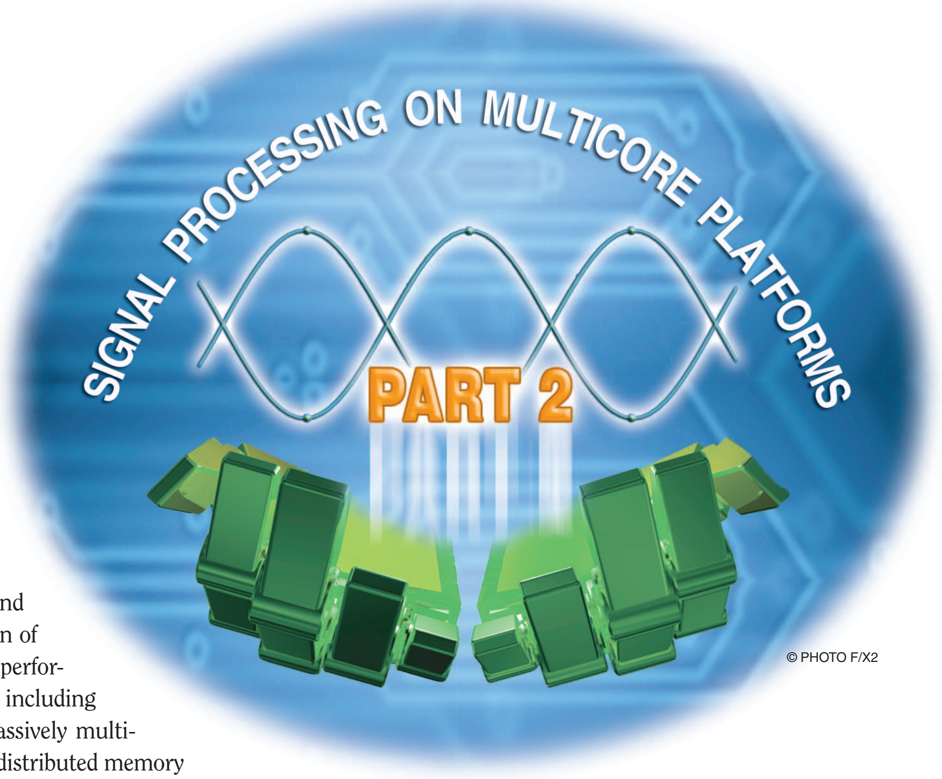
[Ramtin Shams, Parastoo Sadeghi, Rodney A. Kennedy, and Richard I. Hartley]

A Survey of Medical Image Registration on Multicore and the GPU

[A look at early, recent, and state-of-the-art methods using high-performance computing architectures]



© PHOTO F/X2

In this article, we look at early, recent, and state-of-the-art methods for registration of medical images using a range of high-performance computing (HPC) architectures including symmetric multiprocessing (SMP), massively multiprocessing (MMP), and architectures with distributed memory (DM), and nonuniform memory access (NUMA). The article is designed to be self-sufficient. We will take the time to define and describe concepts of interest, albeit briefly, in the context of image registration and HPC. We provide an overview of the registration problem and its main components in the section "Registration." Our main focus will be HPC-related aspects, and we will highlight relevant issues as we explore the problem domain. This approach presents a fresh angle on the subject than previously investigated by the more general and classic reviews in the literature [1]–[3]. The sections "Multi-CPU Implementations" and "Accelerator Implementations" are organized from the perspective of high-performance and parallel- computing with the registration problem embodied. This is meant to equip the reader with the knowledge to map a registration problem to a given computing architecture.

IN AN OPERATING ROOM NOT SO FAR INTO THE FUTURE

A surgeon is performing a potentially life-saving pancreatectomy on a patient in early stages of pancreatic cancer. Two small incisions of no more than half an inch allow laparoscopic tools including a video camera and an ultrasound probe to be guided inside the abdominal cavity. A third, larger incision, is occupied by a hand-access device that facilitates the operation. The surgeon is able to locate the tumor in the ultrasound view with ease. This is largely possible due to a newly installed

Digital Object Identifier 10.1109/MSP.2009.935387

1053-5888/10/$26.00©2010IEEE

three-dimensional (3-D) navigation and visualization system that virtually renders the patient transparent.

The visualization system combines data from preoperative magnetic resonance (MR) and computed tomography (CT) scans with intraoperative laparoscopic ultrasound data to produce real-time high quality and dynamic 3-D images of the patient, in a process better known as multimodal registration and fusion. The high quality 3-D images of the tumor and the surrounding tissue allow the surgeon to resect the malignant cells with little damage to healthy structures.

Such a minimally invasive approach avoids the trauma of open surgery, and a faster recovery time means that the patient will be released from the hospital in just two days.

MULTIPROCESSING IN AN OPERATING ROOM

Image-guided therapy (IGT) systems play an increasingly important role in clinical treatment and interventions. By providing more accurate information about a patient during a procedure, these systems improve the quality and accuracy of procedures and make less invasive options for treatment available. They contribute to reduced morbidity rate, intervention time, post-intervention care, and procedure costs. For practical reasons, however, imaging systems that can be deployed in an operating room produce images with lower resolutions and lower signal to noise ratios than can be achieved by the state-of-the-art imaging systems preoperatively. Therefore, it is desirable to be able to use preoperative images of a patient together with those acquired during a procedure for best results. In brain surgery, for example, the main challenge is to remove as much as the malignant tissue as possible without affecting critical structures and while minimizing damage to healthy tissue. The surgeon uses high quality CT and MR scans of the patient to carefully plan a procedure. During a procedure, however, the brain undergoes varying levels of deformations at different stages of the surgery known as the brain shift. This brain shift, a result of change in the intracranial pressure, leakage of cerebrospinal fluid and removal of tissue, affects the accuracy of earlier planning and needs to be compensated for. The surgeon may take a number of intraoperative scans to correct the plan based on patient's current state and also to detect complications such as bleeding. To support the surgeon, the IGT system needs to register intraoperative scans with the patient and with preoperative images.

Modern medical imaging technologies are capable of producing high resolution 3-D or four-dimensional (4-D) (3-D + time) images. This makes medical image processing tasks at least one dimension more compute-intensive than standard two-dimensional (2-D) image processing applications. The higher computational cost of medial image analysis together with the time constraints imposed by the medical procedure determine the range of tools that can be practically offered through an IGT platform. It also often means that an IGT platform has to rely on HPC hardware and highly parallelized software. There are other practical considerations. For example, equipment used in an operating room should be designed to minimize footprint, power consumption, operating noise, and cost.

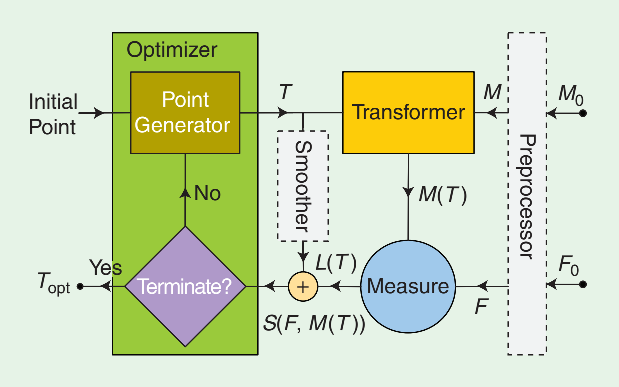
The continued development of multicore and massively multiprocessing architectures in recent years holds great promise for interventional setups. In particular, massively multiprocessing graphics units with general-purpose programming capabilities have emerged as front runners for low-cost high-performance processing. HPC, in the order of 1 TFLOPS, is available on commodity single-chip graphics processing units (GPUs) with power requirements not much greater than an office computer. Multi-GPU systems with up to eight GPUs can be built in a single host and can provide a nominal processing capacity of eight TFLOPS with less than 1,500 W power consumption under full load.

Hardware and architectural complexities in designing multicore systems aside, perhaps as big a challenge is an overhaul of existing application design methodologies to allow efficient implementation on a range of massively multicore architectures. As one quickly might find, direct adaptation of existing serial algorithms is more often than not neither possible due to hardware constraints nor computationally justified.

REGISTRATION

Registration is a fundamental task frequently encountered in image processing applications [1]. In medical applications, images of similar or differing modalities often need to be aligned as a preprocessing step for many planning, navigation, data-fusion and visualization tasks. Registration of images has been extensively researched in the medical imaging domain. Image based registration may use features that are derived from the subject's anatomy or those artificially introduced specifically for registration purposes. The former class of registration methods are known as intrinsic and the latter as extrinsic [2]. Extrinsic methods involve introduction of foreign objects such as stereotactic frames or fiducial markers and may be invasive. Once attached to the subject, markers remain fixed for multiple imaging sessions and can be used to align the images. Intrinsic methods, on the other hand, are noninvasive and can be used retrospectively. They may match a small set of corresponding anatomical and geometrical landmarks, use a set of structures obtained through segmentation, or be based on the entire content of images (e.g., voxel intensities). Content-based methods are particularly of interest since they can be fully automated but are typically compute-intensive. The focus of this survey is content-based registration methods.

Figure 1 shows various components of a general registration solver, with the main components being a transformer, a measure, and an optimizer. Registration as depicted here is an iterative process where one image is transformed within a predetermined parameter space and compared against the other. We call the former the moving and the latter the fixed image. A measure of similarity or distance is computed between the images at each step and used to determine if they are "sufficiently" aligned. This process is controlled by the optimizer that starts from an initial guess and determines subsequent



[FIG1] A general registration solver and its main components: *F, M,* and *M(T)* are fixed, moving, and transformed moving images, respectively.

steps to reach an optimal alignment. We will discuss each component in more detail in the following subsections.

TRANSFORMER

The transformer maps points in the moving image to new locations in the transformed image. Depending on the registration problem, a transformation can be collinear or deformable. Collinear transformations are line-preserving i.e., map straight lines onto straight lines. Collinear transformations can be described by a 4 × 4 matrix acting on homogeneous vectors representing 3-D points. Examples of collinear transformations include rigid, similarity, affine, and projective (projective transformations are rarely required in medical imaging applications). For this reason, these types of transformations have nearly identical complexity. Methods that implement rigid registration can be easily extended to affine, often without any change to the transformer.

Deformable transformation methods can be further categorized as parametric and nonparametric. Nonparametric methods are based on a variational formulation of the registration problem, where the transformation is described by an arbitrary displacement field regularized by some smoothing criteria [4]. Parametric methods are based on some piecewise polynomial interpolation of a displacement field using a set of control points placed in the image domain. B-splines are the favorites because they induce local deformations that limit the computational complexity of a large grid of control points [5]. Other functions such as thin-plate splines and Bezier functions have also been used. There are efficient methods for nonparametric registration including multigrid solvers. While parametric methods are more demanding, they yield themselves more easily to multimodal registration applications.

The transformer determines the intensity of the points in the transformed image by interpolating intensity values of corresponding points in the moving image. The simplest and fastest interpolation method is the nearest neighbor interpolation. Nearest neighbor should never be used in practice, as it results in poorly shaped cost functions, but may be useful to establish the baseline performance of the transformer. The most commonly used interpolation method is linear interpolation. Other methods include quadratic, cubic, cubic B-spline, and Gaussian interpolation [6].

A transformer spends the majority of its time performing interpolations. As noted by Castro-Pareja et al. [7] interpolation of the transformed moving image does not benefit from standard memory caching mechanisms due to nonsequential pattern of access to memory with low locality. As a result, transformer performance can well become memory bound.

MEASURE

A method of measuring the similarity or distance between images is required for automatic registration. Ideally a similarity measure attains its maximum, where the images are perfectly aligned and decreases as the images move farther away. A distance measure, on the other hand, attains its minimum where the images are aligned.

Commonly used similarity and distance measures are summarized in Table 1. Just as different classes of transformations are suitable for modeling different geometric distortions between the images, different similarity measures are used for different intensity distortions between the images. Measures are broadly categorized based on their suitability for single-modality and multimodality problems. All of the single-modality measures listed in Table 1 can be calculated by independent computations at each spatial location. From a parallelization point of view, this makes them readily adaptable to single instruction multiple data (SIMD) instruction sets and architectures such as GPUs. Multimodality measures determine statistical (mutual information) or functional (correlation ratio) dependance of images where each image is assumed to be a realization of an underlying discrete random variable. These methods require estimation of joint and marginal probability mass functions (pmfs) of the underlying discrete random variables from image data. Methods of pmf computation can be parallelized with varying degrees of difficulty and performance improvement. We will discuss this issue in more detail in the context of MI computation on the GPU in the section "GPUs."

OPTIMIZER

The optimizer is responsible for an efficient and often nonexhaustive strategy to search the transformation parameter space for the best match between the images. In image registration, optimizers can be broadly categorized as gradient-based or gradient-free, global or local, and serial or parallelizable.

Gradient-based methods require computation of partial derivatives of a cost function in addition to frequent computation of the cost function itself. Therefore, from an implementation perspective, they are more involved than gradient-free methods. The gradient computation can be based on the numerical estimation of the derivatives using finite differences. Alternatively, direct computation of the gradient can be performed when closed-form equations for the partial derivatives can be derived.

[TABLE 1] COMMONLY USED MEASURES.

MEASURE	ACRONYM	TYPE	USAGE	FORMULA[1]
SUM OF SQUARED DIFFERENCES	SSD	DIST.	SINGLE-MOD	$\mathcal{D}_{\text{SSD}}(\mathcal{I},\mathcal{J}) = \sum_{\mathbf{x}\in\Omega}(\mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathbf{x}))^2$
SUM OF ABSOLUTE DIFFERENCES	SAD	DIST.	SINGLE-MOD	$\mathcal{D}_{\text{SAD}}(\mathcal{I},\mathcal{J}) = \sum_{\mathbf{x}\in\Omega}\lvert\mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathbf{x})\rvert$
NORMALIZED CROSS CORRELATION [1]	NCC	SIM.	SINGLE-MOD	$\mathcal{S}_{\text{NCC}}(\mathcal{I},\mathcal{J}) = \sum_{\mathbf{x}\in\Omega}\frac{\mathcal{I}(\mathbf{x})\,\mathcal{J}(\mathbf{x})}{\sqrt{\mathbb{E}[\mathcal{I}(\mathbf{x})^2]\mathbb{E}[\mathcal{J}(\mathbf{x})^2]}}$
CORRELATION COEFFICIENT [1]	CC	SIM.	SINGLE-MOD	$\mathcal{S}_{\text{CC}}(\mathcal{I},\mathcal{J}) = \sum_{\mathbf{x}\in\Omega}\frac{(\mathcal{I}(\mathbf{x}) - \mathbb{E}[\mathcal{I}(\mathbf{x})])(\mathcal{J}(\mathbf{x}) - \mathbb{E}[\mathcal{J}(\mathbf{x})])}{\sigma(\mathcal{I})\sigma(\mathcal{J})}$
GRADIENT CORRELATION	GC	SIM.	SINGLE-MOD	$\mathcal{S}_{\text{GC}}(\mathcal{I},\mathcal{J}) = \frac{1}{d}\sum_{i=1}^{d}\mathcal{S}_{\text{CC}}\left(\frac{\partial\mathcal{I}}{\partial x_i}, \frac{\partial\mathcal{J}}{\partial x_i}\right)$
MUTUAL INFORMATION [8, 9]	MI	SIM.	MULTI-MOD	$\mathcal{S}_{\text{MI}}(\mathcal{I},\mathcal{J}) = \sum_{i}\sum_{j}p_{\mathcal{I}\mathcal{J}}(i,j)\log\frac{p_{\mathcal{I}\mathcal{J}}(i,j)}{p_{\mathcal{I}}(i)p_{\mathcal{J}}(j)}$
NORMALIZED MUTUAL INFO. [10]	NMI	SIM.	MULTI-MOD	$\mathcal{S}_{\text{NMI}}(\mathcal{I},\mathcal{J}) = \frac{2\mathcal{S}_{\text{MI}}(\mathcal{I},\mathcal{J})}{H(\mathcal{I}) + H(\mathcal{J})}$ (SEE NOTE 2)
CORRELATION RATIO [11]	CR	SIM.	MULTI-MOD	$\mathcal{S}_{\text{CR}}(\mathcal{I};\mathcal{J}) = \frac{\sigma^2(\mathbb{E}[\mathcal{J}\lvert\mathcal{I}]}{\sigma^2(\mathcal{I})}$

[1] $\Omega \subset \mathbb{R}^d$ represents a d-dimensional image domain.

[2] Entropy is defined as $H(\mathcal{I}) = \sum_i p_{\mathcal{I}}(i)\log 1/p_{\mathcal{I}}(i)$, where image $\mathcal{I}$ is assumed to be a discrete random variable with a probability mass function (pmf) given by $p_{\mathcal{I}}(\cdot)$.

Local methods find a local optimum in the vicinity of an initial point and within their capture range. They may converge to an incorrect alignment if not properly initialized. Global methods, however, find the global optimum within a given range of parameters. They are robust with respect to selection of the initial point but at the cost of slower convergence. Global and local methods may be combined to improve robustness while maintaining a reasonable convergence rate.

Some optimization algorithms are only suited for serial execution, where each optimization step is dependent on the outcome of previous step(s). Others may be amenable to parallelization. For example, each step of the gradient descent optimization in N-dimensional space requires computation of N partial derivatives of the cost function. Here, there is limited opportunity to run up to N tasks in parallel, and of course the additional line minimization step that may follow cannot be readily parallelized. We call such methods partially parallelizable. And finally, we refer to optimization methods that have been designed for parallel execution with minimal interstep dependency as fully parallelizable.

Table 2 lists some optimization algorithms used for image registration and their respective classification.

The overall performance of a registration algorithm is dependent on the effectiveness of the optimization strategy. This in turn depends on the iterations needed for the algorithm to converge. For gradient-free optimization, we define an iteration as a step that involves a single computation of the cost function. For gradient-based optimization, an iteration is defined as a step that involves a single computation of the gradient. Depending on the type of gradient-based method this may involve several evaluations of the cost function.

[TABLE 2] CLASSIFICATION OF SOME OPTIMIZATION METHODS.

METHOD	CLASSIFICATION		
POWELL [12]	GRADIENT FREE	LOCAL	SERIAL
SIMPLEX [13]	GRADIENT FREE	LOCAL	PARTIALLY PARALLELIZABLE
SOBLEX[1] [14]	GRADIENT FREE	COMBINED	PARTIALLY PARALLELIZABLE
MDS[1,2] [15]	GRADIENT FREE	LOCAL	PARTIALLY PARALLELIZABLE
GRADIENT DESCENT [12]	GRADIENT BASED	LOCAL	PARTIALLY PARALLELIZABLE
QUASI-NEWTON [12]	GRADIENT BASED	LOCAL	PARTIALLY PARALLELIZABLE
LEVENBERG-MARQUARDT [12]	GRADIENT BASED	LOCAL	PARTIALLY PARALLELIZABLE
SIMULATED ANNEALING [12]	GRADIENT FREE	COMBINED	PARTIALLY PARALLELIZABLE
DIRECT[3] [16]	GRADIENT FREE	GLOBAL	FULLY PARALLELIZABLE
GENETIC [17]	GRADIENT FREE	GLOBAL	FULLY PARALLELIZABLE

[1] A simplex variant, [2] multidirectional search, [3] dividing rectangles.

Gradient-based optimizers do more in a single iteration and they also converge with a significantly lower number of iterations compared to gradient-free methods. The convergence rate of an optimizer depends on many factors including the size of the parameter space, optimizer settings (e.g., convergence criteria), and the misalignment between the images. It is also often data dependent.

The computational bottleneck of registration is not the optimizer but the computation of the transformation and the measure. Most researchers have focused on fine-grained parallelization of these components. A few have considered coarse-grained parallelization, which involves parallelization of the optimizer itself [18], [19].

PREPROCESSOR

We have shown the preprocessor in dotted lines in Figure 1 to emphasize that it is an optional component. Preprocessing encapsulates a wide range of tasks that may be performed on images outside the optimization loop and at the beginning of the process. These may include filtering, rectification, gradient computation, pyramid construction, feature detection, etc. An example is given in one of the earlier efforts to parallelize image registration by Warfield et al. [20]. They extract features based on tissue labels given by prior segmentation and parallelize a feature-based interpatient registration method on a cluster of multiprocessor computers. They use the number of mismatching labels (NML) as a measure of distance in their registration algorithm.

Given that preprocessor is not in the critical pass, there is little incentive for parallelizing it. Unless of course the registration process itself is sped up to the point that preprocessing becomes a bottleneck. This is likely to become the case as registration algorithms enter the real-time domain.

COMPUTATIONAL EXPENSE OF IMAGE REGISTRATION

Image registration in general is computationally expensive and has been largely confined to preoperative applications. The main bottlenecks are typically the transformer and the computation of the measure. Single modality measures such as sum of squared differences (SSD) and correlation coefficient (CC) are less compute-intensive than multimodality measures such as mutual information (MI) and correlation ratio (CR). (Some authors use "normalized cross correlation" to refer to correlation coefficient. We prefer correlation coefficient, which is the accepted term in statistics.) Computation of MI requires an estimation of the joint probability density of image intensities. This typically entails, computing a joint histogram of image intensities. A seemingly simple task that is far from trivial on some massively parallel architectures such as GPUs [21].

A sample breakdown of computations in one iteration of a gradient-free optimization algorithm is given in Table 3 for affine registrations using a single modality and a multimodality measure. The measurements are based on a Quad core Intel Core i7 920 and an NVIDIA GTX 295. The time spent outside of the measure and transformation components is negligible compared to the measure and transformation. On the CPU, the execution time is dominated by the transformer whereas on the GPU, the time spent in computing the measure, particularly for the MI, exceeds the transformer time. This is expected as GPUs are designed to speed up geometric transformations. Obviously, for more complex transformation models such as the deformable B-splines, more time will be spent in the transformer for both platforms.

[TABLE 3] A SAMPLE BREAKDOWN OF COMPUTATIONS FOR AFFINE REGISTRATIONS ON A MULTICORE CPU AND A GPU.

	AFFINE (SSD)		AFFINE (MI)	
	MEASURE	TRANSFORM	MEASURE	TRANSFORM
CPU	4.3%	95.7%	13.5%	86.5%
GPU	50.4%	49.2%	86.9%	13.0%

We note that optimization algorithms make decisions based on the measure and do not directly use the intermediate results of the transformer. As such, transformation and similarity measure computations may be performed in one step and within the same module to remove the need for storage and subsequent retrieval of transformed image data. This obviously improves performance, especially where input/output traffic is an issue. However, it also makes it more difficult to modularize the implementation and cater for arbitrary combinations of transformations and measures.

MULTI-CPU IMPLEMENTATIONS

SYMMETRIC MULTIPROCESSING

In SMP architectures, multiple CPUs/cores have access to a single shared main memory. This makes parallelization of serial code relatively straightforward. The main methods for parallelization on SMP architectures are POSIX threads (pthreads) and OpenMP [22], [23]. The pthreads standard defines an application programming interface (API) for explicit instantiation, management and synchronization of multiple threads, whereas OpenMP mainly consists of a set of compiler directives (and a supporting API) that allows for implicit parallelization.

Most serial programs can be parallelized on SMP architectures with minimal modification. The ease with which parallelization can be achieved, especially with OpenMP, can be deceiving. There is an adage in HPC circles that says "OpenMP does not make parallel programming easy, it only makes bad parallel programming easy." We should emphasize that there is nothing inherently inhibiting about OpenMP or SMP platforms. It is only that optimal parallelization usually requires a change in the algorithm, programming model and memory access pattern in addition to the syntax. We encourage the reader to be prepared to reevaluate the approach to solving a problem on parallel systems and avoid the temptation of simply mapping a serial code to multiple threads. We also advise that use of synchronization primitives should be limited to a minimum and alternative methods to achieve an outcome without synchronization should be investigated. Synchronization refers to any mechanism for coordinating multiple threads or processes to complete a task. Examples of synchronization primitives include mutual exclusion (mutex), thread-join, and barrier. Atomic operations also involve implicit synchronization.

A good example of SMP parallelization of a registration algorithm is given by Rohlfing et al. [24]. They use pthreads to parallelize B-spline deformable registration on 64 CPUs. They exploit a combination of procedural (precomputation, multiresolution, and adaptive activation of control points) and architectural elements (e.g., data partitioning) to optimize their method. While the hardware has been long superseded, their approach is still relevant today. We would not change much about their method except that they use synchronized reduction of partial joint histograms into a global histogram in the MI computation phase by using the mutex lock. One can avoid the need for synchronization by dividing partial histograms and the resulting global histogram among the available threads. For N threads, this divides each partial histogram into N equally sized nonoverlapping regions. Each thread, then, computes part of the global histogram by summing values across corresponding regions of partial histograms. Since the regions are nonoverlapping, the computations are guaranteed to be free of write-conflicts and no synchronization is required.

MULTIPROCESSING WITH NUMA

Efficient memory access is an important design consideration in multiprocessor systems with many cores where maintaining an efficient cache coherency on a single-shared-bus becomes less practical as the number of processors increases. NUMA architecture divides memory into multiple banks; each assigned to one processor. Processors have faster access to their local bank than remote banks attached to other processors.

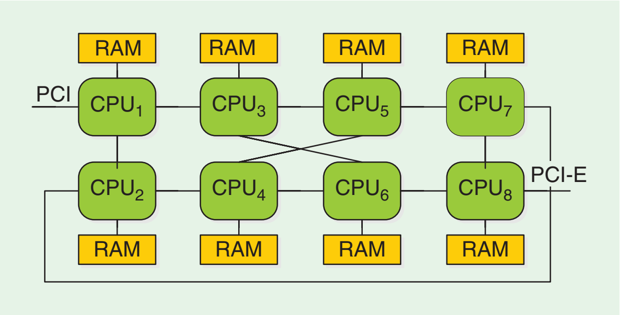
Access to memory on remote banks can be several times slower than access to local memory. This is due to data traveling through a longer path and also transient access requests by other processors that may require the memory bus to be shared. Figure 2 shows the schematic of a multiprocessor system with a NUMA architecture. An algorithm that is optimally designed for NUMA makes only infrequent attempts to access data on remote banks. A parallel application can theoretically achieve linear scalability with respect to memory throughput whenever proper distribution of memory to local banks is possible.

Image registration can be efficiently implemented on NUMA architectures as shown in Figure 3. Both the transform and measure computation can work on a spatial subset of the images. To achieve optimal performance, the fixed image F is divided among the memory banks and the corresponding portion of the transformed moving image $M(T)$ will also be stored on the same memory bank. However, the path taken by the optimization algorithm cannot be determined a priori and the transformer will use different areas of M to create the local portion of $M(T)$ at each iteration. As such, each memory bank will need to receive a local copy of the moving image M during the initialization step. Given that the optimization algorithm will take several iterations to converge, this initial overhead is justified.

The distribution of resources to specific memory banks requires setting an appropriate memory and processor affinity. Processor affinity refers to explicit binding of a thread to a specific processor. Memory affinity is explicit allocation of data on a specific memory bank. This is operating system dependent and will make the code less portable. The alternative is, of course, to be completely oblivious to the memory architecture and hope that the compiler and the operating system will make the right decisions. This may not be an entirely unreasonable strategy, depending on the number of processors and whether a program is memory bound or CPU bound. However, as the number of available CPUs increases or for programs that are memory intensive, it becomes more important to design an optimal memory access strategy.

MULTIPROCESSING WITH DISTRIBUTED MEMORY

DM architectures are characterized by lack of access to a global shared memory available to all processors. DM systems are typically built by clustering SMP or NUMA nodes. As such, in distributed architectures, subgroups of processors have access to shared memory.

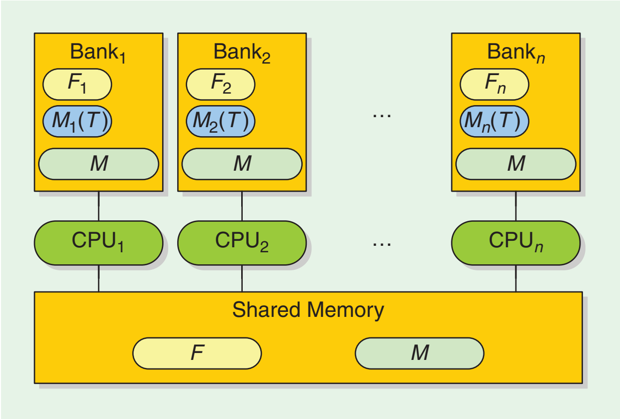


[FIG2] SunFire X4600 M2 schematic with eight NUMA nodes. A CPU can access remote memory through a maximum of three hops.

From a programming standpoint, these systems are characterized by the need for explicit data distribution and interprocess communication. The former has to be built into the application design and the latter is most commonly achieved through the message passing interface (MPI) [25].

The model given for data distribution in NUMA Figure 3 can be equally applied here. An early implementation is given by Butz and Thiran [18], where a Linux cluster was used to speed up MI-based registration for a global genetic optimizer. In [26], Ino et al. further partition the moving image to reduce the memory usage. This is motivated by the need to process large images in the order of 1,024 × 1,024 × 590 voxels. Partitioning both images also reduces traffic on the network during initialization. This can be an important consideration as the number of nodes increases and the overhead of the initialization phase compared to the optimization phase can no longer be ignored. Partitioning the moving image requires a prior estimate of the range of transformation parameters to ensure that a large enough portion of the image is loaded for the transformer.

A variation is given by distributed shared memory (DSM) architectures, where a large virtual address space is made available to all processes across all nodes. DSM can only hide the



[FIG3] Partitioning of the data set among multiple memory banks for improved access. The original data is loaded from a shared storage medium.

mechanism of communication between processes and not the associated latency. We argue that if the end goal is to achieve the highest performance, little benefit can be drawn from the convenience of a DSM architecture and the program should be designed to be aware of the locality of data.

Wachowiak and Peters [19] develop MI-based registration for a DSM architecture. Their implementation does not take memory locality into account, but they use DIRECT and MDS parallel optimization methods to their advantage. This coarse-grained parallelization results in lower communication-to-computation overhead.

As some authors have pointed out [27], a major benefit of DM clusters is their lower cost compared to many-core SMPs or DSM systems.

ACCELERATOR IMPLEMENTATIONS

GRAPHICS PROCESSING UNITS

The majority of recent research in multicore adaptation of registration algorithms has been focused on GPUs [28]–[34]. There are several reasons for the interest in GPUs. Thanks to fierce competition and driven by the gaming industry, GPUs today provide some of the highest performance per dollar and the lowest power consumption per FLOPS of any computing platform. While not every radiology department can afford the cost and space needed by a conventional HPC data center, they can certainly benefit from unlocking the computational power of the GPUs in their existing computers.

GPU implementations tend to be more challenging than multicore CPU implementations and are more rewarding in terms of achievable performance gains. Earlier work in this area (mainly prior to 2007) [35]–[42] involved devising methods to map the registration problem onto a graphics pipeline that was not specifically designed for general-purpose computing. The GPU landscape has since gone through a seismic change with the introduction of native general-purpose computing capabilities in late 2006. The GPU registration literature prior to 2007 has been superseded from both hardware and software perspectives. We will focus on the latest technology for general-purpose computing on GPUs in this section.

The modern software platforms for general-purpose programming on the GPU are currently NVIDIA's CUDA [43] and AMD/ATI's Brook+ [44]. These platforms are vendor-specific, however, OpenCL compliant implementations that provide hardware-independence are being gradually released by the vendors. This essentially invalidates the only remaining argument in favor of using the graphics pipeline for general-purpose programming, which has been better portability.

None of the papers we considered for this survey developed their methods for ATI Brook+. It appears that the research community has almost exclusively adopted CUDA as their preferred GPU platform. This is likely to change with wider support for OpenCL in non-GPU architectures such as IBM's Cell/BE and Intel's Larrabee.

Modern GPUs extend the single instruction multiple data (SIMD) paradigm to a single instruction multiple threads architecture (SIMT). SIMT provides more flexibility by parallelism for (almost) independent threads as well as data-parallel code. GPUs achieve their computational performance by dedicating more transistors to their arithmetic logic units (ALUs) for data processing, at the expense of reduced flow control and data caching. They extend the conventional thread-level parallelism by introducing two additional layers of parallelism in the form of closely knit groups of threads known as warps or wavefronts, and groups of warps/wavefronts known as thread blocks or simply blocks. Warps are significant since they define the unit of flow control in a GPU. Threads in a warp are bound to execute the same instruction (on different data). Diverging paths of execution for threads in a warp result in serial execution of all paths. Hence, an important consideration in adapting parallel code to GPU architecture is minimizing diversion in warps. This can be achieved by designing warp-aware algorithms and reorganizing data to optimize flow control. An example of such an approach is given in [33].

Another notable technical feature in the current generation of GPUs is the availability of an abundance of high bandwidth on-board RAM. The memory bandwidth of top-of-the-line GPUs exceeds 140 GB/s and cards with up to 4 GB of memory are available. This is particularly important for medical image analysis applications that have to deal with large 3-D data sets. Despite an extremely high bandwidth, the GPU's main memory is largely uncached and suffers from a rather large latency. Hence to fully utilize the bandwidth and achieve an optimal performance, one needs to understand the hardware architecture and its various memory and limited caching models. Optimum use of memory such as coalesced transfers may speed up an application by an order of magnitude. This level of flexibility is typically available with lower-level APIs and runtime SDKs such as CUDA (NVIDIA) [43] and CAL (ATI/AMD) [44]. Programs developed with a lower-level API lack portability and need to be maintained as the hardware evolves. Abstraction layers such as OpenCL and Brook+ avoid these issues by hiding memory management details from the developer. However, better portability may come at the cost of suboptimal performance.

GPUs are well equipped for speeding up geometric transformations. Geometric transformations (regardless of their type) require some sort of interpolation that involves reading the content of adjacent voxels in a cubic region of memory. Standard computer architectures are designed to optimize sequential memory access through their caching mechanism. This does not fully benefit 3-D interpolations over a cubic mesh. Modern GPUs, on the other hand, support a 3-D texture addressing mode that takes the geometric locality into account for caching purposes. This greatly improves the efficiently of transformations on the GPU.

Different MI computation methods on the GPU have been reported in the literature. Shams et al. compute MI by computing joint histograms on the GPU in [21], [29], and [33]. A main finding is that for different sized histograms (number of bins used for MI computation), the optimal algorithm differs. For bin ranges typical in MI computation (100 × 100 and above) an efficient histogram computation algorithm specifically designed for massively multiprocessing architectures is presented in [33]. The paper describes a new method for histogram computation (sort and count) that removes the need for synchronization or atomic operations, based on sorting chunks of data with a parallel sort algorithm such as bitonic sort. Lin and Medioni [30] report an adaptation of Viola's MI computation approach [8]. Their method approximates the joint pmf by stochastic sampling of the image intensities and using Parzen windowing. This method lends itself well to parallelization on the GPU, reduces the computational burden of transformations by only using a subset of image data, and provides analytic equations for computation of MI derivatives. However, sparse sampling of the data set may compromise accuracy of the registration [37]. A sampling method specifically designed for the GPU is given by Shams and Barnes [29]. This method samples the bin space for computing histograms rather than the intensity space. The method improves performance of computations and is subject to the same trade off between performance and accuracy. We note that a majority of researchers use direct computation of the histogram [3].

A natural extension to parallelization of registration algorithms on the GPU is horizontal parallelization across multiple GPUs. Multi-GPU systems belong to DM class of parallel architectures. An implementation on such systems involves data partitioning and the use of MPI as discussed in the section "Multiprocessing with Distributed Memory." We recommend the reader to refer to a more detailed discussion of the subject by Plishker et al. [45].

CELL BROADBAND ENGINE

Cell broadband engine (Cell/BE) is an asymmetric heterogeneous multicore processor with a DM architecture. It comprises a general-purpose PowerPC core known as a PPE and eight specialized vector processing cores known as SPEs. Each SPE is equipped with a four-way SIMD engine and has its own small (uncached) memory known as the local storage. Local storage is only 256 KB in the current generation of hardware, and it is shared between data and kernel instructions.

Optimal implementation of registration algorithms on Cell/BE architectures involves task-level parallelization, data partitioning, and vectorization of the code for the SPEs' SIMD engine. It also involves handling the transfer of data between the system memory and SPEs' local storage. The results by Ohara et al. [46], [47] and Rohrer and Gong [48] provide good insight into challenges involved in designing registration on this architecture for collinear and deformable registration, respectively.

FIELD PROGRAMMABLE GATE ARRAYS

A custom field programmable gate array (FPGA) accelerator prototype for MI-based rigid registration is given by Castro-Pareja et al. in [7]. They argue that a major bottleneck in MI computation using Collignon's method [9] is partial volume (PV) interpolation and that it is memory bound. They improve performance by parallelizing access to memory and assigning independent memory controllers and types of memory for storage and access to the fixed image, the moving image, and the joint histogram. A cubic addressing scheme is used for the moving image to speed up the interpolation. This is similar to caching available in GPUs for access to texture memory. An enhanced version of [7] is presented in [49] and a multirigid version with volume subdivisions is given by Dandekar [50].

FPGAs allow one to design customized hardware for specific registration tasks. However, they provide less flexibility compared to software-based implementations. With flexible general-purpose programming capabilities of modern GPUs, it is doubtful if FPGA-based implementations present any real benefit in this area.

SUMMARY OF THE LITERATURE

We have summarized existing contributions in HPC of registration methods in Table 4. The table serves as a quick reference to an array of methods on various platforms and by different groups.

Researchers have used various methods to present their performance results. All groups report at least the speedup results compared to a single-core CPU implementation. When interarchitecture comparisons are drawn, it is not always clear how well the CPU implementation has been optimized, if the streaming SIMD extensions (SSE) instruction set has been used, whether the code has been compiled as 64- or 32-b, or if 64- or 32-b floating point operations have been used. For these reasons, speedup results should be interpreted with caution, more so when the reported speedups are in the order of a hundred times or more.

Most groups report their speedups for the entire registration algorithm and for specific data sets. Comparison of different results is further complicated as authors may have implemented a multiresolution scheme to further speed up the process and used different convergence criteria. We have reported/estimated the results for the finest resolution in Table 4, whenever possible. As discussed earlier, the execution time is an almost linear function of the number of iterations of the optimization algorithm. Convergence criteria are most commonly based on the value of the measure and its relative improvement in a given step of the optimization. A less common approach is to set a fixed number of iterations as the convergence criterion. We call the former strategy dynamic convergence and the latter static convergence. Lack of associativity for floating point operations have the inevitable consequence that the same optimization algorithm operating on the same data set converges with different number of iterations on different architectures when dynamic convergence is employed. Even on the same architecture, compiler optimization of floating point operations results in variations. Unless experiments are performed on a large set of images, this skews the performance results one way or the other.

[TABLE 4] SUMMARY OF HIGH-PERFORMANCE IMAGE REGISTRATION METHODS IN THE LITERATURE.

	TRANSFORM		MEAS.	OPTIMIZER	HARDWARE	YEAR	PERF.[1]	COMMENTS	GROUP
CPU	COLLINEAR	SIMIL.	NML	POWELL	2 × SUN ENT. 5000 (2 × 8 ULTRASPARC I 167 MHZ)	1998	–	SMP CLUSTER, FEATURE BASED	WARFIELD [20]
		AFFINE	MI	GENETIC	PC CLUSTER (10 × 2 PENTIUM III 550 MHZ)	2001	–	DM, MI IS GRADIENT BASED	BUTZ [18]
		RIGID	LLC	?	PC CLUSTER (10 × 2 PENTIUM III 933 MHZ)	2002	–	BLOCK MATCHING WITH LOCAL LINEAR CORRELATION MEASURE (LLC)	OURSELIN [27]
		RIGID	MI, NMI	DIRECT, MDS	SGI ALTIX 3000 (20 ITANIUM II 1.3 GHZ)	2006	–	DMS (NUMAFLEX)	WACHOWIAK [19]
		RIGID	MI	POWELL	SUN SPARC T5120 (8 × ULTRASPARC T2 1.2 GHZ)	2009	47.7	SMP, SOLARIS	SHAMS[2]
		RIGID	MI	POWELL	INTEL Q6600 (PENTUIM CORE 2 QUAD 2.4 GHZ, FOUR CORES)	2009	15.8	SMP, 64-B LINUX	SHAMS[2]
		RIGID	MI	POWELL	INTEL CORE i7 920 (QUAD 2.66 GHZ, EIGHT THREADS)	2009	13.2	SMP, 64-B WINDOWS VISTA	SHAMS[2]
		RIGID	MI	POWELL	SUNFIRE X4600 M2 (8 × 2 OPTERON 2.6 GHZ)	2009	10.5	NUMA, 64-B LINUX	SHAMS[2]
	DEF.	B-SPLINE	NMI	GRAD. DESC. (D)	SGI ORIGIN 3800 (128 MPIS 12K)	2003	–	SMP, MAX. CPUS USED: 64	ROHLFING [24]
		B-SPLINE	NMI	GRAD. DESC. (D)	PC CLUSTER (64 × 2 PENTIUM III 1GHZ)	2005	–	DM (MYRINET)	INO [26]
GPU	COLLINEAR	RIGID	SSD	SIMPLEX	GEFORCE 6800	2006	98.0	CODED IN OPENGL AND GLSL	KÖHN [51]
		RIGID	SSD	GRAD. DESC. (B)	GEFORCE 6800	2006	858	CODED IN OPENGL AND GLSL	KÖHN[3] [51]
		RIGID	GC	?	QUADRO FX 1400, FX 3400, GTX 7800	2006	–	2-D/3-D REGISTRATION	INO [38]
		RIGID	VARIOUS	CUSTOM	GEFORCE 6800 GT	2006	–	VARIOUS MEASURES (E.G., SSD, CC, GC)	KHAMENE[3] [37]
		RIGID	VARIOUS	ARS + BN	GEFORCE 7800 GS	2008	–	2-D/3-D, VARIOUS MEASURES (E.G., SSD, CC, GC), ADAPTIVE RANDOM SEARCH + BEST NEIGHBOR	KUBIAS [42]
		RIGID	MI	SIMPLEX	GTX 8800 (16 MP/ 128 CORES)	2007	6.17	CUDA 1.0 (NO SUPPORT FOR 3-D TEXTURES), MI ESTIMATED BY BIN SAMPLING	SHAMS [29]
		RIGID	SSD	SIMPLEX	GTX 8800 (16 MP/ 128 CORES)	2008	6.05	CUDA 2.0	PLISHKER[3] [31]
		AFFINE	MI	GRAD. DESC. (A)	GTX 8800 (16 MP/ 128 CORES)	2008	–	MI ESTIMATED BY SAMPLING	LIN [30]
		RIGID	MI	POWELL	GTX 280 (30 MP/ 240 CORES)	2009	4.06	CUDA 2.0, USING 3-D TEXTURES, MI COMPUTED USING BITONIC SORT AND COUNT	SHAMS [33]
	DEFORMABLE	BEZIER	MI	POWELL	GEFORCE3 64 MB	2002	–		SOZA [35]
		NON-PAR.	SSD	GRAD. DESC.	GEFORCE FX 5800 ULTRA	2004	–	2-D/2-D, MULTIGRID SOLVER USED	STRZODKA [36]
		NON-PAR.	SSD	GRAD. DESC. (B)	GEFORCE 6800	2006	465	CODED IN OPENGL AND GLSL	KÖHN[3] [51]
		NON-PAR.	MI + KL	GRAD. DESC. (C)	GTX 7800	2007	2860	COMBINED MI AND KULLBACK-LEIBLER MEASURE, CODED IN OPENGL AND GLSL	VETTER[3] [39]
		NON-PAR.	MI + KL	GRAD. DESC. (C)	GTX 8800 ULTRA (16 MP/128 CORES)	2008	324	COMBINED MI AND KULLBACK-LEIBLER MEASURE, CODED IN OPENGL AND GLSL	FAN[3] [40]
		DEMONS	SSD	ITERATIVE	QUADRO FX 1400	2007	1050	CODED IN CG, PUBLISHED IN 2008	COURTY [41]
		DEMONS	SSD	ITERATIVE	GTS 8800 (12 MP/96 CORES)	2007	11.7	CODED IN BROOK, SSD EXCLUDED IN PERFORMANCE RESULTS	SHARP[3] [28]
		DEMONS	CC	ITERATIVE	QUADRO FX 5600 (16 MP/128 CORES)	2008	9.25	CUDA 0.9	ÖZÇELIK [32]
		B-SPLINE	SSD	GRAD. DESC. (C)	GTX 8800 (16 MP/ 128 CORES)	2008	3710	CUDA 2.0	PLISHKER[3] [31]
		POLYNOM.	MI	EXHAUSTIVE	QUADRO FX 5600 (16 MP/128 CORES)	2009	–	2-D/2-D, ULTRA LARGE 2-D IMAGES	RUIZ [34]
OTHER ACCELERATORS	COLLINEAR	RIGID	MI	N/A	FPGA (2 × ALTERA 1K100 80 MHZ)	2003	101	MI PARTIALLY COMPUTED IN H/W	PAREJA [7]
		RIGID	MI	N/A	FPGA (1 × ALTERA EP1S40 200 MHZ)	2004	20.0	MI FULLY COMPUTED IN H/W	PAREJA [49]
		AFFINE	MI	GRAD. DESC.	QS20 (2 × CELL/BE.: 2 × 1 PPE AND EIGHT SPEs)	2007	98.8	MI ESTIMATED BY SAMPLING	OHARA[3] [47]
	DEF.	MULTIRIGID	MI	SIMPLEX	FPGA (1 X ALTERA EP2S180 200 MHZ)	2007	13.4	MI FULLY COMPUTED IN H/W	DANDEKAR[3] [50]
		B-SPLINE	MI	GRAD. DESC.	QS20 (2 × CELL/BE.: 2 × 1 PPE AND EIGHT SPE)	2008	66.9	MI ESTIMATED BY SAMPLING	ROHRER [48]

[1] Normalized performance in milliseconds per mega voxel per iteration (ms/MVoxel/itr). [2] Previously unpublished result. [3] Additional information provided by the authors used to complete the table or to compute normalized performance results.

We have given normalized performance results in Table 4 where possible. The word "performance" is ambiguous in the context of registration. It is sometimes used to refer to the degree of success for a registration algorithm based on accuracy of the registration results. In this article, we use "performance" in its computational capacity referring to execution efficiency of the registration algorithm. The purpose of normalizing the reported results is to give the reader an indication of the speedups expected from a method without dependence on the size of images involved, convergence criteria, use of a multiresolution scheme, and to some extent the type of optimization algorithm. Normalized results are given in terms of average execution time in milliseconds for a single iteration of the optimization algorithm and for processing 1,000,000 voxel pairs (ms/MVoxel/itr).

Many authors have used gradient descent as their optimization algorithm, largely due to its simple structure and ease of implementation. Once the gradient is computed, the choices include taking a single step in a direction opposite to the gradient where the step size may be adjusted over time, or use of a line minimization algorithm such as Brent's [12]. Line minimization usually involves several computations of the cost function alone without its derivatives.

When comparing results it is important to identify which variation of the gradient descent is used. We have come across four different implementations:

- Type A: closed-form differentiation with a single step.
- Type B: closed-form differentiation with line minimization.
- Type C: numerical differentiation with a single step.
- Type D: numerical differentiation with line minimization.

Most authors exclude initialization time, including disk IO and loading data from host memory to GPU memory. This is a reasonable practice since initialization time is typically a small fraction of the registration task. Initialization occurs at the beginning of the registration algorithm whereas the optimization loop is executed several times.

Some of the information presented in Table 4 were not immediately available in the original manuscripts and were provided by the authors of the respective papers. Unless specifically specified, listed methods are for 3-D/3-D registration.

FINAL WORDS

Over the last decade, a rich and diverse literature on HPC of medical image registration has emerged. Research in this area continues to be motivated by the need to minimize the overhead of image registration that is used as an integral part of image-guided intervention and IGT systems. The continued research in this area will also facilitate the adaption of existing preoperative tools to real-time intraoperative environments.

From a technical perspective, there has been a gradual shift away from expensive SMP supercomputers to less expensive clusters of commodity computers and more recently inexpensive massively multiprocessing GPUs. This trend has the potential to lead to more widespread use of medical imaging tools in everyday clinical practice by making them affordable outside of research facilities and expensive operating theaters.

ACKNOWLEDGMENTS

We would like to thank Prof. Alistair Rendell for making staff and computational resources available for our experiments. We would also like to thank Ahmed El Zein and Benjamin Murphy for their efforts in porting the original code to Solaris, Linux, and its adaptation to Brook+.

We also thank many authors, whose work we have referred in this article, for providing additional information and for clarification of their results.

AUTHORS

Ramtin Shams (ramtin.shams@anu.edu.au) is an Australian postdoctoral Fellow in the College of Engineering and Computer Science at the Australian National University (ANU). He received his B.E. and M.E. degrees in electrical engineering from Sharif University of Technology, Tehran, and completed his Ph.D. degree at ANU in 2009 with a thesis in medical image registration. He was the recipient of a Fulbright scholarship in 2008. He has more than ten years of industry experience in the ICT sector and worked as the CTO of GPayments Pty. Ltd between 2001 to 2007. His research interests include medical image analysis, HPC, and wireless communications.

Parastoo Sadeghi (parastoo.sadeghi@anu.edu.au) is a Fellow (senior lecturer) at the Research School of Information Sciences and Engineering at ANU. She received her B.E. and M.E. degrees in electrical engineering from Sharif University of Technology, Tehran, and her Ph.D. degree in electrical engineering from The University of New South Wales in Sydney, in 2006. In 2003 and 2005, she received two IEEE Region 10 Paper Awards for her research in the information theory of time-varying fading channels. Her research interests include applications of signal processing, information theory, and HPC in telecommunications and medical image analysis.

Rodney A. Kennedy (rodney.kennedy@anu.edu.au) received his B.E. degree from the University of New South Wales, Australia, his M.E degree from the University of Newcastle, and his Ph.D. degree from ANU. For three years, he worked for the Commonwealth Scientific and Industrial Research Organization on the Australia Telescope Project. He is currently a professor and director of research at the College of Engineering and Computer Science at the ANU. His research interests are in the fields of signal processing, digital and wireless communications, and acoustical signal processing.

Richard I. Hartley (richard.hartley@anu.edu.au) is a member of the computer vision group in the College of Computer Science and Engineering at ANU. He also belongs to the Vision Science Technology and Applications Program in National ICT Australia. He graduated from the University of Toronto in 1976 with a thesis in knot theory and worked in this area for several years before joining the General Electric Research and Development Center, where he worked from 1985 to 2001. In 1991, he was awarded GE's Dushman Award for this work. In 2000, he coauthored a book on multiple view geometry. He has authored close to 200 scholarly papers and holds 32 U.S. patents.

REFERENCES

[1] L. G. Brown, "A survey of image registration techniques," *ACM Comput. Surv.*, vol. 24, no. 4, pp. 325–376, Dec. 1992.

[2] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," *Med. Image Anal.*, vol. 2, no. 1, pp. 1–36, 1998.

[3] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, "Mutual-information-based registration of medical images: A survey," *IEEE Trans. Med. Imag.*, vol. 22, no. 8, pp. 986–1004, Aug. 2003.

[4] J. Modersitzki, *Numerical Methods for Image Registration*. New York: Oxford Univ. Press, 2004.

[5] 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, Aug. 1999.

[6] T. M. Lehmann, C. Gönner, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Trans. Med. Imag.*, vol. 18, no. 11, pp. 1049–1075, Nov. 1999.

[7] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, "FAIR: A hardware architecture for real-time 3-D image registration," *IEEE Trans. Inform. Technol. Biomed.*, vol. 7, no. 4, pp. 426–434, Dec. 2003.

[8] P. Viola and W. M. Wells, III, "Alignment by maximization of mutual information," in *Proc. Int. Conf. Computer Vision (ICCV)*, June 1995, pp. 16–23.

[9] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, "Automated multimodality medical image registration using information theory," in *Proc. Int. Conf. Information Processing in Medical Imaging: Computational Imaging and Vision 3*, Apr. 1995, pp. 263–274.

[10] C. Studholme, D. L. G. Hill, and D. J. Hawkes, "An overlap invariant entropy measure of 3D medical image alignment," *Pattern Recognit.*, vol. 32, no. 1, pp. 71–86, 1999.

[11] A. Roche, G. Malandain, X. Pennec, and N. Ayache, "The correlation ratio as a new similarity measure for multimodal image registration," in *Proc. Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Oct. 1998, pp. 1115–1124.

[12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge, U.K.: Cambridge Univ. Press, 2007.

[13] J. A. Nedler and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–331, 1965.

[14] R. Shams, R. A. Kennedy, P. Sadeghi, and R. Hartley, "Gradient intensity-based registration of multi-modal images of the brain," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, Rio de Janeiro, Brazil, Oct. 2007.

[15] J. E. Dennis, Jr. and V. Torczon, "Direct search methods on parallel machines," *SIAM J. Optim.*, vol. 1, no. 4, pp. 448–474, 1991.

[16] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, pp. 157–181, 1993.

[17] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles*, vol. 10, no. 2, pp. 141–171, 1998.

[18] T. Butz and J-P. Thiran, "Affine registration with feature space mutual information," in *Proc. Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2001, pp. 549–556.

[19] M. P. Wachowiak and T. M. Peters, "High-performance medical image registration using new optimization techniques," *IEEE Trans. Inform. Technol. Biomed.*, vol. 10, no. 2, pp. 344–353, Apr. 2006.

[20] S. Warfield, F. Jolesz, 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, Sept. 1998.

[21] 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.

[22] (2009). *OpenMP application programming interface, version 3.0, OpenMP* [Online]. Available: http://openmp.org/wp/openmp-specifications/

[23] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. Cambridge, MA: MIT Press, 2008.

[24] T. Rohlfing and C. R. Maurer, Jr., "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, Mar. 2003.

[25] E. Lusk W. Gropp, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd ed. Cambridge, MA: MIT Press, 1999.

[26] 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, Jan. 2005.

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

[28] G. C. Sharp, N. Kandasamy, H. Singh, and M. Folkert, "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," *Phys. Med. Biol.*, vol. 52, no. 19, pp. 5771–5783, 2007.

[29] 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.

[30] Y. Lin and G. Medioni, "Mutual information computation and maximization using GPU," in *Proc. IEEE Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2008, pp. 1–6.

[31] 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.*, Nov. 2008, pp. 53–56.

[32] P. Muyan-Özçelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," in *Proc. Int. Conf. Computational Science and Its Applications (ICCSA)*, 2008, pp. 5–8.

[33] R. Shams, P. Sadeghi, R. A. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," *Comput. Meth. Programs Biomed.*, to be published.

[34] A. Ruiz, M. Ujaldon, L. Cooper, and K. Huang, "Non-rigid registration for large sets of microscopic images on graphics processors," *J. Signal Process. Syst.*, vol. 55, no. 1-3, pp. 229–250, Apr. 2009.

[35] G. Soza, M. Bauer, P. Hastreiter, C. Nimsky, and G. Greiner, "Non-rigid registration with use of hardware-based 3D Bézier functions," in *Proc. Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2002, pp. 549–556.

[36] R. Strzodka, M. Droske, and M. Rumpf, "Image registration by a regularized gradient flow. A streaming implementation in DX9 graphics hardware," *Computing*, vol. 73, no. 4, pp. 373–389, Nov. 2004.

[37] A. Khamene, R. Chisu, W. Wein, N. Navab, and F. Sauer, "A novel projection based approach for medical image registration," in *Proc. 3rd Int. Workshop Biomedical Image Registration (WBIR)*, Utrecht, The Netherlands, June 2006, pp. 247–256.

[38] F. Ino, J. Gomita, Y. Kawasaki, and K. Hagihara, "A GPGPU approach for accelerating 2-D/3-D rigid registration of medical images," in *Proc. Parallel and Distributed Processing and Applications*, Feb. 2006, pp. 939–950.

[39] C. Vetter, C. Guetter, C. Xu, and R. Westermann, "Non-rigid multi-modal registration on the GPU," in *Proc. SPIE Medical Imaging: Image Processing*, Feb. 2007, pp. 651228-1–651228-8.

[40] Z. Fan, C. Vetter, C. Guetter, D. Yu, R. Westermann, A. Kaufman, and C. Xu, "Optimized GPU implementation of learning-based non-rigid multi-modal registration," in *Proc. SPIE Medical Imaging: Image Processing*, 2008.

[41] N. Courty and P. Hellier, "Accelerating 3D non-rigid registration using graphics hardware," *Int. J. Image Graph.*, vol. 8, no. 1, pp. 1–18, Jan. 2008.

[42] A. Kubias, F. Deinzer, T. Feldmann, S. Paulus, D. Paulus, B. Schreiber, and T. Brunner, "2D/3D image registration on the GPU," *Pattern Recognit. Image Anal.*, vol. 18, no. 3, pp. 381–389, Sept. 2008.

[43] (2009). *Compute unified device architecture (CUDA) programming guide, version 2.2, NVIDIA* [Online]. Available: http://developer.nvidia.com/object/cuda.html

[44] (2009). *ATI stream computing user guide, version 1.4.0.a, ATI* [Online]. Available: http://developer.amd.com/

[45] W. Plishker, O. Dandekar, S. S. Bhattacharyya, and R. Shekhar, "Utilizing hierarchical multiprocessing for medical image registration," *IEEE Signal Processing Mag.,* vol. 27, no. 2, pp. 62–68, Mar. 2010.

[46] M. Ohara, H. Yeo, F. Savino, G. Iyengar, L. Gong, H. Inoue, H. Komatsu, V. Sheinin, and S. Daijavad, "Accelerating mutual-information-based linear registration on the cell broadband engine processor," in *Proc. IEEE Int. Conf. Multimedia and Expo*, 2007, pp. 272–275.

[47] M. Ohara, H. Yeo, F. Savino, G. Iyengar, L. Gong, H. Inoue, H. Komatsu, V. Sheinin, S. Daijavad, and B. Erickson, "Real-time mutual-informatoin-based linear registration on the cell broadband engine processor," in *Proc. IEEE Int. Symp. Biomedical Imaging (ISBI)*, 2007, pp. 33–36.

[48] J. Rohrer and L. Gong, "Accelerating mutual information based 3D non-rigid registration using the cell/B.E. processor," in *Proc. Workshop on Cell Systems and Applications (WCSA)*, 2008, pp. 32–40.

[49] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, "FPGA-based acceleration of mutual information calculation for real-time 3D image registration," in *Proc. SPIE Medical Imaging: Image Processing*, 2008, pp. 212–219.

[50] 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, June 2007.

[51] A. Köhn, J. Drexl, F. Ritter, M. König, and H. O. Peitgen, "GPU accelerated image registration in two and three dimensions," in *Proc. Bildverarbeitung für die Medizin*, 2006, pp. 261–265.