Evaluation of Medical Image Algorithms on Multicore Processors

Damir Demirović
Department of Computer Science and Informatics, Faculty of Electrical Engineering,
University of Tuzla, Bosnia and Herzegovina
E-mail: damir.demirovic@untz.ba, http://www.fe.untz.ba

Zekerijah Šabanović
Medical Faculty, University of Tuzla, Bosnia and Herzegovina
E-mail: zekerijah.sabanovic@untz.ba, http://www.medf.untz.ba

Keywords: medical image processing, multicore processor, GPU, GPGPU, filtering, image registration

Received: April 26, 2016

Introduction: In recent time medical image processing and analysis became an essential component inclinical practice. Medical images contain huge data to process due to increased image resolution. Thesetasks are inherently parallel in nature, so they naturally fit to parallel processors like Graphics Processing Unit (GPU). In this work several commonly used image processing algorithms for 2-D and 3-D were evaluated regarding the computation performance increase using the GPUs and CPUs on a personal computer. For tested algorithms, GPU outperforms CPU from 1.1 to 422 times.

Povzetek: V zadnjem času je obdelava in analiza medicinskih slik postala bistvena sestavina v klinični praksi. Medicinske slike vsebujejo ogromne količine podatkov, vendar je procesiranje slik vzporedne narave, posebej primerno za obdelavo z grafično procesno enoto (GPU). V tem delu smo ocenili več pogosto uporabljenih algoritmov za obdelavo slik za 2-D in 3-D glede povečanja zmogljivosti računanja z grafičnimi procesorji na osebnem računalniku. Za testirane algoritme je grafični procesor omogočil zmanjšanje časa računanja od 1,1 do 422-krat.

1 Introduction

In the last decade parallel processing has become the most dominant for high-performance computing. Increasing the processor clock rate in single-core processors has slowed down due to the problems with heat dissipation. Application developers cannot count on Moore's law to make complex algorithms computationally feasible. Consequences are that they are increasingly shifting the algorithms to parallel computing architectures [1][2]. These architectures are multicore Central Processing Units (CPU), Graphical Processing Units (GPU) and Field-Programmable Gate Array (FPGA).

The amount of data processed in clinical practice is also increasing. Increased resolution of medical images and a huge amount of data for processing is exploding. Trends like 3-D and 4-D imaging technologies used in treatment planning need a lot of computer power. Due to its nature, these tasks are inherently data-parallel, i.e. data from such dataset can be processed in parallel using multiple threads. GPUs originally designed for acceleration of computer graphics, become a versatile platform for running massively parallel computation. This is due to its nature, like high memory bandwidth, high computation throughput etc. [2]. In the year, 2004 programmable GPUs were introduced. Firstly, they could run in parallel custom programs called shaders. This is the first time to accelerate the non-graphical applications with GPUs.

Today GPU become a viable alternative to CPUs in time-consuming tasks. When same computations can be performed on many image elements in parallel, so it can easily fit on GPUs. Two dominant parallel computing platforms are NVidia CUDA and OpenCL.

OpenCL [3] is a software framework for writing programs that run across heterogeneous platforms like CPUs, GPUs, digital signal processors (DSPs) and FPGAs. Heterogeneous refers to systems with more the one kind of processors or cores. Both CUDA and OpenCL support heterogeneous computing. OpenCL is based on a C programming language, and it is an open standard. NVIDIA CUDA [4] is a parallel computing platform and Application Programming Interface (API), which supports programming framework OpenCL.

In [5] authors gave the introduction to the GPU architecture, and its applications in image processing, software development, and numerical applications.

In [2] authors review the principles of GPU computing in the area of medical physics. Segmentation of anatomical structures from image modalities like Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) were given in [6]. Due to its computational complexity most segmentation procedures require vast processing power like GPU. A brief literature review of several segmentation methods is given here.

In [7] authors give a review of applications for GPU in medicine, which covers the past and current trend in this field, like commonly used method and algorithm which are specific to individual image modalities. Also, in the field of medical visualization GPU can be effectively used.

Algorithm Marching Cubes that extract surfaces from volumetric data was presented [8]. Fast extraction in medical applications is necessary, so near real-time applications are very desirable. Their algorithm implementation is completely data-parallel, which is ideal for application on a GPU.

In [10] authors implement widely known Demons algorithm for medical image registration [16] on a GPU, for registering 3-D CT lung images. Speedups of 55 times were reported over non-optimized CPU version.

In [20] authors were using OpenCL to evaluate reconstruction of 3-D volumetric data from C-arm CT projections on a variety of high-performance computing platforms, like FPGAs, graphic cards and multi-core CPUs.

Three-dimensional reconstruction task in cone-beam CT, a computation complex algorithm was implemented using CUDA [21].

Book [9] covers developing data-parallel version of registration algorithms suitable for execution on GPU.

Our main objective was to compare algorithms using CPU and GPU, and their assessment on a different processor architecture. Some of the most used image processing algorithms, which are suitable for algorithm parallelization, were evaluated and speedups were compared to a single core of the CPU. CPU results were used as a base for comparison of the results from the GPU.

2 Methods

In this work, time-consuming algorithms were evaluated on a CPU and GPU. Algorithms for 2D and 3D were tested, and running times were evaluated.

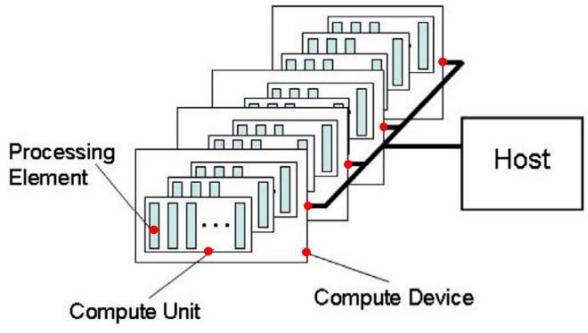

Figure 1: OpenCL platform model [3].

There are several software packages for image processing and analysis of medical images. For the purpose of this research, the different software packages were used, as described as follows.

Plastimatch [11] is an open source software for image computation. The main focus is high-performance volumetric registration of medical images, such as X-ray CT, MRI, and positron emission tomography (PET).

Capabilities	Processor	
	GPU GTX 560Ti	CPU Intel i5-2500
OpenCL version	1.1	not available
Compute capability	2.1	not available
Double precision	Yes	Yes
Number of cores	384	4
Max clock freq. (GHz)	1.7	3.7
Global memory (MB)	1023	6
Power rating (W)	170	95

Table 1: Processor specifications.

Software features include methods for medical image registration, segmentation etc.

OpenMP (Open Multi-Processing) [12] is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, processor architectures and operating systems, including Solaris, AIX, HP-UX, Linux, OS X, and Windows.

OpenCL and CUDA allow heterogeneous programming model, so a typical sequence of operations is the same in both of them. In both platforms, host refers to the CPU and its memory, while device refers to GPU and its memory. Kernels are functions executed on the device (GPU) in parallel. A typical program has the following steps: declaring and allocating host and device memory, initialize host data, transfer data from the host to the device, execute one or more kernels, transfer results from device to the host.

OpenCL is portable API, based on the C99 standard of the C programming language. OpenCL platform model (Figure 1) consists of a host of several computing devices which each contain several computing units. Further, a computing unit contains several processing units. The serial code runs on a Host (which is a CPU) thread, and the parallel code executes in many devices (GPUs) threads across multiple processing elements.

Functions executed on OpenCL devices are called kernels. Both CUDA and OpenCL support built-in functions which can take scalar and vector arguments. Native functions are built-in functions with reduced precision which is implementation defined, but with decreased execution time. Built-in functions conform to IEEE 754 compatible rounding for single precision floating point calculations.

OpenCV [13] is a library of functions for computer vision. It is cross platform and released under the BSD license, written in C++ language, and supports Intel Integrated Performance Primitives (IPP) optimized routines, support for GPUs for CUDA and OpenCL.

In this work nine commonly used algorithms were evaluated. First, algorithms in 2D which can be used for

filtering medical images were evaluated. Medical image datasets usually come as volumes like CT image. They have usually 100 or mores slices, so running times are exceptionally high, which prevents their clinical usage in real time.

All experiments presented in this work were evaluated on a PC computer with Intel CPU and NVidia GPU with 8GB of RAM memory. For the GPU implementation of algorithms NVIDIA CUDA Toolkit version 7.5 was used. CPU implementations were implemented using Microsoft Visual Studio Express 2013. Specifications of the processors for this research are given in Table 1.

For the purpose of research, we choose nine image processing algorithms with frequent usage in medical practice. We have split the analysis of algorithms for 2-D and 3-D images as described in the following sections.

2.1 2-D algorithms

In medical practice 2-D algorithm can be used on a single image slice or extracted images from 3-D volumes. For the purpose of this research, we choose the rotation, Gaussian filter, Sobel filter, Fast Walsh transform, Farneback method and Horn-Schunk optical flow.

Image rotation is a geometric operation which maps the image pixel in an input image onto the position in an output image by rotating the image around the specified angle about an origin. Rotation is a case of an affine transformation, and it is widely used in image processing (for example image registration).

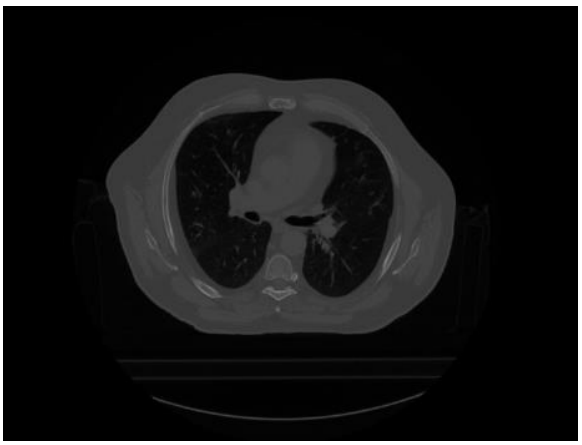

Figure 2: Image used for all 2-D experiments.

Gaussian filter is the most common used in filtering and have significant usage in medical applications (for example in image registration which acts as smoother). Gaussian filter was evaluated for input image of 2048x2048 with parameters sigma 10 pixels and kernel size 81. For purpose of these experiments, Gaussian kernel were implemented on the CPU and the GPU. For CPU, we used up to 4 threads with standard CPU optimizations. For these experiments the image showed in Figure 2 was used.

Fast Walsh or Hadamard transform is a special case of generalized Fourier transforms, which has the same complexity like Fourier transform but without multiplications.

Farneback method for computation of optical flow was presented in [14]. Optical flow was used for the finding of relative motion between two images. It can be used to recover motion for example between two organs. The method is based on approximation of each neighborhood of two frames by quadratic polynomials, using the polynomial expansion transform (images are shown in Figure 3 and Figure 4). Obtained deformation field is shown in Figure 5 and Figure 6, where colors correspond to different values of deformation obtained. Two deformations appear similar but a significant value difference can be seen in the lower and the right part of Figure 5. If we take the CPU implementation as the golden truth, the difference between these two results originates from a loss of computation precision of the GPU.

Horn-Schunck is optical flow method is a classical method for finding the apparent motion in images [15]. The method assumes smoothness in the flow over the whole image and tries to minimize global energy functional which consists of two parts, intensity and regularization. The method employs iterative scheme using Jacobi method. For this experiment, image showed in Figure 3 and Figure 4 were used. Deformation field after registration obtained with this algorithm are showed in Figure 7 and Figure 8. Comparing the obtained deformation fields from two algorithms we found some small differences on the pixels on GPU image (Figure 5 and Figure 6).

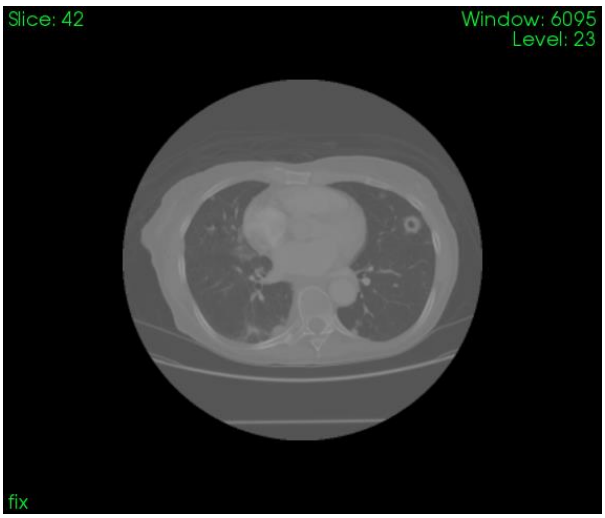

Figure 3: Static image used for all 3-D experiments.

Results for described 2-D algorithms are given in Table 2 and corresponding Figure 9. From the results one can see that almost all algorithms, with exception of image rotation, execute faster on the GPU, and depending on the algorithm speedups are from 10x to 84x compared to one CPU thread. Significant improvements can be also obtained with some loss of the accuracy. Almost all algorithms can be run in a real-time on the GPU, and just one on the CPU.

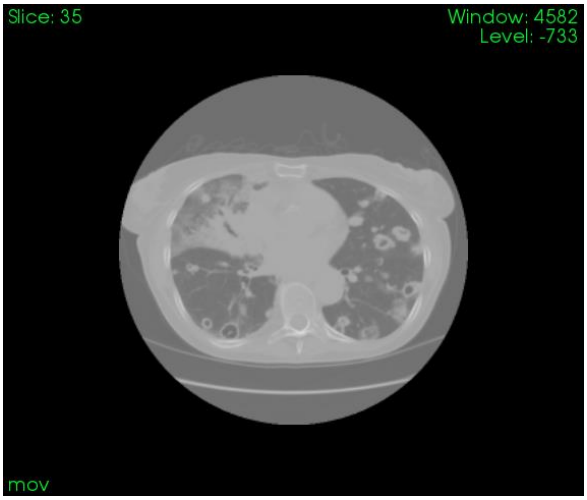

Figure 4: Moving image used for all 3-D experiments.

The Sobel operator is a widely used filter in image processing for edge detection. In 2-D Sobel operator is 3x3 for one dimension, whereas in 3-D 3x3x3 for each of 3 dimensions. The result of Sobel operator is a gradient vector. The filter is separable so it can be written as product of two simpler filters.

For this experiment, we used up to four CPU thread for evaluation. Speedups are given in Table 3 and Figure 10. For the best experiment we can expect the speedup of 38 times for the four CPU cores, or in worst case 141 times compared to one CPU core. From these results can be clearly seen that Sobel algorithm can benefit significantly from implementation on the GPU compared to one CPU thread.

2.2 3-D algorithms

3-D algorithms in medical practice are very important. Most of medical images are 3-D volumes and needs to be preprocessed, analyzed or visualized in some way. In the next part the five widely used algorithms in 3-D were evaluated.

For the purpose of this evaluation, we implemented 3-D Gaussian filter in C programming language. Volume dimensions for tested images were 482x360x141 with kernel size of 5 and sigma 0.5 voxels.

The Sobel operator in 2D has the dimension of 3x3, whereas in 3-D 3x3x3 for each of 3 dimensions. The result of Sobel operator is a gradient vector. The filter is separable so it can be written as the product of two simpler filters, thus reducing the computation time. For this experiment, the same volume was used as in the previous experiment.

All 3-D registration was evaluated for the three resolution levels, with maximal 30, 50 and 50 iterations respectively. Threading in CUDA, OpenMP, and single thread have been used. For registration bspline,

Demons, and affine algorithms from Plastimatch were used. Registration using bsplines falls into a category of Free-Form Deformations (FFD) in which object to be registered is embedded into bspline object [19]. Deformation of bspline object represents the transformation of the registration [17]. Affine image registration falls into a category of linear registration, which is a composition of linear transformations with translations. In this category falls rigid transformations (translating plus rotations), rigid plus scaling and affine. Another category of non-linear registration is non-rigid, deformable, fluid elastic etc. Affine transformation preserves points, straight lines, and planes. After transformation set of parallel lines remains parallel. Affine transformations define translation, scale, shear, and rotation.

Obtained deformations of Horn-Schunk algorithm are shown in Figure 7 and Figure 8, where colors correspond to different values of deformation which was obtained from algorithms running on the GPU and CPU respectively. In contrast to the 2-D Farneback method, some small differences can be spotted between the two deformation fields, which corresponds to very small error for the GPU.

All results obtained with 3-D registration are showed in Table 4 and Figure 11. For these experiments, OpenMP were used with four CPU threads, except for filtering algorithms Gaussian and Sobel. Obtained speedups are from 1x to 422x depending on the algorithm. Lowest speedup is for the affine registration where CPU version of the algorithm is little faster. Highest speedup is for

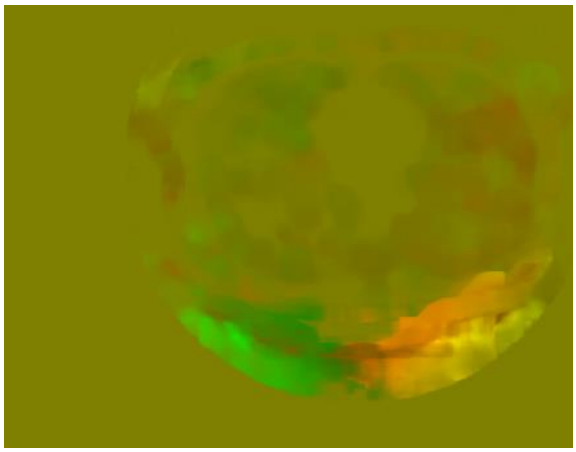
Figure 5: Color representation of deformation field using Farneback algorithm (GPU).

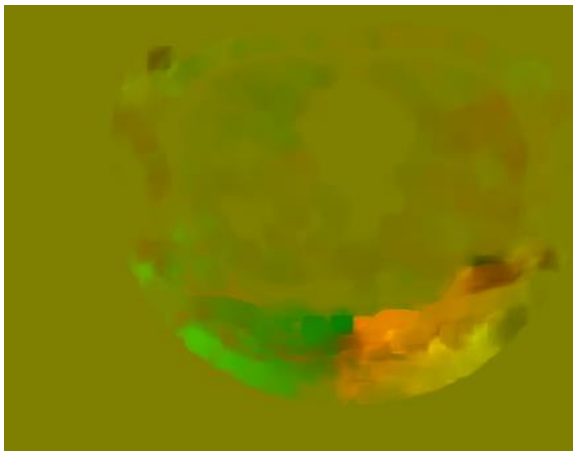
Figure 6: Color representation of deformation field using Farneback algorithm (CPU).

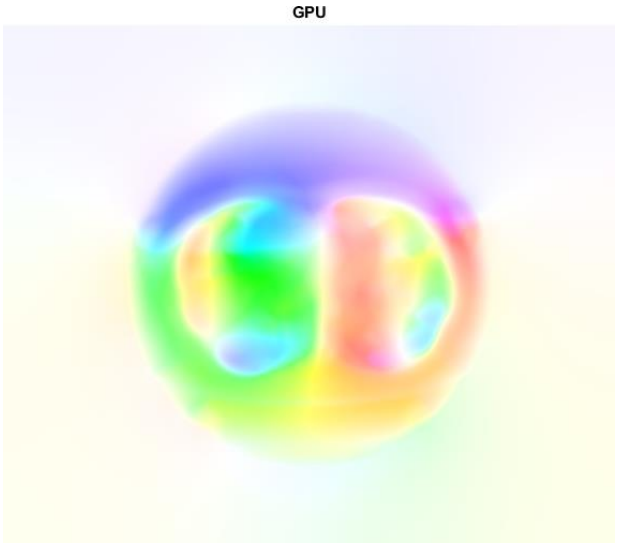

Figure 7: Color representation of deformation field using Horn-Schunk algorithm (GPU).

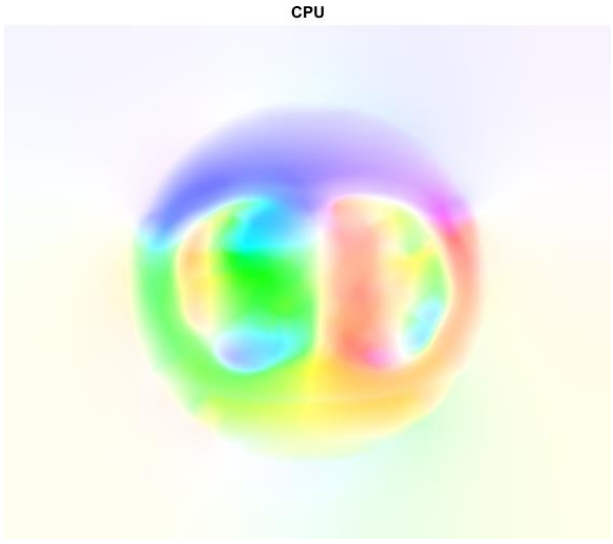

Figure 8: Color representation of deformation field using Horn-Schunk algorithm (CPU).

filtering, from 127x to 422x compared with single CPU thread. Registration algorithm Demons and affine have little or no speedup for 4x, whereas bspline have a significantly lower performance in this case. Algorithms for image registration are highly computing extensive and obtained speedup is from about 1x for affine to 15x for Demons algorithm.

It is worth to mention that Demons algorithm uses Gaussian filter in each iteration to smooth the deformation field. From the running times for Demons, one can see that speedup is almost the same for CPU, which indicates the single thread implementation for this algorithm. Similar to 2-D implementations, there is a trade-off between precision and running time.

3 Conclusions

In this paper was presented an evaluation of speed gain using modern GPU cards compared to the standard CPU. In total, nine common used algorithms on different processors were evaluated using parallel processing for 2-D and 3-D. For the CPU up to 4 threads were used,

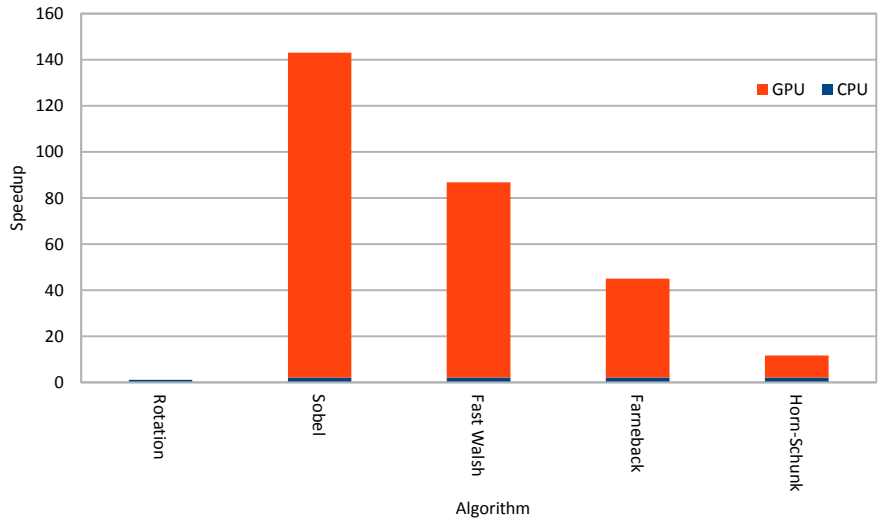

Figure 9: Speedups for 2-D experiments.

Algorithm	**GPU (s)**	**CPU (s)**	**speedup (in times)**
Image rotation	0.0090	0.10	0.01
Fast Walsh transform	0.0399	3.38	**84**
Farneback optical flow [14]	0.0116	0.50	**43**
Horn-Schunk optical flow [15]	1.4200	13.69	**10**

Table 2: Running times and corresponding GPU Speedups for 2-D experiments.

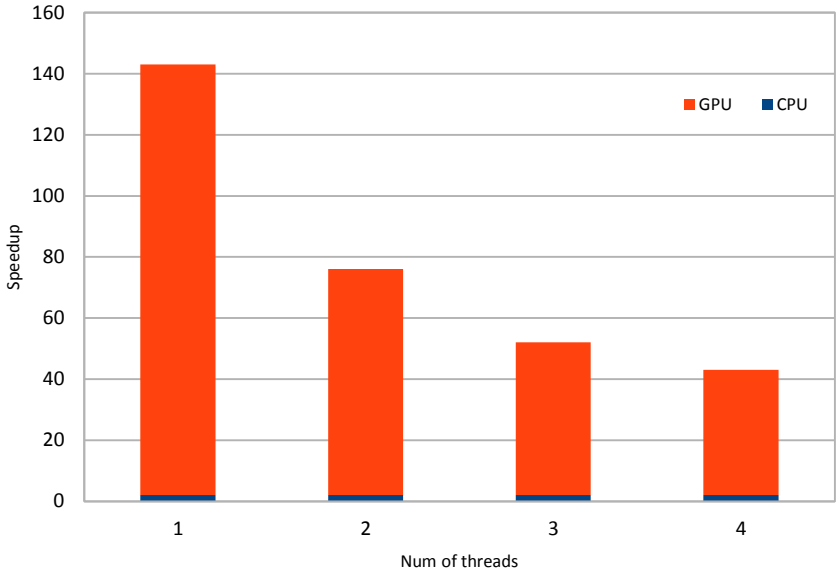

Figure 10: Speedups for Sobel algorithm.

Algorithm	GPU (s)	Number of CPU threads (s)				speedup (in times)			
		1	2	3	4	1	2	3	4
Sobel filter	0.0155	2.2	1.1	0.8	0.6	**141**	**71**	**51**	**38**

Table 3: Running times (in seconds) and corresponding GPU speedups for 2-D Sobel filter.

depending on the algorithm implementation. For the GPU, algorithms were used with simple naïve implementation, without optimization and all available cores.

In almost all cases processing times decrease due to highly parallelizable algorithms. Obtained speedups varied from 1.1x to 422x depending on the algorithm. Some of the tested algorithms was not well suited to parallel implementation, i.e. their running times increased with larger number of threads. Obtained results on a GPU suffers small loss of accuracy, and show near real-time performance.

Future work can evaluate the specific optimizations for CPU and GPU, instructions like SSE, AVX for CPU. Native instructions, determining the optimal local and global block size for CUDA and OpenCL and instructions with lower precision can be analyzed for the GPU. Another possibility for detecting and reducing the bottlenecks in the GPU implementation can be done using a GPU profiler.

4 References

[1] Xue X, Cheryauka A, Tubbs D. Acceleration of fluoro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: a simulation study. Proc. SPIE 6142, Medical Imaging 2006: Physics of Medical Imaging, 61424L (2 March 2006); 2006.

[2] Pratx G, Xing L. GPU computing in medical physics: A review. *Medical Physics*. 2011; 38(5): p. 2685-2697.

[3] Khronos. OpenCL. [Online]. [cited 2016 03 21. Available from: https://www.khronos.org/opencl/.

[4] NVidia. NVIDIA CUDA. [Online].; 2016 [cited 2016 03 15. Available from: http://www.nvidia.com/object/cuda_home_new.html.

[5] Couturier R. Designing Scientific Applications on GPUs: Chapman & Hall CRC; 2013.

[6] Smistad E, Falch TL, Bozorgi M, Elster AC, Lindseth F. Medical image segmentation on GPUs – A comprehensive review. *Medical Image Analysis*. 2015; 20(1): p. 1-18.

[7] Eklund A, Dufort P, Forsberg D, LaConte SM. Medical image processing on the GPU - past, present and future. *Medical Image Analysis*. 2013; 17(8)

[8] Smistad E, Elster AC, Lindseth F. Fast surface extraction and visualization of medical images using OpenCL and GPUs. *The Joint Workshop on High Performance and Distributed Computing for Medical Imaging*. 2011; 2011.

[9] Shackleford J, Kandasamy N, Sharp G. High Performance Deformable Image Registration Algorithms for Manycore Processors. 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2013.

[10] Samant P, Muyan-Ozcelik , Owens JD, Xia J, S. S. Fast Deformable Registration on the GPU: A CUDA Implementation of Demons. *In proceedings of the 1st technical session on UnConventional High Performance Computing (UCHPC) in conjunction with the 6th International Conference on Computational Science and Its Applications (ICCSA)*; 2008; Perugia, Italy. p. 223-233.

[11] Plastimatch. Plastimatch. [Online]; 2016 [cited 2016 04 20]. Available from:http://plastimatch.org/.

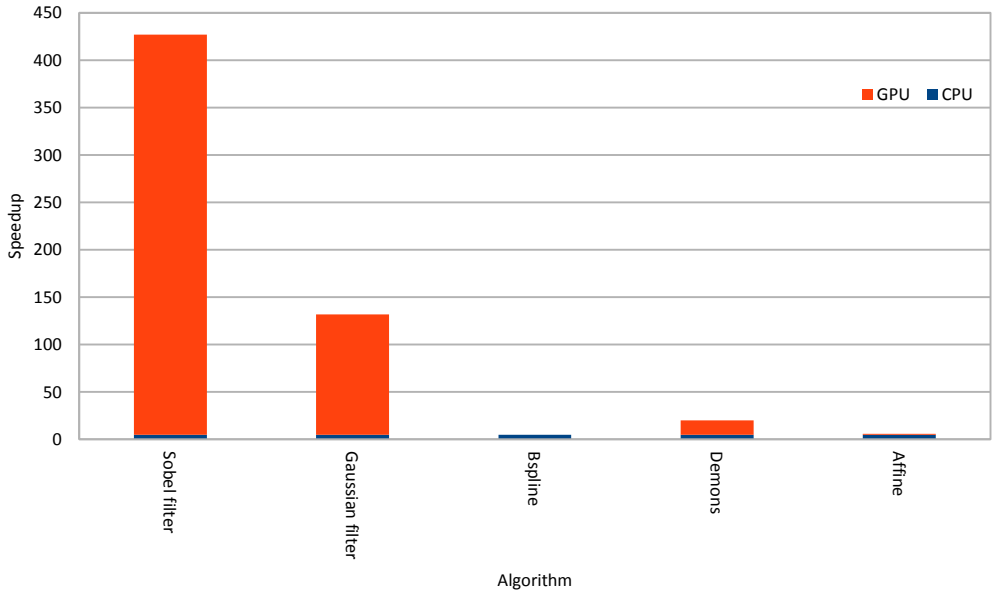

Figure 11: Speedups for 3-D algorithms.

Algorithm	**GPU (s)**	**Number of CPU threads (s)**		**speedup (in times)**	
		1	4	1	4
Sobel filter	0.0557	23.5	-	**422**	-
Gaussian filter	0.7860	100.1	-	**127**	-
Bspline registration [17]	41.4000	323.4	99.6	**8**	**2**
Demons registration [18]	6.5100	98.9	99.0	**15**	**15**
Affine registration	74.7100	69.2	81.2	0.92	**1.1**

Table 4: Running times (in seconds) and corresponding GPU speedups for 3-D algorithms.

[12] OpenMP. http://openmp.org/wp/. [Online].; 2016 [cited 2016 04 02]. Available from: http://openmp.org/wp/.

[13] OpenCV. OpenCV. [Online].; 2016 [cited 2016 04 15. Available from: http://opencv.org/.

[14] Farneb. Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29-July 2, 2003 Proceedings. In Bigun J, Gustavsson T, editors. Berlin, Heidelberg: Springer Berlin Heidelberg; 2003. p. 363-370.

[15] Horn BKP, Schunck BG. Determining Optical Flow. *Tech. rep. Cambridge, MA, USA*; 1980.

[16] Maintz JBA, Viergever MA. A Survey of Medical Image Registration. *Medical Image Analysis, Volume 2 , Issue 1 , 1 - 36* 1998.

[17] Pennec X, Cachier P, Ayache N. Understanding the demon's algorithm: 3D non-rigid registration by gradient descent. *In: Proc. MICCAI'99*; 1999.

[18] Thirion JP. Image matching as a diffusion process: an analogy with Maxwell's demons. *Medical Image Analysis*. 1998 sep; 2(3): p. 243-260.

[19] Tustison NJ, Avants BA, Gee JC. Improved FFD B-Spline Image Registration. *Computer Vision, IEEE International Conference on*. 2007; 0: p. 1-8.

[20] Siegl C, Hofmann HG, Keck B, Prümmer M, Hornegger J. OpenCL: a viable solution for high-performance medical image reconstruction? *Proceedings of SPIE (Medical Imaging 2011: Physics of Medical Imaging)*, Lake Buena Vista, Florida, USA, 12 - 17 Feb 2011, vol. 7961, pp. 79612Q, 2011

[21] Scherl H, Keck B, Kowarschik M, Hornegger J. Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA). *In Nuclear Science Symposium Conference Record*, 2007. NSS '07. IEEE; 2007 Oct. p. 4464-4466.