Published in final edited form as:
J Signal Process Syst. 2009 April 1; 55(1-3): 229–250. doi:10.1007/s11265-008-0208-4.

Non-rigid Registration for Large Sets of Microscopic Images on Graphics Processors

Antonio Ruiz,
Computer Architecture Department, Campus Teatinos, University of Malaga, 29071 Malaga, Spain

Manuel Ujaldon,
Computer Architecture Department, Campus Teatinos, University of Malaga, 29071 Malaga, Spain

Lee Cooper, and
Biomedical Informatics Department, Ohio State University, 333 West 10th Avenue, Columbus, OH 43210, USA

Kun Huang
Biomedical Informatics Department, Ohio State University, 333 West 10th Avenue, Columbus, OH 43210, USA

Antonio Ruiz: aruiz@ac.uma.es; Manuel Ujaldon: ujaldon@ac.uma.es; Lee Cooper: cooperl@ece.osu.edu; Kun Huang: khuang@bmi.osu.edu

Abstract

Microscopic imaging is an important tool for characterizing tissue morphology and pathology. 3D reconstruction and visualization of large sample tissue structure requires registration of large sets of high-resolution images. However, the scale of this problem presents a challenge for automatic registration methods. In this paper we present a novel method for efficient automatic registration using graphics processing units (GPUs) and parallel programming. Comparing a C++ CPU implementation with Compute Unified Device Architecture (CUDA) libraries and pthreads running on GPU we achieve a speed-up factor of up to 4.11× with a single GPU and 6.68× with a GPU pair. We present execution times for a benchmark composed of two sets of large-scale images: mouse placenta (16*K* × 16*K* pixels) and breast cancer tumors (23*K* × 62*K* pixels). It takes more than 12 hours for the genetic case in C++ to register a typical sample composed of 500 consecutive slides, which was reduced to less than 2 hours using two GPUs, in addition to a very promising scalability for extending those gains easily on a large number of GPUs in a distributed system.

Keywords

Microscopic imaging; Image registration and segmentation; Pattern analysis; Feature detection; Graphics processors; High-performance computing

Correspondence to: Kun Huang, khuang@bmi.osu.edu.

1 Introduction

Characterizing the phenotypes associated with specific genotypes is critical for elucidating the roles of genes and gene interactions. Cellular and tissue structure morphologies are aspects of phenotype that provide information necessary for understanding important biological processes such as cancer initiation in the tumor microenvironment and the formation of neural networks in the brain cortex. Nevertheless, existing techniques for obtaining high magnification 3D information from biomedical samples are rather limited. The commonly used confocal/multiphoton fluorescent microscopy provide three dimensional information but require the use of fluorescent markers and have limited field of view. Therefore another fundamental approach to gather 3D information is to perform reconstruction using 2D images obtained from tissue sectioning and light microscope. The basis for this approach is the alignment of 2D images of thin serial sections using *image registration* [1–22].

The key challenge for image registration in this scenario is to compensate for the distortion between consecutive section images that is introduced by the sectioning process. Tissue sections are extremely thin (3 to 5 μm) and delicate as a result. The preparation process (i.e., sectioning, staining, and coverglass application) can introduce a variety of nonrigid deformations including bending, shearing, stretching, and tearing. At micron resolutions, even minor deformations become conspicuous and may prove problematic when accuracy is critical to the end application. In order to compensate for such deformations, a *nonrigid registration* is essential and success depends on establishing a large number of precise *feature correspondences* throughout the extent of the image. This precision requires comparison of intensity information and is very time consuming with popular comparison measures such as Mutual Information (MI).

In this paper, we develop an effective high-performance computing method for generating precise feature correspondences between microscopic images at the scale of hundreds of millions to billions of pixels. For feature selection and matching in this context, there are several challenges:

1. **Feature rich environment.** The textural quality of microscopic image content provides a unique challenge to the problems of feature selection and matching. Traditional feature detection schemes such as corner detection generate an overwhelming abundance of features that are regular in appearance making matching unrealistic.
2. **Large-scale images.** Currently high-resolution slide scanners can generate images with resolutions of 0.46 μm/pixel (with 40× objective lens) and possibly higher. At this resolution, a shift of 1 mm corresponds to around 2,000 pixels. The search for corresponding features are therefore almost unfeasible without a good initialization.
3. **Heavy computational requirements.** A common approach for the registration of large images is to conduct multiscale registration where the images are down-sampled to multiple low-resolution images so that nonrigid transformation can be computed and applied to the images with the next higher resolution. This requires

transformation of the free-floating image at each scale which is a nontrivial computational task. For instance, in this paper we are dealing with images with sizes up to $23K \times 62K$ pixels. With a scale factor of two, this implies transformation of an image containing around $12K \times 31K$ pixels prior to the final stage.

In order to deal with these challenges, we develop a high-performance computing approach to registration comprised of fast rigid registration for initialization and an efficient and parallelizable algorithm for nonrigid registration with implementation on graphics processing units (GPUs). Our approach has the following advantages:

1. **GPU Implementation.** Most importantly, we take advantage of the computing capacity of the GPU, which has become a cost-effective parallel platform to implement grand-challenge biomedical applications [23, 24]. Compute unified device architecture (CUDA) offers an alternate programming model to the underlying parallel graphics processor without requiring a deep knowledge about rendering or graphics. The interface uses standard C code with parallel features to transform the GPU technology to massive parallel processors for commodity PCs.
2. **Fast Rigid Registration for Initialization.** The nonrigid registration algorithm is initialized by a fast rigid registration algorithm that we have previously developed. This algorithm uses conspicuous anatomical regions (e.g., blood vessels) as high-level features and the rigid transformation is derived using a voting scheme in the Euclidean transformation space. It is highly efficient and accurate for common histological images. In addition, it can accommodate arbitrary rotation and translations. This provides us with a good initialization for the nonrigid registration and the search space for point correspondence is significantly reduced.
3. **Feature Selection.** Point features for precise matching in nonrigid registration are selected based on neighborhood complexity rather than the presence of geometric content such as corners. This not only reduces computational burden but also allows the user to gain a more uniform distribution of features.
4. **Fast Normalized Cross-Correlation (NCC) for Precise Matching.** Precise feature matching is based on the NCC between tile neighborhoods in each image. NCC calculation can be implemented efficiently using fast Fourier transform (FFT) resulting in a very fast execution as compared to measures like MI. Additionally NCC has an intuitive interpretation which simplifies the selection of threshold parameters used to discriminate good matches from bad.
5. **Single Transformation Output.** For precise matching, the Euclidean transformation parameters obtained from rigid initialization are used to locate and transform corresponding neighborhoods to avoid applying an expensive rigid transformation to an entire image.
6. **Parallelization for Precise Matching.** The process of precise matching is embarrassingly parallel, lending itself to execution on multiple cores, sockets, or a computing cluster.

GPU implementation is focused on the most computationally demanding portion of our algorithm: the calculation of cross-correlations for precise matching, which require up to 60% of total computation time. We demonstrate results for our method by comparing serial and parallel implementations on both CPU and GPU, using a variety of parameter choices to explore the efficiency and scalability of our approach (see Table 4.) Our benchmark of image datasets (see Table 5) is taken from two of our quantitative phenotyping projects. The first project is a morphometric study on the role of the retinoblastoma gene (a well-known tumor suppressor) in mouse placenta development. In this study, three control placentas and three mutant placentas with Rb gene deletion were obtained. Each sample was sliced into 5 μm sections and each section was stained using standard hematoxylin and eosin staining. The stained sections were digitized using an Aperio ScanScope high resolution scanner with a 20× objective lens which produces a resolution of 0.46 μm/pixel. The six samples yielded more than 3,000 images with typical dimensions $16K \times 16K$ pixels for a total of more than three terabytes (uncompressed) of data. The second project is part of ongoing work studying the breast cancer tumor microenvironment in mice. Images from this study are typically $23K \times 62K$ pixels and around four gigabytes in uncompressed form.

The paper is organized as follows: Sections 2 through 4 provide an overview of the image registration method including a review of our fast rigid registration algorithm in Section 2, a description of the feature selection and precise matching process in Section 3, and a brief overview of image transformation in Section 4. A summary of GPU architecture and a description of our implementation is provided in Section 5. The experimental setup is presented in Section 6. Performance results and analysis are contained in Section 7. The paper concludes in Sections 8 through 10 with a discussion on related work and future directions.

2 Fast Rigid Registration for Initialization

Although the development of our rigid registration algorithm is outside the scope of this paper, we provide a brief review of it here for completeness as it is an important part of our high-performance approach to nonrigid registration; the initialization provided by this rigid registration reduces the extent of search used for precise matching in nonrigid registration and in turn significantly reduces computation.

The basis of the rigid registration algorithm is the matching of *high level features*, regions that correspond to specific anatomical structures such as blood vessels or mammary ducts. There are several advantages to using high level features over the more primitive features generated by methods such a corner detection. First, high-level features are easy to segment as they often correspond to relatively large contiguous regions of pixels with similar colors. All that is required for extraction is a simple color classification of pixels and a sequence of morphological operations [25], both of which admit optimized implementations in commonly available image-processing libraries. This extraction can also often be performed on downsampled representations of the original images without loss of fidelity. For instance, to calculate rigid registration we typically operate on 5× magnification decimations of 20× images. Since the rigid registration only serves as an initialization to the nonrigid stage, which operates on the original full magnification images, the loss of information and

aliasing due to downsampling does not affect the final results. Second, even in microscopic images the number of high-level features is usually limited, keeping the number of possible matches reasonable. Finally, these features can be matched based on characteristics such as shape and size, and are therefore "global" meaning that the matching process is not limited to local search and can therefore accommodate the full range of possible misalignments between images. The use of characteristics such as size and shape as matching criteria also reduces match ambiguity.

The key issue for any feature matching scheme is the detection of mismatches. To this end, we have developed two approaches. Any two pairs of matched features can generate a purported rigid transformation specified by the rotation angle θ and translation T if their intra-image distances are consistent. It can be conceived that a large portion of the transformations based on likely matchings should concentrate around the true parameters in the Euclidean transformation space. Therefore, a *voting* process can be adopted to select the optimal rigid transformation that agrees with the majority of matchings. An example of this process is shown in Fig. 1. In the case where the total number of features present in the image is small (e.g., less than 10), the results of voting may be less reliable. For this reason, we have also developed a graph-theoretic approach for feature matching that is based on a similar principle [20].

This algorithm has reasonable execution times even with a modest implementation. Using Matlab, the rigid parameter estimates for the example in Fig. 1 were calculated in less than 4 s on a system similar to the one described in Section 6.

3 Feature Extraction and Matching

With the rigid initialization described, we turn our attention to the nonrigid registration process. Correcting nonrigid distortion to the accuracy necessary for end applications in quantitative phenotyping requires the establishment of a large number of precise correspondences. These correspondences should also be fairly distributed throughout the areas of the image that are of interest, in order to produce a result that has uniform quality. We address these considerations in our approach to feature extraction and matching where we use sampling and rigid initialization parameters to identify corresponding tile regions and compare them using NCC.

3.1 Feature Extraction

The first issue in feature extraction is the selection of unambiguous features that are likely to result in specific and accurate matchings. This is especially important for matching in nonrigid registration since using the collection of matches to infer something about the quality of any individual match is difficult due to the freedom and minor scale of nonrigid distortion. In this sense the matchings in nonrigid registration are local in nature: the only information available to judge their quality comes from the feature neighborhood.

In our case, we want to select tiles that have rich content, a mixture of different tissues or tissue and background that form a distinctive appearance and are likely to produce very specific matches. To enforce this, we select tiles whose variance meets a certain minimum

threshold. That is, for any feature point p with coordinate $\begin{bmatrix} x \\ y \end{bmatrix}$ centered in the $W \times W$-pixel window we require

$$\frac{1}{W^2-1}\sum_{i,j}(t(i,j)-\bar{t})^2 \geq \sigma^2 \quad (1)$$

where t is the *template*, a grayscale representation of the p-centered pixel window with mean value $\bar{t}$, and σ^2 is a variance threshold. There are cases where the variance threshold can be met and an ambiguous result can still occur (consider matching a small template with upper half white and lower half black inside a similar but larger template) although these type of cases are hardly ever encountered in practice.

To keep the total number of features reasonable and attempt even feature distribution, we sample features uniformly over the image space with a $W \times W$ tiling. For example, in the $16K \times 16K$ placenta images, we would typically tile in the range of 150–350 pixels to generate a total of 2025–11236 possible features, the large majority of features are discarded due to insufficient variance. With a set of features identified in one image, their unique correspondences in the next image are simple to determine.

3.2 Feature Matching

For any selected feature point p_1 with coordinate $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ in the first image, we first select a $B \times B$-pixel window centered at p_1. This window is transformed to grayscale and rotated by the angle θ obtained from the rigid registration initialization. The central $W_1 \times W_1$-pixel patch is then used as the $p1$ matching template for identifying p_2, the correspondence point of p_1 in the second image. B is calculated from θ and W_1, taken large enough to accommodate the template.

The coordinate p_2 can be estimated using

$$p_2^{'} = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \mathbf{T} \quad (2)$$

with $\mathbf{T}$ being the translation vector obtained from the rigid registration initialization. A larger $W_2 \times W_2$-pixel template centered at $p_2^{'}$ and designated as the *search window* is taken from the second image. The NCC between the template and search window is computed, and the center of the area corresponding to the highest NCC value is set as p_2. If this maximum value exceeds a threshold (usually 0.8 or greater), then the match is considered successful and $p1$, $p2$ are recorded as a correspondence.

This process is illustrated in Fig. 2. The choice of W_1 and W_2 is based on the severity of the deformation as well as computational capacity. Empirically we set $W_2 = 2W_1$. A large deformation requires a large search window size of W_2. We tested different choices of W_1

and W_2 during our experimental evaluation (see Section 7), with some of them favouring the CPU and/or the GPU (see Table 4).

Commonly used similarity measures for intensity information other than NCC include summed square of difference (SSD) and MI. SSD is not a good choice for microscopic images since the content tends to be discrete (e.g., sharp boundaries between cell nucleus and cytoplasm and cell membrane). MI is commonly used as a metric in gradient search strategies but the cost of joint-histogram computation makes exhaustive search very computationally expensive. We choose NCC since it is not only robust in identifying structural similarity but is also highly efficient when implemented with FFT. Robustness is key in our application since intensity variations that are introduced by non-uniform section thickness and staining time are commonly encountered. Furthermore, NCC values have an intuitive interpretation, making threshold parameter selection easier.

3.2.1 Computing NCC—The large number of features that exist within a typical dataset makes efficient computation of NCC critical. Additionally, rather than using a search strategy, we compute NCC between template and search window pairs at all spatial offsets to avoid the problem of local minima.

Given a template t size $W_1 \times W_1$ with mean $\bar{t}$, and search window s size $W_2 \times W_2$, $W_2 > W_1$, the NCC between t and s is the quotient of covariance and individual variances

$$\rho(u,v)=\sum_{x,y}\frac{\{t(x-u,y-v)-\bar{t}\}\ \{s(x,y)-\bar{s}_{u,v}\}}{\left(\{t(x-u,y-u)-\bar{t}\}^2\{s(x,y)-\bar{s}_{u,v}\}^2\right)^{\frac{1}{2}}} \quad (3)$$

where $\bar{s}_{u,v}$ is the mean of the search window portion overlapping the template at offset (u, v).

For calculating normalizing factors in the denominator we use the method of running sums presented in [26]. This avoids the expensive local calculations of search window mean and variance for the template overlap region as the template is shifted through each of $(W_1 + W_2 - 1)^2$ positions, reducing the operation count from $3W_2^2(W_1-W_2+1)^2$ to approximately $3W_1^2$.

The unnormalized cross-correlation from the numerator is calculated via the convolution theorem of the Discrete Fourier Transform that relates the product of DFT spectra to circular convolution in the spatial domain. For cross correlation we are interested in ordinary convolution so t and s are padded with zeros to size $W_1 + W_2 - 1$ prior to forward transform to ensure that the circular overlap portions of the convolution result are null. Figure 3 shows some randomly selected examples of the matched regions.

4 Image Transformation

The collection of point correspondences generated by the precise matching process provides the information needed to calculate a mapping that transforms one image into conformation with the other. A variety of nonrigid mappings are used in practice, differing in

computational burden, robustness to erroneous correspondences, and existence of inverse form. High-performance implementation of the transformation process is a non-trivial task and is not addressed in this paper.

4.1 The Polynomial Transformation

In choosing a transformation type, we seek something that is capable of correcting complex distortions, robust to match errors, admits a closed inverse form and is computationally efficient to apply. Of the commonly used nonrigid mapping types such as thin-plate spline, local weighted mean, affine, polynomial, and piece-wise variations we choose polynomial mapping. Thin plate spline provides a minimum energy mapping which is appealing for problems involving actual physical deformation, however perfect matching at correspondence locations can potentially cause large distortion in other areas and excessive mapping error if an erroneous correspondence exists. The lack of an inverse form means the transformed image must be calculated in a forward direction, likely leaving holes in the transformed result. Methods such as gradient search can be used to overcome this but at the cost of added computation which can become astronomical when applied for the transformation at every pixel in a gigapixel image. Kernelbased methods such as local weighted mean require a uniform distribution of correspondences. Given the heterogeneity of tissue features this distribution cannot always be guaranteed.

Polynomial warping admits an inverse form, is fast in application, and from our experience is capable of satisfactorily correcting the distortion encountered in sectioned images. Polynomial warping parameters can be calculated using least squares or least squares variants which can mitigate the effect of matching errors. Affine mapping offers similar benefits but is more limited in the complexity of the warpings it can represent.

In our algorithm, we use second degree polynomials. Specifically, for a point *(x, y)* in the reference image, the coordinate *(x′, y′)* of its correspondence in the second image is

$$\begin{cases} x' = a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1, \\ y' = a_2x^2 + b_2xy + c_2y^2 + d_2x + e_2y + f_2, \end{cases} \quad (4)$$

Since each pair of matched point correspondence provide two equations, we need at least six pairs of point correspondences to solve for the coefficients in Eq. 4. In practice, we obtain a much larger number of point correspondences.

4.1.1 3D reconstruction—For 3D reconstruction applications, where a sequence of images are to be registered, the matching process is applied successively to each ordered pair in the sequence. Images are transformed starting at one end of the sequence and at each stage the transformations from prior image pairs are propagated in order to achieve a coherent set of transformed images.[1]

[1]The pairwise registration in some cases is not sufficient for 3D reconstruction. The 3D structural constraints may have to be imposed and the discussion is outside the scope of this paper.

The workflow for the entire registration process is summarized in Fig. 7. An example of the reconstruction results from 50 sections of a mutant placenta are illustrated in Fig. 10. Due to the limitations of the rendering software in handling sets of large images, only a portion of the reconstruction can be rendered at any one time.

5 Implementation

The performance of algorithms on GPUs depends on how well they can exploit parallelism, closer memory, bus bandwidth, and GFLOPS.

Parallelism: Programs running on GPU are decomposed into threads and are executed on a massively parallel multiprocessor composed of 128 cores or stream processors (see central row in Fig. 4).

Memory access: Data is stored on L1 caches, L2 caches and video memory (see lower rows in Fig. 4), with closer memory being faster. Spatial locality is best exploited by caches, which are around a thousand times larger on the CPU, whereas temporal locality benefits the GPU, whose architectural rationale and programming model are inspired by the producer/consumer paradigm.

Bus bandwidth: A state-of-the-art 2008 graphics card delivers a peak performance memory bandwidth around 80 GB/s, as compared to 10 GB/s for CPU. This is mainly due to its wider data path (384 bits, decomposed into six partitions of 64 bits in Fig. 4).

Computational units: The GPU capacity for floating-point operations exceeds 500 GFLOPS, in contrast with around 10 GFLOPS for a 2008 state-of-the-art CPU. This advantage is a result of design for the color and position interpolations that are required for performance graphics applications.

We combine these outstanding features of the GPU with CPU to create a bi-processor platform that balances workload and optimizes the execution of our nonrigid registration. The rest of this section focuses on the GPU implementation.

5.1 The CUDA Programming Model

The CUDA[27] programming interface consists of a set of library functions which can be coded as an extension of the C language. The CUDA compiler generates executable code for the GPU, which is seen as a multicore processor resource by the CPU. CUDA is designed for generic computing and hence it does not suffer from constraints when accessing memory, though the access times vary for different types of memory.

5.1.1 Computation Paradigm—General-purpose on GPUs (GPGPU) [28] is designed to follow the general flow of the graphics pipeline (consisting of vertex, geometry and pixel processors – see Fig. 4), with each iteration of the solution being one rendering pass. The CUDA hardware interface (see Fig. 5) attempts to hide all these notions by presenting a program as a collection of threads running in parallel. The elements for this approach are:

- A **warp** is a collection of threads that can actually run concurrently (no time sharing) on all of the multiprocessors. The size of the warp (32 on the G80 GPU) is less than total available cores (128 on G80) due to memory access limitations. The programmer decides the number of threads to be executed, but if there are more threads than the warp size, they are time-shared.
- A **block** is a group of threads that are mapped to a single multiprocessor. Since each multiprocessor has multiple cores (8 on the G80) and a shared memory, threads in a block are executed together and can efficiently share memory. All threads of a block executing on a single multiprocessor divide its resources equally amongst themselves, with each thread and block having a unique ID accessed during its execution to process different sets of data in a SIMD (Single Instruction Multiple Data) fashion.
- A **kernel** is the core code to be executed on each thread, which performs on different sets of data using its ID. The CUDA programming model does not allow you to select a different kernel to be executed on each of the multiprocessors. The hardware architecture, however, allows multiple instruction sets to be executed on different multiprocessors, so this may be simulated using conditionals.
- A **grid** is a collection of all blocks in a single execution. That way, a program is decomposed into kernels, each implemented through a grid which is composed of blocks consisting of threads (see Fig. 6).

A single block should contain 128–256 threads for an efficient execution. The maximum possible thread total is 512. Other hardware limitations are listed on Table 1, where we have ranked them according to impact on the programmer's job and overall performance based on our experience.

5.1.2 Memory and Registers—In CUDA, all threads can access any memory location, but as expected, performance will increase with the use of closer shared memory whenever data to be collectively read by threads within a block belong to different memory banks. The use of shared memory is explicit within a thread and cannot exceed 16 Kbytes. Optimizations using shared memory may speed-up the code up to a 10× factor for vector operations, and latency hiding up to 2.5× [29]. Other performance issues are summarized in the last two rows of Table 2.

The role of 32-bit registers becomes more important as a limiting factor for the amount of parallelism we can exploit, rather than as the conventional mechanism to hide memory latency. A multiprocessor contains 8192 registers, each owned exclusively by a thread. Registers should be split among the threads so that the number of threads created reaches the maximum occupancy on each multiprocessor given the constraints outlined in Tables 1 and 2. For example, if a thread consumes 10 or less registers, we may have up to 819 threads, but only 768 are allowed on a multiprocessor and only 512 are allowed for a block: A possible solution is to build 3 blocks of 256 threads each. Reversely, if a thread consumes 16 registers, a maximum of 512 threads is allowed ($512 \times 16 = 8192$), and all threads may belong to a single block.

5.1.3 Developing in CUDA—A typical CUDA development cycle is as follows. First, the code is compiled using a special CUDA compiler that outputs the hardware resources (registers and local shared memory) that are consumed by the kernel. Using these values, the programmer determines the number of threads and blocks that are needed to use a multiprocessor efficiently. If a satisfactory efficiency cannot be achieved, the code needs to be revised to reduce the memory foot print (registers and local shared memory). Due to the high FLOPS performance of the streaming processor, memory access becomes the bottleneck in our registration algorithm.

5.2 Image Registration on the GPU

The workflow for our registration algorithm is summarized in Fig. 7. From a performance perspective, the most interesting phase is the set of FFTs used to compute the NCCs for precise point matching in nonrigid registration, since they entail most of the execution time. For example, in our experiments registering a pair of 23K × 62K images on the CPU, more than 60% of the total running time is spent in computing NCC.

We optimize this process by implementing it on the GPU (see section 5.2.1 below), including the two forward FFTs and the subsequent inverse. The point-wise multiplication of FFT spectra which is required between the forward and inverse transforms was also implemented on the GPU to save data movement between processors and to take advantage of higher arithmetic intensity versus computation on the CPU (see Table 6). Table 3 depicts the percentage weight for these operations on each platform.

The remaining parts of the registration algorithm including voting, variance calculations, and simple transformations (e.g. rotating) did not show any significant speed-up on the GPU for three major reasons:

1. They were already computationally cheap on the CPU.
2. More importantly, it was remarkable how much time was required to ship code and data back and forth between the CPU and the GPU through the memory bus, hypertransport link, and PCI-express controller (see Fig. 9). This cost could not be amortized during the subsequent computation despite of the high GFLOPS rate.
3. Most of these operations contain conditionals and are not arithmetic intensive, which makes them more appropriate for the CPU processor. Additionally, this enables our bi-processor platform for a more balanced execution.

5.2.1 NCC Using CUDA—The NCC can be efficiently implemented on the GPU using the CUDA programming model. Our computation strategy is based on the theorem that circular convolution in geometric space amounts to point-wise multiplication in discrete frequency space. This way, using the CUFFT library [30] as an efficient direct/inverse FFT implementation, Fourier-based correlation can be more efficient than a straightforward spatial domain implementation, and allows us to leverage the floating-point power and parallelism of the GPU without having to develop a custom, GPU-based implementation.

The FFT is a highly parallel “divide and conquer” algorithm for the computation of the Discrete Fourier Transform of single or multidimensional signals. The convolution theorem applies to an image (search window) and convolution kernel (template window) that share the same sizes. In cases like ours where the image is bigger than the kernel, the kernel has to be expanded to the image size as shown in Fig. 8. Also, ordinary convolution requires the template and search windows to be padded with zeros on the bottom and right borders as anticipated in Section 3.2.1.

The 2D-FFT dimensions are fundamental in CUDA for optimizing performance. When the template and search window are multiples of either a power of two or a small prime number, the memory footprint generated by the CUDA algorithm minimizes conflicts accessing banks on shared memory and performance increases. For the counterpart C++ implementation on the CPU we have used FFTW [31], one of the most popular and efficient CPU-based FFT libraries, for a fair comparison with the GPU results. FFTW also favours certain 2D-FFT dimensions, and the optimal cases arise when the sum of the template window and the search window sizes minus one is a power of two. With a careful selection of FFT dimensions, we have created a benchmark fulfilling most of these rules on both CPU and GPU implementations. Table 4 summarizes all sizes selected for our experimental evaluation and evaluates their adequacy for each type of processor.

For the cases in which the data size cannot fulfill the previous rules, FFTW and CUFFT provide a simple configuration mechanism called a *plan* that completely specifies the optimal - that is, the minimum floating-point operation - plan of execution for a particular FFT size and data type. The advantage of this approach is that once the user creates a plan, the library stores on file whatever state is needed to execute the plan multiple times, thus avoiding the penalty of carefully planning the transforms at run-time. For example, with a template window equal to 350 × 350 pixels and a search window equal to 700 × 700 pixels, FFTW takes around 0.7 s, whereas the pre-planned computation takes only 0.32 s with a previous 6 s penalty required to pre-compute the plan (a cost which can later be amortized by loading the plan at run-time on subsequent 2D transforms of the same size).

6 Experimental Setup

6.1 Input Data Set

In order to evaluate our optimization techniques and implementation methods, we applied our registration algorithm to a series of microscopic images of consecutive sections of (1) mouse placenta for a morphometric study on the role of the retinoblastoma gene and (2) mammary gland for studying the breast cancer tumor microenvironment [16]. For details about these sets of images, see Table 5. The goal in both cases is to reconstruct 3D tissue models for the study of microanatomy.

6.2 Hardware

Our application for automatic registration was implemented on a GPGPU visualization node where the features of dual-core AMD Opteron 2218 CPU are combined with dual-socket high-end Nvidia Quadro FX 5600 GPU (see Fig. 9). The CPU is endowed with 4 GB of DDR2 DRAM running at 667 MHz, whereas each of the dual GPUs contains 1.5 GB of on-

board GDDR3 DRAM at 1600 MHz (see remaining features in Table 6). This leads to a total available DRAM memory of 7 GB. The system is completed with a 750 GB, 7200 RPM local SATA II hard disk with 16 MB cache and an InfiniBand card for communication purposes.

In our experiments, we do not consider the time for reading the input images from file. This time can be partially hidden by overlapping input/output (I/O) communications with internal computations on the GPUs due to the asynchronous communications supported within CUDA 1.1. In addition, we have seen shared I/O slightly affecting the computational time. To minimize this variation, several runs were performed for each experiments, taking the average among all of them.

6.3 Software

The GPU was programmed using the CUDA Programming Toolkit, version 1.1 (December, 2007), and for the cases where we used two GPUs, *pthreads* were used to run the code on each GPU.

On the CPU side, we used the Microsoft Visual Studio 2005 8.0 C++ compiler. Matlab 7.1 was also used to validate the results from our implementation as well as to provide the departure sequential execution time.

7 Empirical Results

A broad number of experiments were conducted on one hundred images in the placenta image set and four images for the mammary image set as reflected in Table 5.

7.1 Image Registration Results

Figure 10 demonstrates an example of the 3D reconstruction of part of the mouse placenta using 50 slides and the matched features before and after the registration. Evaluation of registration quality for microscopic images is a difficult task due to the absence of ground truth. Although validation is a complex problem that is outside the scope of this paper, we present results here for visual inspection to motivate the necessity of nonrigid registration. Figure 11a–d show the difference between rigid and nonrigid registration by overlaying the two images. Clearly the rigid registration alone cannot adequately compensate for the nonrigid deformations encountered in our application. In another study [32], we noted that after nonrigid registration discrepancies at inspected areas of interest were typically within ten pixels (some discrepancy is expected since there are morphological differences between adjacent tissue sections). Nevertheless, an ultimate evaluation is on whether the registration algorithm can lead to realistic 3D reconstruction of the samples. Figure 11e and f demonstrate renderings of 3D reconstructions from registered placenta images, both before and after nonrigid registration. The virtual cross-sections demonstrate the necessity for nonrigid registration in this case. With rigid registration alone, no coherent structures are apparent while the nonrigid registration leads to smooth structures in the 3D reconstruction.

7.2 Characterizing the Workload

A preliminary issue to mention is that the execution time for each slide within the same working image set experiences variations due to the content and consequentially the different number of features processed. As we described in Section 3.1, the variance is computed on a 200 × 200-pixel window to retain only feature points that are meaningful. This may lead images of similar sizes to produce different workloads based on their contents (the more homogeneous an image is, the less computation required). Table 7 summarizes the number of features extracted for each input image belonging to the mammary data set as well as the total and computational time required for the registration algorithm to be completed on an Opteronb CPU.

The percentage of features processed ranges from 4% to 30% of total image area, with those percentages varying slightly when using small, medium or large window sizes (see Table 4). However we may consider them stable for each image if we select the smaller window size as the most representative (higher search resolution). Under this assumption, Fig. 12 provides details about the percentage of features processed for the placenta and mammary image sets: For the placenta images the minimum percentage corresponds to image 5 with 10.48% and the maximum to image 99 with 30.38%, and a total average of 19.88%. For the mammary gland images the minimum percentage is 4.82% by image 4, with a maximum of 20.71% by image 3, and an average of 10.77%. According to our definition of feature, the placenta image set constraints nearly double the density of meaningful information. While the mammary gland set is a larger image, it represents a matrix with a higher sparsity rate.

7.3 Execution Times on the CPU

Figure 13 presents the execution time for the registration algorithm depicted in Fig. 7 when it is entirely computed on the CPU using the FFTW library. On the left, we show the results for the placenta image set, mammary is on the right. Within each case, we run experiments for three different template and search windows (see Table 4): small (leftmost), medium (center) and large (rightmost). According to the hints provided by the FFTW library, the small and large sizes fulfill optimal conditions, whereas the medium size breaks all rules (from now on, this case will be referred to as *non-compliant*). This has a major impact on the execution time, with an average time for the placenta case of 294.57 s using the medium size versus 57.97 s in the small case and 91.33 s in the large one. This results in an increment of 57% when the windows are doubling size within optimal conditions and an additional 222% when using non-compliant sizes. Mammary offers a similar behavior, though the last two overheads are reduced to 26% and 147% respectively.

7.4 Execution Times on the GPU

Figure 14 shows execution times for the registration algorithm when the GPU helps the CPU by computing the FFT-based cross-correlation using CUDA. The left side represents the placenta image set and the right side the mammary image set, with the legend differentiating the small, medium and large window size cases (see Table 4). This time, the small and large sizes fulfill all conditions imposed by the CUFFT library and also the medium search window size of 749 pixels satisfies being a multiple of a small prime number (7). Nevertheless, its overhead is still significant. The average times for the placenta case are

19.27 s (small), 47.80 (medium) and 22.22 s (large), and the slowdown is of 15% when the windows are doubling size within optimal conditions and an additional 115% for the non-compliant case. For mammary, the large sizes perform slightly better than the small ones, and the non-compliant overhead (medium size) reaches the top: 531%.

7.5 CPU-GPU Comparison

The central row in Table 8 reports the average speed-up factors on the GPU when helping to compute the FFT-based cross-correlation using CUDA. Gains are unstable for the non-compliant cases, and the most realistic results are the small and large cases where window sizes strictly follow the guidelines provided by the FFTW and CUFFT libraries. For the placenta image set, small windows produce a three times acceleration factor and large windows extend gains to reach 4.11×. For the mammary image set, those gains are more modest: 2.00× and 2.59×, respectively.

Figure 15 demonstrates that the improvement factor on the GPU depends much more on the input image when using mammary rather than placenta, where numbers are more consistent. Additionally, gains are more volatile when increasing the window sizes. This is because the image contents become more heterogeneous on a larger search, showing also higher disparities among images. This effect is corroborated in Fig. 12.

7.6 Parallelism and Scalability on the GPU

The GPU has gained popularity as an outstanding scalable architecture over the past decade, being able to succeed in its goal of sustaining performance doublings every 6 months. In addition to this intra-chip trend, other initiatives like SLI from Nvidia and Crossfire from ATI have emerged to explore inter-chip parallelism (SMP - Symmetric Multi-Processing). The initiative has achieved a remarkable success within the video-game industry, but so far has not been explored for general-purpose computing to our knowledge.

This section evaluates the performance of our registration algorithm on a pair of GPUs when applying SMP parallelism. Our programming techniques are straightforwardly extensible to higher number of graphics cards, and the methods we have used for partitioning the problem guarantees excellent scalability beyond that point. Nevertheless, in this ambitious project we have to warn against the critical role assumed by the input/output system: Dozens or even hundreds of GPUs working in parallel can find an easy way of distributing different search windows efficiently when working on large-scale input images, but there must be a high-performance file system able to read the image tiles in parallel at a sustainable bandwidth high enough to provide data to be processed over the Teraflop rate. During our experiments, we didn’t investigate this bottleneck on a larger number of GPUs. Table 7 quantifies in its last two columns the execution time (including input/output) and the computational time (excluding I/O) to reveal that I/O is responsible of 10–20% of the total execution time. This time has not been included in our subsequent analysis since it is the same for both the CPU and the GPU-optimized versions of our registration algorithm, and I/O is out of the scope of this work. This implicitly assumes that image data are available in DRAM memory or that they can be retrieved efficiently from file using either a parallel file system or a RAID system.

Once data reaches the CPU, there are two basic ways of distributing the workload among multiple GPUs in our registration algorithm: BLOCK or CYCLIC. For the particular case of a pair of GPUs (but without losing any generality), BLOCK assigns the upper half of an image to the first GPU and its lower half to the second GPU. CYCLIC, on the contrary, numbers image tiles and assign even tiles to the first GPU and odd tiles to the second GPU. Because interesting image features tend to be spatially concentrated, BLOCK presents higher potential risk for an unbalanced data partitioning, so we have selected CYCLIC during our experiments.

Our parallelization method works the following way: We create a thread for each image region (tile) which computes the variance on a given CPU to assess whether it is worth computing. If the tile passes this test, it is sent to a predetermined GPU to compute the NCC and search for features. Table 9 outlines the number of tiles processed and discarded on each GPU depending on the input image used from the mammary data set. Workload unbalances range from 2.76% on image 2 to 13.33% on image 4, always growing for lower number of tiles to process (sparsity rate of the input image).

Finally, Fig. 16 shows that gains produced when enabling a second GPU are very diverse, starting with 30–50% on small window sizes, continuing with 60% on large window sizes and ending with an optimal scalability (100% gain) on medium sizes. Those gains are proportional to the computational workload, showing that the GPU is a more scalable processor when it can exploit its arithmetic intensity. In other words, GFLOPS are not limited by data shortages coming from insufficient bandwidth between the video memory and the GPU.

7.7 Summary and Conclusions

Several conclusions can be drawn from our experimental analysis:

1. The placenta image set shows higher speed-up factors on the graphics platform. This is because the images have a larger portion of meaningful content, leading to a denser workload which exploits its arithmetic intensity and memory bandwidth better. Also, a lower number of features processed means higher presence of conditionals in the code, one of the most harmful instructions for GPU performance.
2. The placenta image set is more scalable on multiple GPUs, and gains are more stable among different window sizes. The higher sparsity of the mammary images plays a negative role in the workload distribution, introducing unbalances and preventing parallelism from being fully exploited.

Overall, the GPU achieves a 3–4 speed-up factor in the most typical scenarios (boxed slots in Table 8) versus the CPU, and a pair of GPUs show a satisfactory scalability but unstable gains under different image sets and window sizes.

8 Related Work

Large scale image registration has many applications in both biomedical research [10, 22, 32] and geophysics [33]. However, there are currently few works addressing image

registration algorithms intended to run efficiently on high performance computing environments.

The work on parallel image registration on multicomputers is limited [22] and is restricted to either large computer clusters [34–36] or IBM cell clusters [37]. Clusters of GPUs have been used to implement other heavy workload tasks [38], mostly within the simulation and visualization fields. For example, numerical methods for finite element computations used in 3D interactive simulations [39], and nuclear, gas dispersion and heat shimmering simulations [40].

On the other hand, commodity graphics hardware has become a cost-effective parallel platform to implement biomedical applications in general [23]. Applications similar to ours such as the registration of small radiological images [41] and computation of joint histogram for image registration [42], and others within the fields of data mining [43], image segmentation and clustering [44] have applied commodity graphics hardware solutions. Those efforts have reported performance gains of more than ten times [24] but were mostly implemented using shaders with the Cg language [24].

The present work enhances the graphics implementation through CUDA [27] which exploits parallelism to a wide variety of layers. Our combined implementation of CPU and GPU on a bi-processor platform is one step ahead in performance and provides the first parallel processing solution on large microscopic images for users without requiring an expensive multiprocessor.

In 2004 it was reported that on real numbers, the MxM product may run slower on the GPU due to the lack of high bandwidth access to cached data [45]. The same set of operations that we describe for the correlation phase (two direct FFTs, point-wise multiplication in frequency space, and a inverse FFT) took 0.625 s on a 2003 Intel Xeon CPU for a 1024 × 1024 matrix, versus 2.7 s on a counterpart GeForce 5 GPU [46]. We reverse this situation in 2008 for two major reasons:

1. On the software side, the CUDA programming model makes explicit the use of shared memory, which overcomes the lack of high bandwidth access to closer data.
2. On the hardware side, we exploit the higher scalability of the GPU, doubling performance every 6 months during the present decade versus the 18 month period that takes the CPU that achievement [29].

GPUs for general-purpose computations are an emerging field evolving quickly within computer architecture. Tesla [47] is the latest and more powerful contribution from Nvidia to this area, offering multiple GPUs without video connectors into either a board or a desk-side box to reach near supercomputer levels of single-precision floating-point operations at a cost starting around $1500 (a price similar to the Quadro FX 5600 we have used during our experiments). At a lower price range, there have been recent announcements on double precision graphics architectures from Nvidia (GeForce 9 Series) and ATI (FireStream - see [48]) to provide a definitive solution to software requiring high-precision arithmetic in floating-point operations.

9 Conclusions

With the advances in imaging hardware, tasks like the nonrigid registration of large images with billions of pixels become increasingly popular, evolving towards computationally demanding algorithms for which parallel and scalable solutions become essential. Within this scope, the contribution of our work is twofold:

- First, we provide a parallelizable method which has been successfully applied to biomedical studies for reconstructing the 3-D structures of biological specimens with micron resolution. While the algorithm is motivated by biomedical applications, the principle of using high-level region features for rigid registration and using uniform sampling for nonrigid feature matching are ubiquitous for other applications.
- Second, we have developed a computational framework to expedite the execution of our method on graphics processors. A solid heterogeneous and cooperative multiprocessor platform is established using an AMD Opteron CPU and a pair of Nvidia Quadro GPUs, where the best features of each processor are fully exploited for applying higher degree of parallelism at a variety of levels: Multitask for simultaneous executions of CPU and GPU codes, SMP (Symmetric MultiProcessing) for multicard GPUs using pthreads, and SIMD (Simple Instruction Multiple Data) for the 128 stream processors of the GPU using CUDA.

The CUDA programming model exploits all the capabilities of the GPU as a massively parallel coprocessor to achieve a remarkable speed-up factor as opposed to an expensive supercomputer. Experimental numbers show the success of our techniques, first by decreasing the execution time a 2–4× factor on a single GPU and later extending those gains to a pair of GPUs. For our genetic studies of a mouse placenta sample composed of 500 slides of $16K \times 16K$ pixels each, it takes more than 12 h for our C++ code to accomplish the registration process. This was reduced to less than 2 h using two GPUs, and in addition, we demonstrate promising scalability for extending those gains easily on a large number of GPUs.

Overall, our study provides an illustrative example for how a graphics architecture in conjunction with its CUDA programming model may assist non-computer scientists by adapting grand-challenge biomedical applications to provide almost real-time response to pathologists in computer-aided methods.

10 Future Work

GPUs are highly scalable and evolve towards general-purpose architectures [28], and we envision biomedical image processing as one of the most exciting fields to benefit from them. The present work shows that it is possible to reduce by almost an order of magnitude the execution time on a grand-challenge application without requiring a multicomputer, but the final total execution time still remains around 100 min.

We feel that, even more remarkable than reducing the execution time with our methods, was finding a scalable solution with great future potential. Recently, the Ohio Supercomputing

Center has acquired a visualization cluster, BALE [49], composed of a total of 70 nodes similar to the one we used in our experiments (see Fig. 9). This encouraged us to extend our techniques to a larger number of GPUs, and with the implementation phase already underway, we anticipate preliminary results showing additional performance gain of an order of magnitude running on sixteen nodes. This aims for a prediction of an execution time lower than 3 min on BALE for our registration process over the entire set of 500 slides taken from a placenta sample, which is much closer to our final goal of a real-time response.

In addition, in this work we have not addressed the problem of parallel image transformation. Since GPU is designed to carry out image transformations, our second step forward will be to develop a parallel image transformation scheme on the GPU where different types of transformations like piecewise affine and polynomial may be tested.

Finally, in addition to image registration and 3D reconstruction, we will also explore the applications of GPU computing on other biomedical imaging problems, including blind deconvolution of microscopic images, image denoising, segmentation, and compression.

Acknowledgments

This work was partially supported by the Ministry of Education of Spain (TIC2003-06623, PR-2007-0014), Junta de Andalucía of Spain (P06-TIC-02109), US NIH grant R01 DC06458-01A1 and the startup fund from the Department of Biomedical Informatics at the Ohio State University, US.

We thank Dr. Gustavo Leone from the Ohio State University Cancer Center for providing us the images from mouse placenta and mouse mammary gland we used during the experiments outlined in this paper. We also thank Dr. Dennis Sessanna and Dr. Donald Stredney from the Ohio Supercomputing Center for providing us access to the BALE visualization cluster where most of our execution times were obtained.

References

1. Levinthal C, Ware R. Three-dimensional reconstruction from serial sections. Nature. 1972; 236:207–210.
2. Capowski J. Computer-aided reconstruction of neuron trees from several sections. Computational Biomedical Research. 1977; 10(6):617–629.
3. Johnson E, Capowski J. A system for the three-dimensional reconstruction of biological structures. Computational Biomedical Research. 1983; 16(1):79–87.
4. Huijismans D, Lamers W, Los J, Strackee J. Toward computerized morphometric facilities: A review of 58 software packages for computer-aided three-dimensional reconstruction, quantification, and picture generation from parallel serial sections. The Anatomical Record. 1986; 216(4):449–470. [PubMed: 3541684]
5. Moss V. The computation of 3-dimensional morphology from serial sections. European Journal of Cell Biology. 1989; 48:61–64.
6. Brandt R, Rohlfing T, Rybak J, Krofczik S, Maye A, Westerhoff M, et al. A three-dimensional average-shape atlas of the honeybee brain and its applications. The Journal of Comparative Neurology. 2005; 492(1):1–19. [PubMed: 16175557]
7. Hajnal, J.; Derek, H.; Hawkes, D. Medical image registration. Boca Raton: CRC; 2001.
8. Goshtasby, A. 2-D and 3-D image registration: For medical, remote sensing, and industrial applications. New York: Wiley-Interscience; 2005.
9. Streicher J, Markus D, Bernhard S, Sporle R, Schughart K, Muller G. Computer-based three-dimensional visualization of developmental gene expression. Nature Genetics. 2000; 25:147–152. [PubMed: 10835627]

10. Braumann U, Kuska J, Einenkel J, Horn L, Luffler M, Huckel M. Three-dimensional reconstruction and quantification of cervical carcinoma invasion fronts from histological serial sections. IEEE Transactions on Medical Imaging. 2005; 24(10):1286–1307. [PubMed: 16229416]
11. Crum W, Hartkens T, Hill D. Non-rigid image registration: Theory and practice. The British Journal of Radiology. 2004; 77:S140–S153. [PubMed: 15677356]
12. Hill W, Baldock R. The constrained distance transform: Interactive atlas registration with large deformations through constrained distance. Proceedings of the workshop on image registration in deformable environments. 2003
13. Yoo, T. Insight into images: Principles and practice for segmentation, registration, and image analysis. AK, Peters; 2004.
14. Sarma S, Kerwin J, Puelles L, Scott M, Strachan T, Feng G, et al. 3d modelling, gene expression mapping and post-mapping image analysis in the developing human brain. Brain Research Bulletin. 2005; 66(4–6):449–453. [PubMed: 16144630]
15. Jenett A, Schindelin J, Heisenberg M. The virtual insect brain protocol: Creating and comparing standardized neuroanatomy. BMC Bioinformatics. 2006; 7:544. [PubMed: 17196102]
16. Wenzel P, Wu L, Sharp R, de Bruin A, Chong J, Chen W, et al. Rb is critical in a mammalian tissue stem cell population. Genes & Development. 2007; 21(1):85–97. [PubMed: 17210791]
17. Cooper, L.; Huang, K.; Sharma, A.; Mosaliganti, K.; Pan, T. Registration vs. reconstruction: building 3-d models from 2-d microscopy images. Proceedings of the workshop on multiscale biological imaging, data mining and informatics; 2006. p. 57-58.
18. Huang, K.; Cooper, L.; Sharma, A.; Pan, T. Fast automatic registration algorithm for large microscopy images. Proceedings of the IEEENLM life science systems & applications workshop; 2006. p. 1-2.
19. Koshevoy, P.; Tasdizen, T.; Whitaker, R. Technical Report UUSCI-2006-018. University of Utah, SCI Institute; 2006. Implementation of an automatic slice-to-slice registration tool. (Online) Available: http://www.sci.utah.edu/publications/SCITechReports/UUSCI-2006-018.pdf
20. Prescott, J.; Clary, M.; Wiet, G.; Pan, T.; Huang, K. Automatic registration of large set of microscopic images using high-level. Proceedings of the IEEE international symposium on medical imaging; 2006. p. 1284-1287.
21. Mosaliganti, R.; Pan, T.; Sharp, R.; Ridgway, R.; Iyengar, S.; Gulacy, A., et al. Registration and 3d visualization of large microscopy images. Proceedings of the SPIE medical imaging meeting; 2006. p. 923-934.
22. Schmitt O, Modersitzki J, Heldmann S, Wirtz S, Fischer B. Image registration of sectioned brains. International Journal of Computer Vision. 2007; 73(1):5–39.
23. Botnen, M.; Ueland, H. Tech Rep. Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology; 2004. The GPU as a computational resource in medical image processing.
24. Owens JD, Luebke D, Govindaraju N, Harris M, Kruger J, Lefohn AE, et al. A survey of general-purpose computation on graphics hardware. Journal of Computer Graphics Forum. 2007; 26:21–51.
25. Sharp R, Ridgway R, Mosaliganti K, Wenzel P, Pan T, de Bruin A, et al. Volume rendering phenotype differences in mouse placenta microscopy data. Computing in Science & Engineering. 2007; 9(1):38–47. (January/February).
26. Lewis, JP. Fast normalized cross-correlation; *Vision interface*. Canadian image processing and pattern recognition society. 1995. p. 120-123.(Online) Available: citeseer.ist.psu.edu/lewis95fast.html
27. Compute Unified Device Architecture (CUDA). [Accessed 1 May 2008.] Home page maintained by Nvidia. 2007. http://developer.nvidia.com/object/cuda.html
28. GPGPU. A web site dedicated to the general-purpose on the GPU. 2007. http://www.gpgpu.org
29. Fatica M, Luebke D, Buck I, Owens D, Harris M, Stone J, et al. Cuda tutorial at supercomputing. 2007 Nov.
30. CUFFT library. [Accessed 28 Dec 2007.] Home page maintained by nvidia. 2007. http://developer.download.nvidia.com/compute/cuda/1_1/CUFFT_Library_1.1.pdf
31. The FFTW library. [Accessed 1 May 2008.] FFTW home page. 2007. http://www.fftw.org

32. Cooper, L.; Naidu, S.; Leone, G.; Saltz, J.; Huang, K. Registering high resolution microscopic images with different histochemical stainings - a tool for mapping gene expression with cellular structures. Proceedings of the workshop on microscopic image analysis with applications in biomedicine; 2007.
33. Kim T, Im YJ. Automatic satellite image registration by combination of matching and random sample consensus. IEEE Transactions on Geoscience and Remote Sensing. 2003; 41(5):1111–1117.
34. Ino F, Ooyama K, Hagihara K. A data distributed parallel algorithm for nonrigid image registration. Parallel Computing. 2005; 31:19–43.
35. Warfield S, Jolesz F, Kikinis R. A high performance computing approach to the registration of medical imaging data. Parallel Computing. 1998; 24:1345–1368.
36. Rohlfing T, Maurer C. Nonrigid image registration in shared-memory multiprocessor environments with applications to brains, breasts, and bees. IEEE Transactions on Information Technology in Biomedicine. 1998; 7(1):16–25. [PubMed: 12670015]
37. Ohara, M.; Yeo, H.; Savino, F.; Iyengar, G.; Gong, L.; Inoue, H., et al. Real time mutual information-based linear registration on the cell broadband engine processor. Proceedings of the IEEE international symposium on medical imaging (ISBI); 2007. p. 33-36.
38. Fan, Z.; Qiu, F.; Kaufman, A.; Yoakum-Stover, S. GPU cluster for high performance computing. Proceedings 2004 ACM/IEEE intl. conference for high performance computing, networking, storage and analysis; Washington DC, USA. 2006. p. 47-53.
39. Wu W, Heng P. A hybrid condensed finite element model with GPU acceleration for interactive 3d soft tissue cutting: Research articles. Computer Animation and Virtual Worlds. 2004; 15(3–4): 219–227.
40. Zhao Y, Han Y, Fan Z, Qiu F, Kuo Y, Kaufman A, et al. Visual simulation of heat shimmering and mirage. IEEE Trans on Visualization and Computer Graphics. 2007; 13(1):179–189.
41. Ino, F.; Gomita, J.; Kawasaki, Y.; Hagihara, K. A GPGPU approach for accelerating 2-d/3-d rigid registration of medical images. Proceedings of the 4th international symposium on parallel and distributed processing and applications (ISPA); Berlin: Springer; 2006. p. 769-780.Lecture Notes in Computer Science 4331
42. Hastreiter P, Rezk-Salama C, Nimsky C, Lurig C, Greiner G. Techniques for the analysis of the brain shift in neurosurgery. Computers & Graphics. 2000; 24(3):385–389.
43. Guha, S.; Krisnan, S.; Venkatasubramanian, S. Data visualization and mining using the GPU. tutorial at 11th ACM international conference on knowledge discovery and data mining (KDD 2005); 2005.
44. Hadwiger, M.; Langer, C.; Scharsach, H.; Buhler, K. Tech Rep. VRVis Research Center; Vienna, Austria: 2004. State of the art report on GPU-based segmentation. TR-VRVIS-2004-17
45. Fatahalian, K.; Sugerman, J.; Hanrahan, P. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. Proceedings of the ACM SIGGRAPH -EUROGRAPHICS workshop on graphics hardware (HWWS'04); Grenoble, France. August; 2004.
46. Moreland, K.; Angel, E. The FFTW on a GPU. Proceedings of the ACM SIGGRAPH - EUROGRAPHICS workshop on graphics hardware (HWWS'03); San Diego, California USA. August; 2004.
47. TESLA. [Accessed 1 Jan 2008.] GPGPU high-end hardware solutions from Nvidia. 2008. http://www.nvidia.com/object/tesla_computing_solutions.html
48. FireStream. [Accessed 1 Jan 2008.] GPU hardware solutions from AMD/ATI. 2008. http://ati.amd.com/products/streamprocessor/specs.html
49. The BALE Supercomputer. the Ohio Supercomputer Center (OSC); http://www.osc.edu/supercomputing/hardware

Biographies

Antonio Ruiz received M.S. degree in Telecommunication Engineering from the University of Malaga in 2006. During 2007 he was a Visiting Scholar in the Biomedical Informatics Department at the Ohio State University, USA. Currently he is pursuing his PhD degree at the Department of Computer Architecture, University of Malaga, Spain.

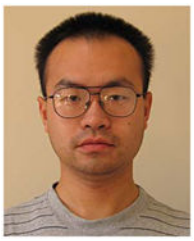
Over the past couple of years, he has published more than ten papers in international journals and conferences. His research interests include image processing on biomedical applications using graphics hardware as well as optimizing and automating data mining algorithms.

Manuel Ujaldon received his B.S. degree in Computer Science from the Univ. of Granada (Spain, 1991) and his M.S. and Ph.D. degrees in Computer Science from the Univ. of Malaga (Spain, 1993 and 1996). During 1994 and 1995 he was a Research Assistant in the Computer Architecture Department at the Univ. of Malaga, where he became Assistant Professor in 1996 and finally Associate Professor in 1998. From 2000 to 2004 he was also vicehead of the School of Computer Engineering at the Univ. of Malaga, as well as Director of its International Programs.

Dr. Ujaldon spent one year as invited researcher at the Institute for Software Technology and Parallel Systems (Univ. of Vienna - Austria) and was also a predoctoral and postdoctoral researcher at the Computer Science Department of the Univ. of Maryland (USA, 1994, 1996/97) and Biomedical Informatics Department of the Ohio State Univ. (USA, 2003-08), where he currently holds an adjunct appointment as visiting professor.

He has published eight books on computer architecture (in Spanish) and more than 40 papers in international peer-reviewed journals and conferences. His research interest includes streaming architectures as well as compiler and software development for running general-purpose scientific applications on graphics processors.

Lee Cooper received the B.S. (2003) and M.S. (2006) degrees in electrical and computer engineering from The Ohio State University, Columbus, where he is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering. His research interests include machine learning, image informatics for biomedical applications, and large scale image analysis.



Kun Huang received two B.S. degrees in biology and computer science from Tsinghua University, Beijing, China, in 1996, the M.S. degree in physiology in 1998, the M.S. degree in electrical engineering in 2000, the M.S. degree in mathematics in 2002, and the Ph.D. degree in electrical and computer engineering in 2004, all from the University of Illinois at Urbana-Champaign, Urbana.

Currently he is an Assistant Professor with the Department of Biomedical Informatics, The Ohio State University (OSU), Columbus, Ohio, USA and the Assistant Director of the OSU Comprehensive Cancer Center Biomedical Informatics Shared Resources. His research interests include biomedical imaging analysis, bioinformatics, computer vision, and machine learning.

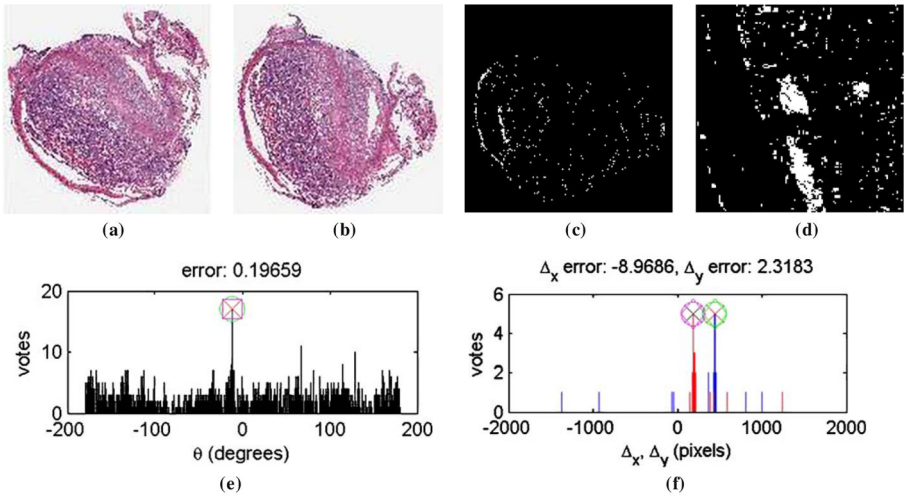


Figure 1.

Fast rigid registration using high-level features. **a**, **b** 5× downsampled placenta images (originally 20×), approximately 4000 × 4000 pixels. **c** Extracted regions of pixels corresponding to blood regions for the image in **b**. Only 50–70 such regions are selected for the matching process. **d** Closeup of result in **c. e** Voting histogram for rotation angle. The difference between the voting result and a manual registration is less than 0.2°. **f** Histogram for the translation components *x* and *y*. The differences between the voting results and a manual registration are each less than 10 pixels.

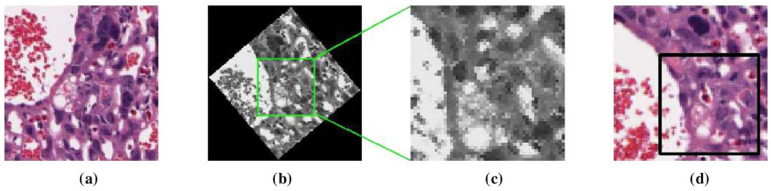


Figure 2.

The process of feature matching. **a** First image. **b** First image rotated a given angle provided from the rigid registration. **c** Local sanity region selected within first image ($W_1 \times W_1$-pixel template patch or feature window). **d** Search region within second image ($W_2 \times W_2$-pixel search window).

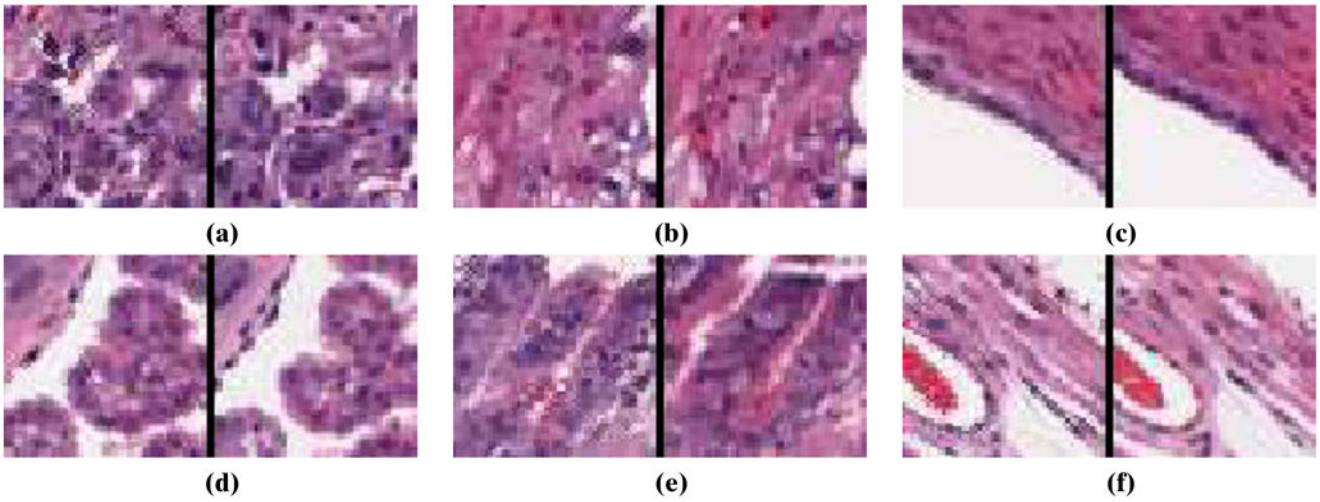


Figure 3.

a–**f** Randomly selected matched points with the image patches around them between the two images.

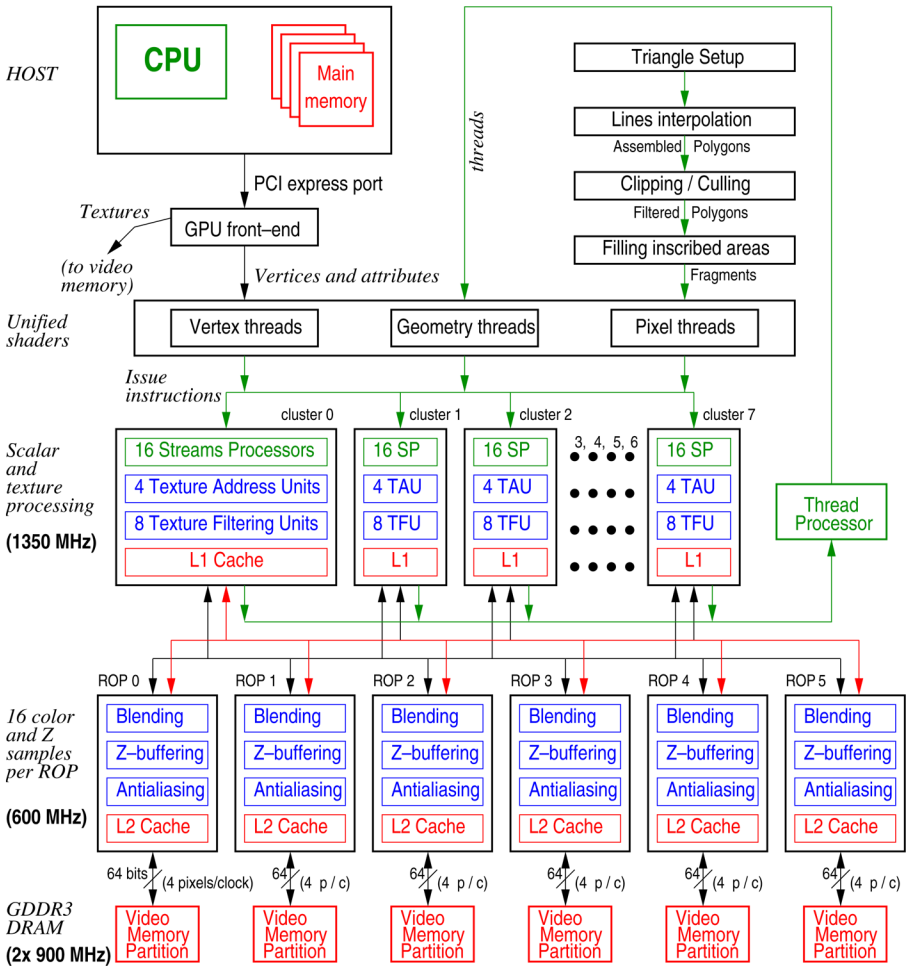


Figure 4.

The block diagram of the Nvidia G80 architecture, the GPU we used during our experiments. The program, decomposed in threads, is executed on 128 streams processors (*central row*). The data are stored on L1 caches, L2 caches and video memory (*lower rows*).

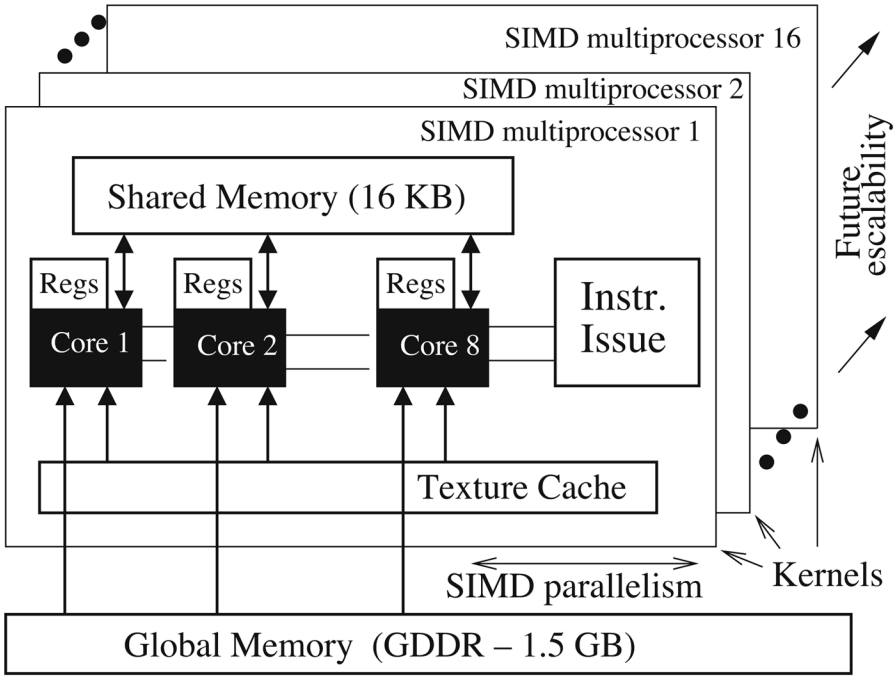


Figure 5.
The CUDA hardware interface for the GPU.

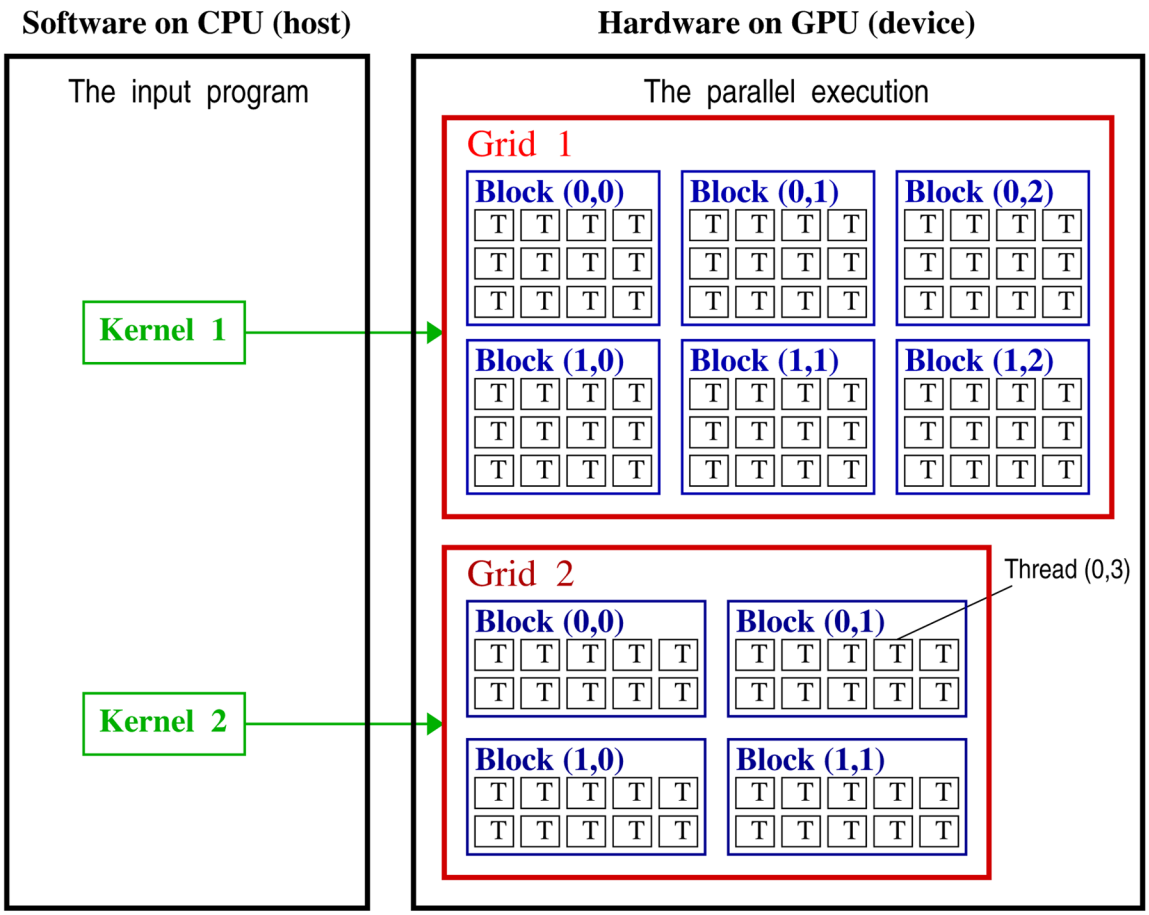


Figure 6.

The CUDA programming model. In this example, a program is decomposed into two kernels, each implemented through a grid, with the first grid composed of 2 × 3 blocks, each containing 3 × 4 threads executed in a SIMD fashion.

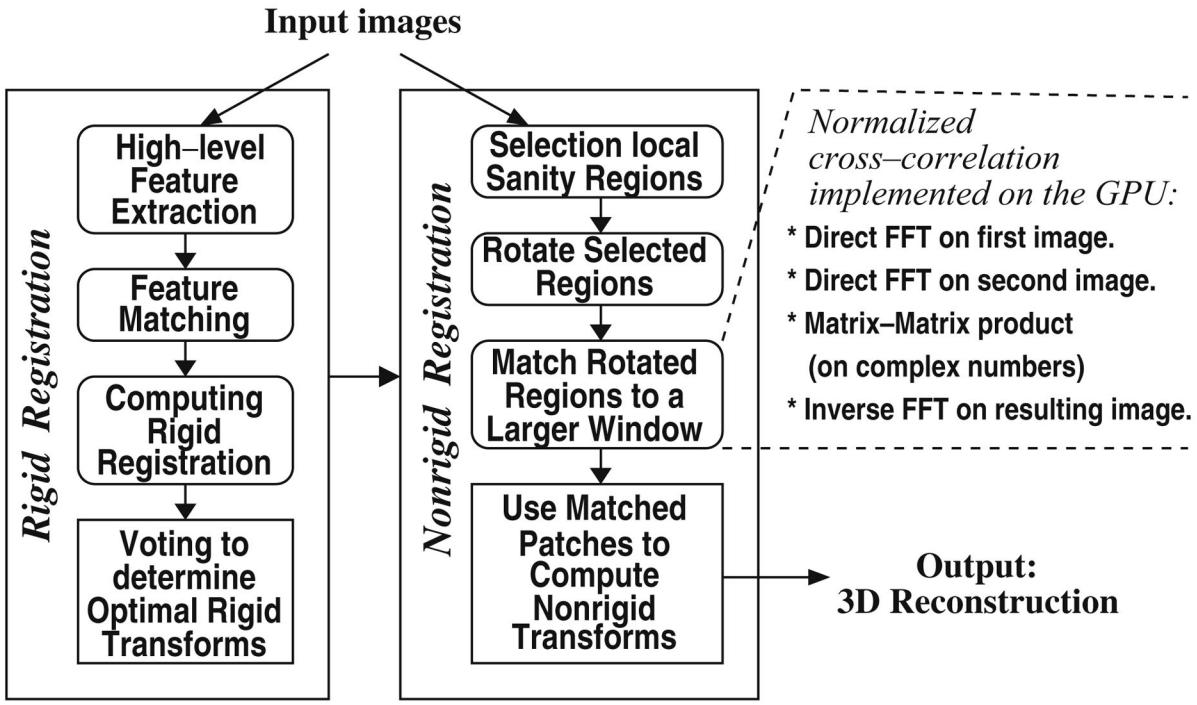


Figure 7.

The workflow of our image registration algorithm, which consists of two stages: rigid registration and nonrigid registration. Rounded boxes are independent local operations that can be straightforwardly carried out in parallel. The most computationally demanding phase is selected to run on the GPU for a much faster execution.

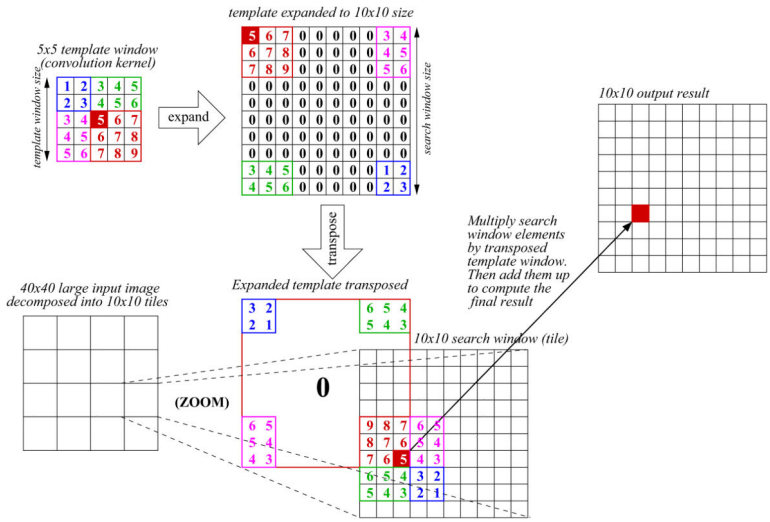


Figure 8.

The computation of FFT-based NCC. The template window has to be expanded to the search window size and convolution with the expanded kernel is equivalent to the one with the initial kernel. The example is shown for a large image having 40 × 40 pixels and decomposed into 4 × 4 tiles, thus resulting a search window of 10 × 10 pixels. The template window has 5 × 5 pixels, half of the search window on each dimension as in our registration algorithm.

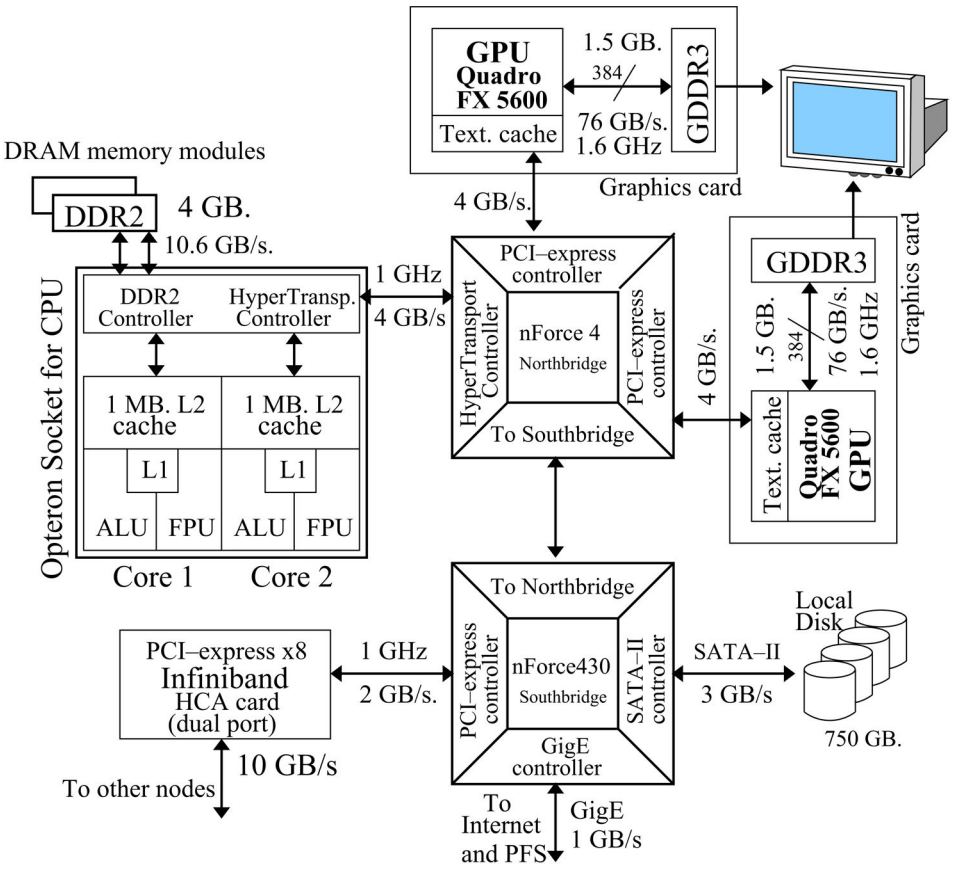


Figure 9.
The block diagram for our computing node, which integrates one CPU and two GPUs.

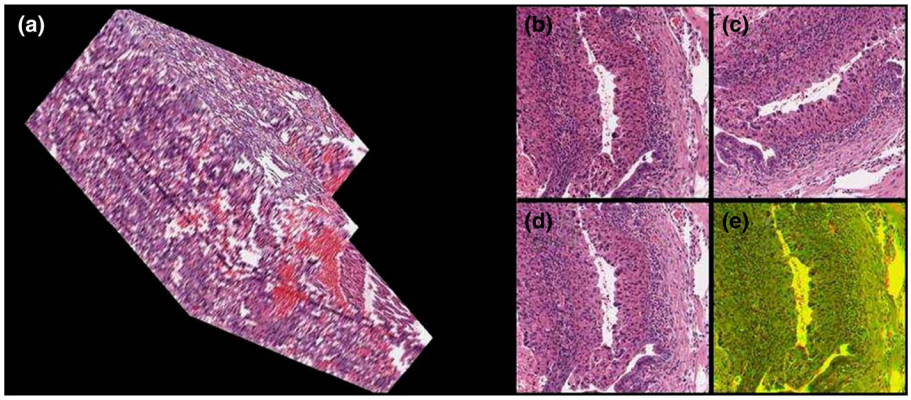


Figure 10.

a An example of 3D reconstruction of the mouse placenta. Since the images are large, we only show a small fraction of the reconstructed 3D model from 30 sections. **b**–**e** Registration of mouse mammary gland images: **b** a 1000 × 1000-pixel patch from the reference image; **c** the corresponding 1000 × 1000-pixel patch from the floating image; **d** the image patch from the floating image after the nonrigid transformation; **e** overlay of the two images.

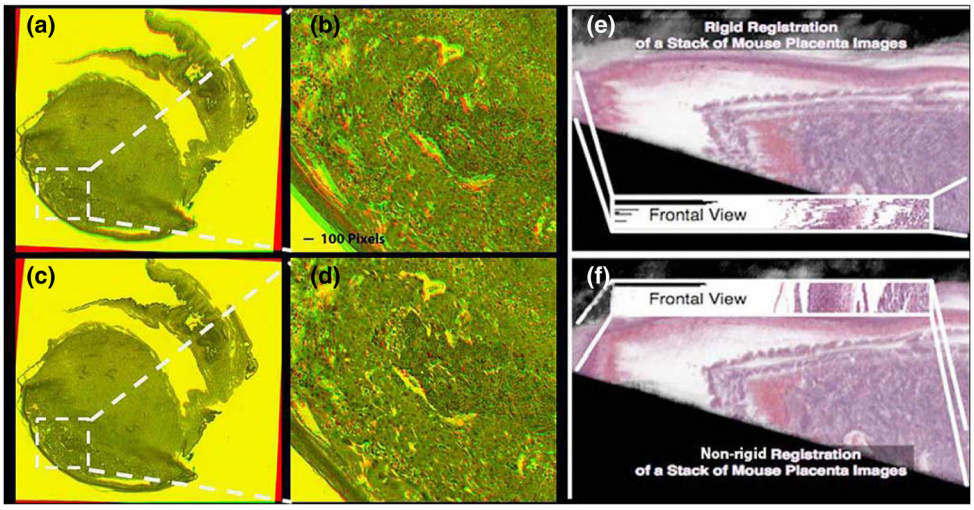
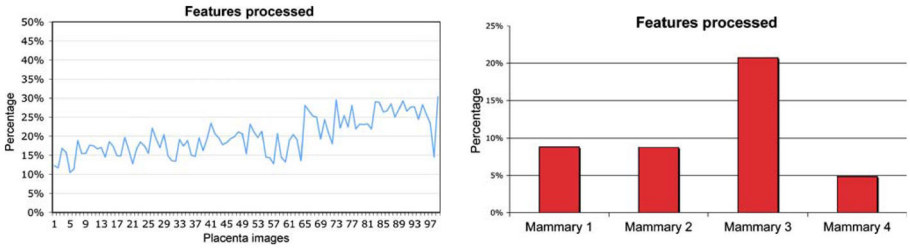


Figure 11.

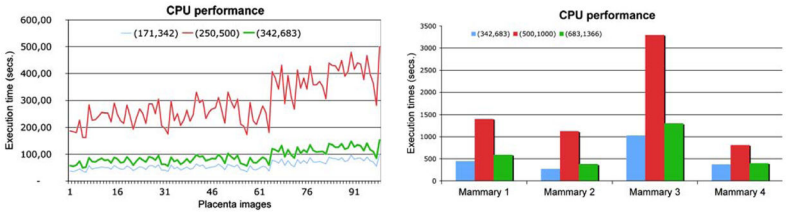
a Overlay of two placenta images after the rigid registration. The two images are transformed into gray scale. The region that is not well aligned shows a blurred boundary; **b** a high resolution patch from the panel **a; c** Overlay of two placenta images after the nonrigid registration; **d** a high resolution patch from the panel **c; e**–**f** the registered images are stacked and rendered using volumetric rendering. The frontal views are the virtual cross sections generated from the 3D image stacks.



(a) Placenta image set. **(b)** Mammary image set.

Figure 12.

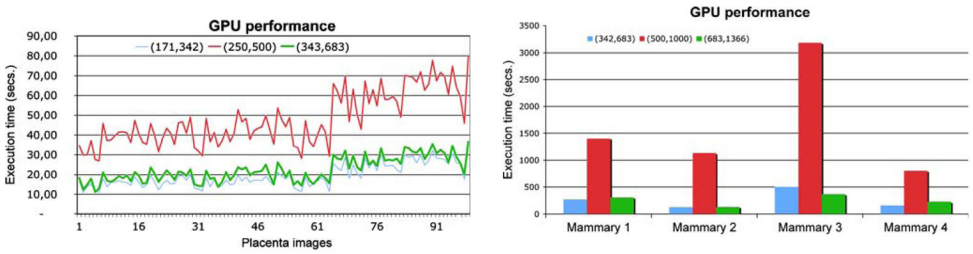
Percentage of features processed per image on each input image set. We take the small size for the template and search window as the most representative.



(a) Placenta image set. **(b)** Mammary image set.

Figure 13.

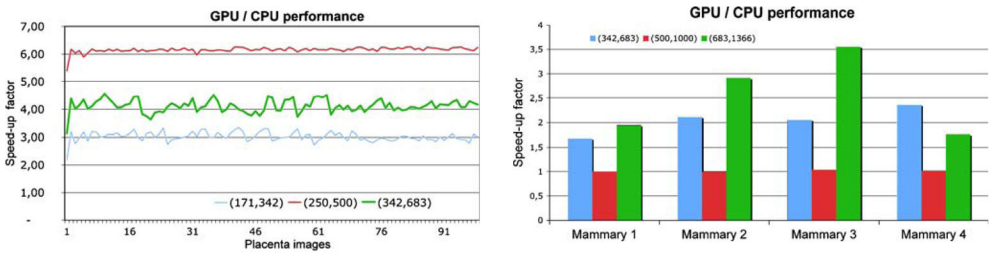
Execution times on the CPU Opteron for the registration algorithm on a pair of images under different image sets and window sizes. The first pair of numbers on chart legends corresponds to the small window sizes (template and search window, respectively), then medium and finally large sizes. **a** For placenta, average times are 57.97 s (small), 294.57 s (medium) and 91.33 s (large). **b** For mammary, average times are 530.41 s (small windows), 1660.91 s (medium) and 669.96 s (large).



(a) Placenta image set. **(b)** Mammary image set.

Figure 14.

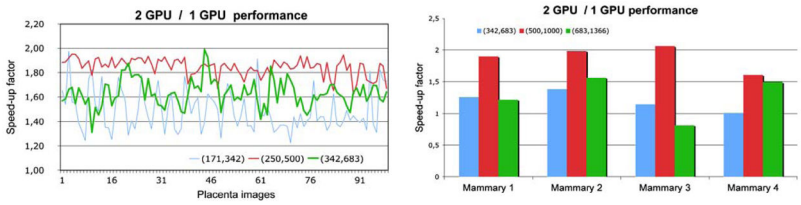
Execution times on the GPU Quadro for the registration algorithm on a pair of images under different image sets and window sizes. The first pair of numbers on chart legends corresponds to the small window sizes (template and search window, respectively), then medium and finally large sizes. **a** For placenta, average times are 19.27 s (small), 47.80 s (medium) and 22.22 s (large). **b** For mammary, average times are 264.09 s (small windows), 1629.72 s (medium) and 257.95 s (large).



(a) Placenta image set. **(b)** Mammary image set.

Figure 15.

Comparison between the GPU and CPU execution time in terms of GPU speed-up factor. When the window sizes increase, times are more irregular in **b**. The first pair of numbers on chart legends corresponds to the small window sizes (template and search window, respectively), then medium and finally large sizes. For placenta, the average speed-up is 3.00× (small), 6.16× (medium) and 4.11× (large). For mammary, the average speedup is 2.00× (small windows), 1.01× (medium) and 2.59× (large).



(a) Placenta image set. **(b)** Mammary image set.

Figure 16.

GPU scalability. Improvement factor when enabling a second GPU. The first pair of numbers on chart legends corresponds to the small window sizes (template and search window, respectively), then medium and finally large sizes. **a** For placenta, the average speed-up is 1.46× (small), 1.83× (medium) and 1.62× (large). **b** For mammary, the average speed-up is 1.17× (small windows), 1.94× (medium) and 1.09× (large).

Table 1

Major limitations for the CUDA programming model on the Nvidia G80 GPU used during our experimental study. The last column assesses its importance according to the impact on the programmer's job and overall performance.

Parameter	Limit	Impact
Multiprocessors per GPU	16	Low
Processors/multiprocessor	8	Low
Threads/warp	32	Low
Thread blocks/multiprocessor	8	Medium
Threads/block	512	Medium
Threads/multiprocessor	768	High
32-bit registers/multiprocessor	8192	High
Shared memory/multiprocessor	16 KB	High

Table 2

Constraints in memory addressing (first five rows) and maximum performance (last two rows) reached by the CUDA programming model in its latest version (1.1, as of May 2008).

Parameter	Value
Constant memory/multiprocessor	64 KB.
Maximum sizes of each dimension of a block	512 × 12 × 64
Maximum sizes of each dimension of a grid	64K×64K×1
CUDA maximum memory pitch	256 KB.
CUDA texture alignment	256 bytes
Geometrical performance	$3*10^8$ triangles/sc.
Fill-rate (textural performance)	$192*10^8$ texels/sc.

Table 3

Percentage weight on average for each of the computational stages before and after porting to the GPU.

Computational phase	CPU (%)	GPU (%)
Two forward FFTs	68	74
Point-wise multiplication	3	2
One inverse FFT	29	24

Table 4

Different sizes used for the template (feature) and search windows in our registration algorithm (in pixels).

Input image:	**Placenta: 16K × 16K**			**Mammary: 23K × 62K**		
Window size:	**Small**	**Medium**	**Large**	**Small**	**Medium**	**Large**
Template window (in pixels)	171	250	342	342	500	683
Search window (in pixels)	342	500	683	683	1000	1366
Aggregate (template+search-1)	512	749	1024	1024	1499	2048
CPU friendly (FFTW library)	Yes	No	Yes	Yes	No	Yes
GPU friendly (CUFFT library)	Yes	(*)	Yes	Yes	No	Yes

We also include an evaluation about whether those sizes contribute to perform further optimizations in the corresponding CPU and GPU codes, considering the libraries we use during the implementation: FFTW on the CPU and CUFFT on the GPU.

(*) This slot is partially in favour of the GPU because 749 is a multiple of seven, a small prime number.

Table 5

The set of images used as input data sets for our registration algorithm.

Field of study	Research area and biomedical goals	Mouse source	Computational workload	Image size (pixels)	Number of slides
Genetic	Role of a gene	Placenta	Medium	16K × 16K	100
Oncology	Breast cancer tumor	Mammary	Large	23K × 62K	4

Table 6

Summary of the major features of our high-end GPU from Nvidia.

Feature	Value
GPU	
Model	Quadro FX 5600
Core clock frequency	600 MHz
Stream processors clock	1.35 GHz
Manuf. technology	90 nm
Video memory	
Clock frequency	1.6 GHz
Bus width	384 bits
Bandwidth	76.8 GB/sc
Memory size	1.5 GB

Table 7

Workload breakdown on single CPU for mammary image set.

	Number of features extracted			Workload on CPU (in seconds)	
Window size: (template, search)	**Small (342,683)**	**Medium (500, 1000)**	**Large (683, 1366)**	**Execution time with I/O**	**Execution time without I/O**
Mammary 1	1196	655	384	650.86	558.54 (85%)
Mammary 2	1048	568	312	497.83	414.17 (83%)
Mammary 3	3119	1528	854	1320.01	1192.69 (90%)
Mammary 4	690	322	168	463.77	340.62 (73%)

The number of features extracted for each input image within the mammary data set differs due to content. Execution times in the last two columns represent the large case.

Table 8

Execution times (in seconds) and speed-up factors for the different implementations developed for computing our registration algorithm on a pair of images with maximum performance.

Input image set:	**Placenta: 16K × 16K**			**Mammary: 23K × 62K**		
Window size: (template, search)	**Small (171,342)**	**Medium (250,500)**	**Large (342,683)**	**Small (342,683)**	**Medium (500,1000)**	**Large (683,1366)**
CPU exec. time	57.97	294.57	91.33	530.41	1660.91	669.96
GPU exec. time	19.27	47.80	22.22	264.09	1629.72	257.95
GPU speed-up	3.00×	6.16×	4.11×	2.00×	1.01×	2.59×
2 GPUs time	13.13	26.05	13.66	225.17	837.51	234.62
2 GPU/1 GPU	1.46×	1.83×	1.62×	1.17×	1.94×	1.09×
2 GPU/1 CPU	4.41×	11.30×	6.68×	2.57×	1.98×	2.85×

The average of all 100 and 4 runs is reported for the placenta and mammary image sets. Boxed numbers highlight the GPU speed-up under the most typical scenarios.

Table 9

Number of tiles processed and discarded for each image within the mammary image set on each GPU under the two GPUs parallel execution.

		Number of tiles			
Input image	**Graphics processor**	**Tested**	**Processed/discarded**	**Workload unbalance (%)**	**Execution time (s)**
Mammary 1	GPU 1	1672	196/1476	4.08	260.41
	GPU 2	1672	188/1484		
Mammary 2	GPU 1	1496	158/1338	2.53	101.32
	GPU 2	1496	154/1342		
Mammary 3	GPU 1	1872	428/1444	2.76	522.43
	GPU 2	1911	426/1485		
Mammary 4	GPU 1	1786	78/1708	13.33	225.37
	GPU 2	1786	90/1696		

Workload unbalance and execution time are shown in the last two columns. The search window size here is 684 × 684 pixels.