for more accuracy towards the end, as the imaging scenario would permit). Alternatively, a different algorithm could be employed for the GPU that would be less sensitive to precision effects or utilizing double precision floating point units now on high-end GPUs. We believe the structured approach presented here will enable our continued exploration into these and other implementations.

As we consider more complex acceleration techniques to combine, a robust system of capturing the parallelism of the application will be needed. Programming with formal underpinnings would give programmers a more natural way of expressing each type of parallelism without having to dive into low-level, idiosyncratic GPU languages, for example.

[TABLE 1] SPEED AND ACCURACY RESULTS OF RIGID REGISTRATION AND NONRIGID REGISTRATION, AVERAGED OVER FIVE SEPARATE CASES EACH.

	PLATFORM	AVERAGE ACCURACY (MM)	AVERAGE NUMBER OF ITERATIONS	AVERAGE TIME PER REGISTRATION (S)
RIGID CASES	ONE GPU	0.10	313	7.9
	FOUR GPUs	0.10	313	2.5
NONRIGID CASES	ONE GPU	2.43	12.6	250
	FOUR GPUs	2.43	12.6	98

AUTHORS

William Plishker (plishker@umd.edu) graduated in 2006 from the University of California at Berkeley with a Ph.D. degree in electrical engineering. He is a post-doctoral researcher at the University of Maryland. His focus is on application acceleration using dataflow modeling and leveraging different forms of parallelism. He has published papers on new dataflow models and scheduling techniques as well as application acceleration on multiple platforms including clusters, GPUs, FPGAs, and network processors. His application areas of interest include medical imaging, software defined radio, networking, and high energy physics. His Ph.D. research centered around the acceleration of network applications on network processors.

Omkar Dandekar (dandekar@ieee.org) received the B.E. degree in biomedical engineering from the University of Mumbai, India, in 2000, the M.S. degree in electrical engineering from The Ohio State University, Columbus, in 2004, and the Ph.D. degree in electrical and computer engineering from the University of Maryland, College Park, in 2008. He is currently a senior engineer with Intel Corporation, Hillsboro, Oregon. He has previously worked as a graduate research assistant at the University of Maryland, School of Medicine and at the Cleveland Clinic Foundation. His primary research interests include medical imaging, digital VLSI design, and hardware acceleration of image processing algorithms, with special focus on real-time 3-D imaging and advanced image processing and analysis for image-guided interventions. He has published over 30 refereed technical articles in this field. Currently, his research work is focused on computational lithography and patterning techniques for advanced technology nodes.

Shuvra S. Bhattacharyya (ssb@umd.edu) received the B.S. degree from the University of Wisconsin at Madison and the Ph.D. degree from the University of California at Berkeley. He is a professor in the Department of Electrical and Computer Engineering University of Maryland, College Park. He holds a joint appointment in the University of Maryland Institute for Advanced Computer Studies and an affiliate appointment in the Department of Computer Science. He is the coauthor/coeditor of five books and the author/coauthor of more than 150 refereed technical articles. His research interests include signal processing systems, architectures, and software; biomedical circuits and systems; embedded software; and hardware/software codesign. He has held industrial positions as a researcher at the Hitachi America Semiconductor Research Laboratory (San Jose, California), and compiler developer at Kuck and Associates (Champaign, Illinois).

Raj Shekhar (rshekhar@umm.edu) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1989, the M.S. degree in bioengineering from Arizona State University, Tempe, in 1991, and the Ph.D. degree in biomedical engineering from The Ohio State University, Columbus, in 1997. He worked as a senior researchengineer for Picker International (now Philips Healthcare) for two years before joining the Department of Biomedical Engineering, Cleveland Clinic Foundation, Ohio, in 1998. He is an associate professor of diagnostic radiology at the University of Maryland School of Medicine. He is also an affiliate professor of bioengineering and electrical and computer engineering. He has been a researcher and innovator in the field of medical imaging for over ten years, during which time he has published over 50 refereed technical papers. His research interests include medical imaging, image processing, platform acceleration, and image-guided interventions.

REFERENCES

[1] W. Plishker, O. Dandekar, S. S. Bhattacharyya, and R. Shekhar, "Towards systematic exploration of tradeoffs for medical image registration on heterogeneous platforms," in *Proc. IEEE Biomedical Circuits and Systems Conf.*, Baltimore, MD, Nov. 2008, pp. 53–56.

[2] W. Plishker, O. Dandekar, S. S. Bhattacharyya, and R. Shekhar, "A taxonomy for medical image registration acceleration techniques," in *Proc. IEEE-NIH Life Science Systems and Applications Workshop*, Bethesda, MD, Nov. 2007, pp. 215–218.

[3] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *IEEE Signal Processing. Mag.,* vol. 26, no. 6, pp. 26–37, Nov. 2009.

[4] F. Ino, K. Ooyama, and K. Hagihara, "A data distributed parallel algorithm for nonrigid image registration," *Parallel Comput.*, vol. 31, no. 1, pp. 19–43, 2005.

[5] R. Shams and N. Barnes, "Speeding up mutual information computation using NVIDIA CUDA hardware," in *Proc. Digital Image Computing: Techniques and Applications (DICTA)*, Adelaide, Australia, Dec. 2007, pp. 555–560.

[6] R. Shams and R. A. Kennedy, "Efficient histogram algorithms for NVIDIA CUDA compatible devices," in *Proc. Int. Conf. Signal Processing and Communications Systems (ICSPCS)*, Gold Coast, Australia, Dec. 2007, pp. 418–422.

[7] NVIDIA, *NVIDIA CUDA, Compute Unified Device Architecture, Programming Guide*. 2007.

[8] C. R. Castro-Pareja and R. Shekhar, "Hardware acceleration of mutual information-based 3D image registration," *J. Imaging Sci. Technol.*, vol. 49, no. 2, pp. 105–113, 2005.

[9] F. Ino, Y. Kawasaki, T. Tashiro, Y. Nakajima, Y. Sato, S. Tamura, and K. Hagihara. "A parallel implementation of 2-D/3-D image registration for computer-assisted surgery," *Int. J. Bioinformatics Res. Appl.,* vol. 2, no. 4, pp. 341–358, 2006.

[10] T. Butz and J.-P. Thiran, "Affine registration with feature space mutual information," in *Medical Image Computing and Computer-Assisted Intervention*, vol. 2208 (Lecture Notes in Computer Science), W. J. Niessen and M. A. Viergever, Eds. Berlin, Germany: Springer-Verlag, 2001, pp. 549–556.

[11] T. Rohlfing and C. R. Maurer, "Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees," *IEEE Trans. Inform. Technol. Biomed.,* vol. 7, no. 1, pp. 16–25, 2003.

[12] S. Ourselin, R. Stefanescu, and X. Pennec, "Robust registration of multimodal images: Towards real-time clinical applications," in *Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI'02)* (Lecture Notes in Computer Science), 2002, pp. 140–147.

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

[14] J. P. Thirion, "Non-rigid matching using demons," in *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'96)*, San Francisco, CA, USA, 1996, pp. 245–251.

[15] R. Stefanescu, X. Pennec, and N. Ayache, "Grid powered nonlinear image registration with locally adaptive regularization," *Med. Image Anal.*, vol. 8, no. 3, pp. 325–342, 2004.

[16] O. Dandekar and R. Shekhar, "FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions," *IEEE Trans. Biomed. Circuits Syst.,* vol. 1, no. 2, pp. 116–127, 2007.

[17] R. Strzodka, M. Droske, and M. Rumpf, "Fast image registration in DX9 graphics hardware," *J. Med. Inform. Technol.*, vol. 6, pp. 43–49, 2003.

[18] K. Warfield Simon, A. J. Ferenc, and R. Kikinis, "A high performance computing approach to the registration of medical imaging data," *Parallel Comput.*, vol. 24, no. 9–10, pp. 1345–1368, 1998.

[19] A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.,* vol. 7, no. 4, pp. 308–313, 1964.

[20] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, "Nonrigid registration using free-form deformations: Application to breast MR images," *IEEE Trans. Med. Imag.,* vol. 18, no. 8, pp. 712–721, 1999. [SP]