

Non-Rigid Volume Registration: A CUDA-based GPU implementation of the Morphon

Daniel Forsberg^{*†‡}, Anders Eklund^{*†}, Mats Andersson^{*†} and Hans Knutsson^{*†}

^{*}Department of Biomedical Engineering, Linköping University, Sweden

[†]Center for Medical Image Science and Visualization (CMIV), Linköping University, Sweden

[‡]Sectra Imtec, Linköping, Sweden

Email: {dafor, andek, matsa, knutte}@imt.liu.se

Abstract—Image registration is frequently used within the medical image domain and where methods with high performance are required. The need for high accuracy coupled with high speed is especially important for applications such as adaptive radiation therapy and image-guided surgery. During the last years, a number of significant projects have been introduced to make the computational power of GPUs available to a wider audience. The most well known project is the introduction of CUDA (Compute Unified Device Architecture). In this paper, we present a CUDA based GPU implementation of a non-rigid image registration algorithm, known as the Morphon, and compare it with a CPU implementation of the Morphon. The achieved speedup, in the range of 51-54x, is also compared with speedups reported from other non-rigid registration methods implemented on the GPU. These include the Demons algorithm and a mutual information based algorithm.

I. INTRODUCTION

Image registration is a well known concept which is frequently applied in a number of different areas, for instance geophysics, robotics and medicine. The basic idea of image registration is to find a deformation field d that geometrically transforms one image (source image, I_S) in order to align it with another image (target image, I_T). This can be more strictly defined as an optimization problem, where the aim is to find a deformation field that maximizes the similarity between the source and the target images. Depending on the metric used to compare the images, the problem could also be defined as minimizing the dissimilarity between two images.

A frequently applied categorization of different image registration algorithms is to classify them as either parametric or non-parametric [1]. Parametric methods refers to methods, where a parameterization has been performed to reduce the number of degrees of freedom. This is either achieved by a limitation of the admissible transformations to rigid or affine, or by setting the transformations to be a linear combination of a set of basis functions. A typical example of the latter is spline based methods. Non-parametric methods on the other hand, which include flow or diffusion based methods, independently estimate a displacement vector for each voxel. In this paper, we will work with a registration and segmentation algorithm known as the Morphon, first presented in [2]. In the presented implementation, the Morphon can be considered as a non-parametric method.

The use of image registration within the medical image

domain is vast and image registration is included in a number of different tasks, such as; surgical planning, radiotherapy planning, image-guided surgery and disease progression monitoring. A common need for all these tasks is high accuracy, in terms of the actual registration result, coupled with high performance, in terms of speed of the registration process. Usually, there is a tradeoff between accuracy and speed, i.e. the better the accuracy is the longer it will take to perform the registration. This problem is particularly relevant for adaptive radiation therapy and image-guided surgery, which more or less require real-time registration in order to achieve optimal or even an acceptable performance.

A large number of the existing image registration algorithms can be parallelized in order to improve the performance. However, for a long time solutions have been proposed that either are not practically feasible in a real world scenario (due to financial, practical or availability aspects) or available techniques for dealing with parallelization have been too difficult to master properly (for instance using the normal graphics pipeline for GPU computing). A recent and very cost-effective trend for parallelization is GPGPU computing (General-Purpose computation on Graphics Processing Units), which provides tools for utilizing the computational power available on modern graphics cards. One technique to achieve parallelization on the GPU is CUDA (Common Unified Device Architecture) from NVIDIA [3]. Although there are some other techniques, such as OpenCL from the Khronos Group and DirectCompute from Microsoft, and that they all share some common concepts for parallel computing on the GPU, CUDA is still the most frequently applied. This is likely due to CUDA being a more mature technique [4]. However, this is likely to change as OpenCL and DirectCompute will mature over the coming years. The focus of this paper is on CUDA and therefore we will use CUDA specific terminology when describing GPGPU related terms.

The purpose of this paper is to present a CUDA based GPU implementation of the Morphon, compare its performance with a CPU based implementation and to compare the achieved speedup with the speedups of other CUDA based GPU implementations of non-parametric image registration algorithms.

II. RELATED WORK

In the papers [5], [6] two different CUDA based implementations of the Demons algorithm are presented. Although they differ in hardware, CUDA version and implemented Demons algorithm, they both report similar computation times, 7-11 seconds, for an image volume of the approximate size 256x256x100. In [5] they compare the GPU implementation with two CPU implementations (one single- and one multi-threaded) and achieve a speedup factor of 55x respectively 35x, whereas in [6] they achieve a speedup factor of 40x when compared to their CPU implementation. Another relevant paper presents a CUDA based implementation of a mutual information driven algorithm [7]. They report computation times of about 19 seconds for datasets of the size 256x256x128 along with a speedup factor of 25x when compared to their CPU implementation.

For a more general and recent survey of medical image registration algorithms employing multi-core architectures (including GPUs) the interested reader is referred to [4].

III. CUDA

The basic building blocks of CUDA consist of *kernels* (functions) that are launched by the *host* (CPU) but executed on the *device* (GPU). Each kernel is executed by a number of *threads* in parallel. All kernels are grouped into different *thread blocks* where each thread block is executed on a single *stream multiprocessor*, which consists of a number of *cores*. All thread blocks are arranged into a structure known as a *grid*. The threads, the thread blocks and the grid form the thread hierarchy.

There is also another hierarchy known as the memory hierarchy. First of all, each thread has a *local* (private) memory that is limited, since it is dependent on the number of threads per thread block. Each thread block has also a certain amount of *shared* memory which is accessible to all threads within the same thread block. Then there exists the *global* memory which is accessible to all threads. Alongside these memory types, there are also two read-only memories, which are accessible by all threads, *constant* and *texture* memory.

It is by utilization of the parallel execution of the threads that the computational performance can be improved. However, care must be taken to correctly use and understand the properties of the different hierarchies; if not the performance improvements will not be as significant as expected or even absent.

Data parallelization is commonly referred to as single instruction multiple data (SIMD), but due to the thread hierarchy it is more commonly known as single instruction multiple threads (SIMT) when referring to GPGPU computing. This is because the threads do not need to operate on data in a linear pattern, i.e. threads i , j and k do not need to operate on data i , j and k respectively but can operate on the data in an arbitrary access pattern, e.g. j , i and k .

IV. THE MORPHON

The Morphon is a phase-based algorithm where a source image, $I_S(\mathbf{x})$, is iteratively deformed, $I_D(\mathbf{x}) = I_S(\mathbf{x} + \mathbf{d}(\mathbf{x}))$, until the phase-difference between the target image, I_S , and the deformed image, I_D , has been minimized. This process is performed over multiple scales, starting on coarse scales to register large global displacements and moving on to finer scales to register smaller local deformations. The algorithm itself consists of the following three sub-steps: *local displacement estimation*, *deformation field accumulation*, *deformation*. For a more detailed review of the different sub-steps the user is referred to [2].

V. IMPLEMENTATION

The CPU algorithm was implemented in MATLAB 2009a from MathWorks. With the aid of the built-in profiling tool in MATLAB various inefficient steps could be tracked down and handled. For instance the function `convn` was replaced with the much more efficient `imfilter`.

The GPU algorithm was implemented using CUDA Toolkit 3.0 and where features available in compute capability 1.3 were utilized. This includes the usage of the extended warp size, the extended maximum number of resident threads per multiprocessor and the improved local memory size. For instance, with the improved local memory size, it is possible to perform more computations per kernel, since the intermediate results can be temporarily stored in the local memory instead of the global memory. To handle some of memory bandwidth limitations associated with the global memory, the shared memory was extensively used in the various convolution kernels. Also the GPU implementation benefited from the built-in profiling tool in CUDA, among other things helped in improving the access pattern to the global memory.

VI. RESULTS

The CPU and GPU implementations were executed on an HP Z400 Workstation with an Intel Xenon Quad Core 2.67 GHz processor, 8 GB RAM and Fedora 12 x64. The CUDA implementation was executed on an NVIDIA GTX 285 with 240 cores and 2 GB on-board memory with NVIDIA Driver 195.36.31. As test datasets, three datasets with different sizes (128x128x128, 196x196x196, 256x256x128 referred to as dataset 1, 2 and 3) were used, two synthetic datasets and one MRI dataset. The synthetic datasets consisted of a cross and a manually deformed cross as source and target images, whereas the MRI dataset consisted of two different patients. Since the purpose of the paper is to compare the relative performance improvement, the tests were executed with a set number of scales and iterations per scale to use. To measure timing results in MATLAB, the functions `tic` and `toc` were used, and in CUDA, the functions `cutResetTimer`, `cutStartTimer`, `cutStopTimer` and `cutGetTimerValue` were used.

TABLE I
COMPUTATIONAL TIME FOR THE WHOLE ALGORITHM.

Dataset	1	2	3
GPU runtime (s)	9.45	30.66	38.12
CPU runtime (s)	496.01	1572.39	2059.16
GPU speedup	52.5	51.3	54.0

TABLE II
COMPUTATIONAL TIME FOR LOCAL DISPLACEMENT ESTIMATION.

Dataset	1	2	3
GPU runtime (s)	6.19	20.07	26.35
CPU runtime (s)	364.41	1143.60	1539.67
GPU speedup	58.8	57.0	58.4

TABLE III
COMPUTATIONAL TIME FOR DEFORMATION FIELD ACCUMULATION.

Dataset	1	2	3
GPU runtime (s)	1.92	6.40	7.27
CPU runtime (s)	56.64	176.47	246.17
GPU speedup	29.5	27.6	33.9

TABLE IV
COMPUTATIONAL TIME FOR DEFORMATION.

Dataset	1	2	3
GPU runtime (s)	0.02	0.05	0.13
CPU runtime (s)	4.03	15.34	17.13
GPU speedup	263.0	287.5	132.1

TABLE V
COMPUTATIONAL TIME FOR RESAMPLING.

Dataset	1	2	3
GPU runtime (s)	1.32	4.13	4.37
CPU runtime (s)	70.81	236.88	256.12
GPU speedup	53.7	57.3	58.6

The obtained results are presented in tables I-V. Table I presents the timing results for the whole algorithm and the achieved speedup is in the range of 51-54x. Note that in the timing results for the GPU, the time needed to transfer the data between the CPU and GPU has not been included. The reason for this is that the time required is negligible, was in the order of 0.5-1.5 s depending on dataset. Tables II-IV present the timing results of different sub-steps of the Morphon, i.e. local displacement estimation, deformation field accumulation and deformation. Here the relative speedups differ between the different sub-steps, with an achieved speedup of approximately 58x, 30x and 200x. Table V presents the time needed for resampling the datasets and the results between different scales. For resampling the achieved speedup is approximately 55x.

VII. DISCUSSION

The results in Table I-VI and V are the expected, i.e. we have an obvious performance improvement with the GPU implementation and the relative speedup is the same regardless of the size of the datasets. The fact that the speedup differs between the different sub-steps is to be expected. For instance, the local displacement estimation and the resampling steps include extensive use of an ordinary convolution kernel whereas the accumulation of the deformation field uses a separable convolution kernel.

Thus, it appears that the performance gain is larger for ordinary convolution than for separable convolution.

However, the results in Table IV are somewhat ambiguous. That the relative speedup would be larger for the deformation step, than compared to the other sub-steps, is to be expected since it is based on built-in trilinear interpolation using 3D textures, which is a highly specialized task for GPUs. Despite this, the results for datasets 1 and 2 seem a bit too extreme. A possible explanation is that the utilized functions for measuring the computational times, see Section VI, have a limited accuracy. This has been indicated by some developers in the user forums of CUDA.

In our tests we have not included a multi-threaded CPU implementation. How much an optimized multi-threaded CPU implementation would affect the results is a highly debated question [8]. A simple way to simulate an optimized multi-threaded CPU implementation would be to divide the results with the number of available cores on the CPU (e.g. four in our case) and with the SIMD (Single Instruction Multiple Data) width (e.g. four in our case). This would lower the relative speedup to 3.5x instead of approximately 50x. However, this is the theoretical speedup that would be achieved by fully exploiting the available multi-threading and SIMD support. To actually achieve this improvement of a CPU implementation is very difficult and dependent on a number of things, such as cache access patterns and inter-core communication. For instance, the multi-threaded solution in [5] only changes the speedup factor from 55x to 35x.

We have not provided any similarity or distance measures to compare the accuracy of the GPU and the CPU implementations, simply because there were not any obvious differences in the end result of the registrations. The relative difference was less than 0.003 for all datasets when using normalized cross-correlation as similarity measure. This is relevant to observe since the GPU implementation uses single precision whereas the CPU implementation uses double. Another interesting point when comparing the GPU and the CPU implementations could be observed when analyzing the implementations with the profiling tools provided in CUDA respectively MATLAB. About 87-89% of the total runtime is used for convolution.

In our comparison, we have used a MATLAB implementation as CPU implementation. Since MATLAB is not considered to be the most computationally efficient platform for a CPU implementation, it must be noted that almost 90% of the total runtime in MATLAB was used by the function `imfilter`, which is a mex-function. Thus, the extensive usage of `imfilter` vouches for that it is fair to use the MATLAB implementation as a reference implementation for a single core, single-threaded CPU implementation.

The achieved speedups are in the same magnitude as the speedups reported in [5], [6], [7]. This further supports the notion that GPGPU computing is an easy (and cheap) method for improving the performance of the algorithms that support parallelization.

An important observation to make here is the difference in actual runtime when comparing the CUDA implementations of the Morphon and the Demons, where the Demons is 3.5-5 times faster than the Morphon for a dataset of the size 256x256x128. Since we have not been able to compare the implementations on the same test data, we cannot say anything about the difference in accuracy of the actual registration result and thus whether the actual runtime results are valid to compare. However, it is important to note that the Demons algorithm is based on the intensity differences between two images whereas the Morphon utilizes the phase-difference, which has been shown to be superior in cases with varying contrast between the images to register [9], [10].

Future work includes better optimization of the convolution kernels, better comparison with other GPU based implementations and comparison with other GPU based parallelization techniques such as OpenCL and DirectCompute.

ACKNOWLEDGEMENT

This work was funded by the Swedish Research Council and grant 2007-4786.

REFERENCES

- [1] J. Modersizki, *Numerical Methods for Image Registration*. Oxford University Press, 2004.
- [2] H. Knutsson and M. Andersson, "Morphons: Segmentation using elastic canvas and paint on priors," in *IEEE International Conference on Image Processing (ICIP'05)*, Genova, Italy, September 2005.
- [3] D. B. Kirk and W.-M. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010.
- [4] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, "A survey of medical image registration on multicore and the GPU," *Signal Processing Magazine, IEEE*, vol. 27, no. 2, pp. 50–60, 2010.
- [5] P. Muyan-Ozcelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," *Computational Science and its Applications, International Conference*, pp. 223–233, 2008.
- [6] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. Choi, E. Castillo, A. Majumdar, T. Guerrero, and S. B. Jiang, "Implementation and evaluation of various demons deformable image registration algorithms on a GPU," *Physics in Medicine and Biology*, vol. 55, no. 1, pp. 207–219, 2010.
- [7] X. Han, L. Hibbard, and V. Willcut, "GPU-accelerated, gradient-free MI deformable registration for atlas-based MR brain image segmentation," *Computer Vision and Pattern Recognition Workshop*, pp. 141–148, 2009.
- [8] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chenupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, 2010.
- [9] B. Rodríguez-Vila, J. Pettersson, M. Borga, F. García-Vicente, E. J. Gómez, and H. Knutsson, "3D deformable registration for monitoring radiotherapy treatment in prostate cancer," in *Proceedings of the 15th Scandinavian conference on image analysis (SCIA'07)*, Aalborg, Denmark, June 2007.
- [10] G. Janssens, L. Jacques, J. O. de Xivry, X. Geets, and B. Macq, "Diffeomorphic registration of images with variable contrast enhancement," *International Journal of Biomedical Imaging*, 2010.