GPU-BASED MOTION CORRECTION OF CONTRAST-ENHANCED LIVER MRI SCANS: AN OPENCL IMPLEMENTATION

Jihun Oh[1], *Diego Martin*[3], *Oskar Škrinjar*[1,2]

[1] School of Electrical and Computer Engineering, Georgia Tech, Atlanta, GA 30332, USA
[2] Department of Biomedical Engineering, Georgia Tech, Atlanta, GA 30332, USA
[3] Department of Radiology, Emory University School of Medicine, Atlanta, GA 30322, USA

ABSTRACT

Clinical diagnosis and quantification of liver disease have been improved through the development of techniques using contrast-enhanced liver MRI sequences. To qualitatively or quantitatively analyze such image sequences, one first needs to correct for rigid and non-rigid motion of the liver. For motion correction of the liver, we have employed bi-directional local correlation coefficient Demons, which is a variation of the original Demons method. However, despite the intrinsic speed of the Demons method, the run-time on the order of an hour of its CPU-based implementation is not sufficiently short for a regular clinical use. For this reason we implemented the method on a graphics processing unit (GPU) using OpenCL. On NVIDIA GTX 260M, which is a laptop GPU, we achieved sub-minute runtime for the motion correction of typical liver MRI scans, which was ∼50 times faster than its CPU-based implementation. A sub-minute runtime of liver MRI motion correction allows for its regular clinical use.

Index Terms— graphics processing unit, GPU, OpenCL, contrast-enhanced liver MRI, image registration, Demons

1. INTRODUCTION

In the United States liver disease is the 4th leading cause of death during the most productive adult years [1]. Hepatitis results from a variety of etiologies, all having the capacity to induce inflammation and fibrosis leading to chronic liver disease (CLD). CLD is a common cause of primary liver malignancy, which is becoming one of the more common malignancies overall. Metastatic liver tumors are relatively common, and the liver normally frequently generates benign tumors. Contrast enhanced MRI has become the primary imaging modality of choice for delineation of liver disease. The capacity to compare sequential pre- vs. post-contrast dynamically enhanced liver images has been limited to observer-based analysis. The use of automated computer aided analysis would be helpful, but currently it is limited by the lack of available commercial software tools.

To qualitatively or quantitatively analyze features in liver image sequences one first needs to correct for rigid and non-rigid motion of the liver, i.e. to register the images in the sequence. As a very popular non-rigid image registration method, Thirion in 1998 introduced Demons [2], in which numerical computation of incremental displacement field for each iteration is simple and highly parallelizable. Its extended version, local correlation coefficient (LCC) Demons [3], replaces the assumption of the Demons method that the intensities of corresponding locations of the two images are identical with a locally affine intensity relations. This is achieved by computing local correlation coefficient between the two images centered at each voxel. Its accelerated version, bi-directional LCC (bi-LCC) Demons, is introduced in Sec. 2.2.2.

Despite the computational efficiency of Demons and its variations, their execution time for the registration of typical liver MRI scans is on the order of one hour when implemented in CPU. However, execution times on the order of one minute or less are needed for regular clinical use. Graphics processing units (GPU), which typically have 100+ processing elements, on-board memory of over 1 GB, and a high bandwidth (25+ time faster than the bandwidth of CPU main memory) for data transfer, have been used for computationally intensive processing tasks. E.g., in the image registration field researchers have studied a CUDA implementation of the Demons method [4, 5]. However, a restriction of CUDA is that it is platform-, vendor-, and hardware-specific, in contrast to Open Computing Language (OpenCL), which supports a diverse mix of parallel CPUs, GPUs, and other processing units [6].

2. METHODS

2.1. MRI Acquisition Protocols and Subjects

Gadolinium-based contrast-enhanced MRI has been utilized to detect the presence of hepatic inflammation in acute and chronic liver disease. Seven scans were obtained at 20 sec, 1 min, 3 min, 5 min, 10 min, 15 min, and 20 min after the administration of the contrast agent. For each post-contrast scan, we corrected for liver motion relative to the pre-contrast scan. The scan had 128 contiguous slices with 256 x 256

pixels, in-plane resolution of $1.46\ mm$ and slice thickness of $3\ mm$.

2.2. Image Registration

2.2.1. Translation-only Registration

Based on inspection of hepatic motion, translational movement represents its major portion, with the dominant component in the cranio-caudal direction, some translation in the anterior-posterior direction and the least translation in the lateral direction. For this reason, we ignore the rotation of the liver in this initialization step. We search for the translational direction (among the current position and six test positions: $\pm\Delta t_x$, $\pm\Delta t_y$, and $\pm\Delta t_z$) for which the normalized cross correlation (NCC) over the segmented liver dilated with a few extra layers of voxels is maximized. This process is iterated until NCC can no longer be improved. Then we reduce the step size and repeat the process to achieve coarse-to-fine translational alignment.

2.2.2. Bi-directional Local Correlation Coefficient (LCC) Demons

While the translation-only registration corrects for most of rigid body motion, due to tissue deformation the liver is not fully aligned. To correct for the remaining non-rigid motion, we introduce an accelerated version of local correlation coefficient (LCC) Demons, bi-directional LCC demons. The incremental displacement field for each iteration is

$$\vec{u} = \frac{2E_M\vec{\nabla}E_M}{\left\|\vec{\nabla}E_M\right\|^2 + 4\alpha^2(E_M)^2} - \frac{2E_S\vec{\nabla}E_S}{\left\|\vec{\nabla}E_S\right\|^2 + 4\alpha^2(E_S)^2}, \quad (1)$$

where E_M, E_S, and their simplified energy gradient expressions are defined as

$$\sigma_p^2(S) = \overline{S^2}_p - \overline{S}_p^2 = G_{\sigma_{lcc}} * (S^2)[p] - (G_{\sigma_{lcc}} * S)^2[p], \quad (2)$$

$$\langle S, M\rangle_p = \overline{SM}_p - \overline{S}_p\overline{M}_p, \quad (3)$$

$$= G_{\sigma_{lcc}} * (SM)[p] - (G_{\sigma_{lcc}} * S)[p](G_{\sigma_{lcc}} * M)[p], \quad (4)$$

$$E_M = LCC_p(S, M\circ u) = \frac{< S, M\circ u >_p}{\sigma_p(S)\sigma_p(M\circ u)}, \quad (5)$$

$$E_S = LCC_p(S\circ u, M) = \frac{< S\circ u, M >_p}{\sigma_p(S\circ u)\sigma_p(M)}, \quad (6)$$

$$\vec{\nabla}E_M \approx \frac{\vec{\nabla}M}{\sigma_p(S)\sigma_p(M)}\left(\hat{S} - \hat{M}\frac{< S, M >_p}{\sigma_p^2(M)}\right), \quad (7)$$

$$\vec{\nabla}E_S \approx \frac{\vec{\nabla}S}{\sigma_p(S)\sigma_p(M)}\left(\hat{M} - \hat{S}\frac{< S, M >_p}{\sigma_p^2(S)}\right), \quad (8)$$

$$\hat{S} = S - \overline{S}_p,\ \hat{M} = M - \overline{M}_p. \quad (9)$$

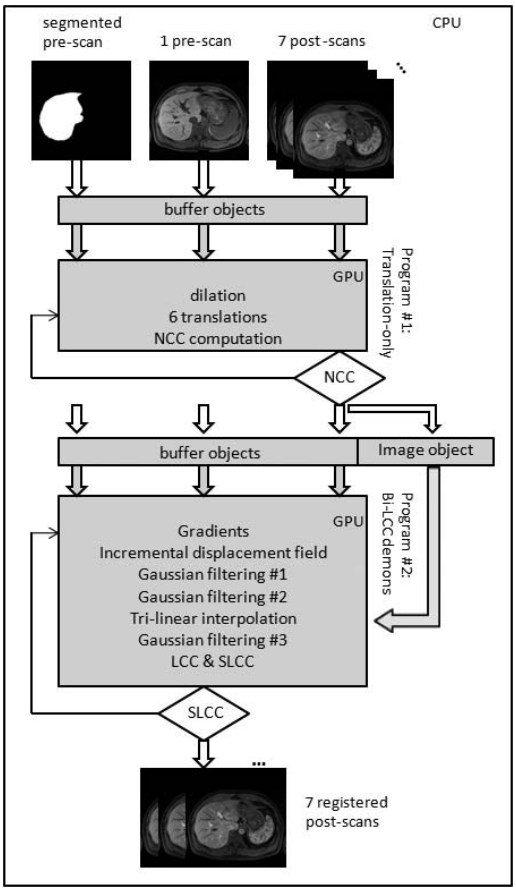


Fig. 1. GPU-based liver image registration system

In the above equations S represent the static image, M the moving image, G_σ Gaussian filtering with standard deviation σ, and p the point at which the equations are computed.

In this method Gaussian filtering is used for three purposes: to compute the local statistics at p (σ_{lcc}), to regularize the incremental displacement field (σ_{fluid}), and to regularize the total displacement field ($\sigma_{diffusion}$). The iterations would normally stop when the sum of LCC over the entire liver no longer improves. However, to avoid small local maxima, we run an additional number of iterations.

2.3. GPU Implementation Using OpenCL

OpenCL supports a relaxed version of the data parallel programming and an implicit model. The programmer specifies the number of work-items in a work-group and the total number of work-items. A division into the work-groups is automatically managed by the OpenCL implementation. Fig. 1 illustrates our system of GPU-based liver MR image registration. It consists of two programs, "translation-only registration" and "bi-LCC Demons," and each program has kernels necessary for its implementation. Input images (seg-

mented, fixed, and moving images) are written into the buffers in global memory of GPU. The output image of each program is saved back to host memory. The rigid-body transformed moving image is additionally written into an image object by utilizing hardware interpolation.

To efficiently implement the motion correction method, we utilize the following GPU resources: pre-compilation, hardware interpolation, and parallel reduction operation. First, in contrast to CUDA in which programs are compiled with an external tool before execution, the OpenCL compiler is invoked at runtime. To pre-compile OpenCL, programmers can use the clGetProgramInfo() API call to retrieve a compiled binary and save it for later use. Then, with the clCreateProgramWithBinary() call, one can create an OpenCL program object directly from the pre-compiled binary. Second, an image object is used to store a two- or three-dimensional texture, frame-buffer, or image. The built-in image function, "read_imagef," reads a sampled (interpolated) value at a non-integer coordinate of the image object with either a nearest neighbor or a linear option. In the kernel, after elements of the input are read from the image object, their hardware-interpolated elements are written back to the buffer object, which stores the updated elements of the moving image. Third, when reducing an array of values to a single value in parallel such as NCC or SLCC, the strategy of parallel reduction is very important for effective processing. The details about conflict-free sequential addressing and complete unrolling techniques involved in parallel reduction are presented in SDK guides provided by NVIDIA. In addition, page-locked memory transfers attain the highest bandwidth (5+ GB/s) between a host and a device, and cached memory (constant or texture memory) is useful for storing frequently-loaded and small data, e.g., Gaussian kernels.

2.4. Validation

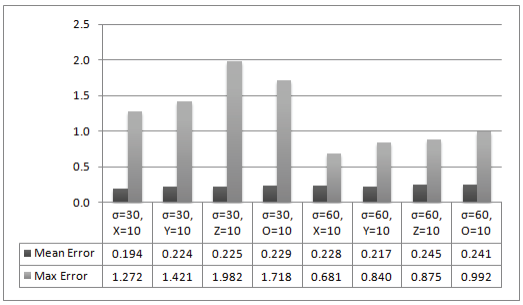
To validate the method, we generated simulated images by artificially deforming real images using a divergence-free deformation model. A selected point $P = (P_x, P_y, P_z)$ is moved in a desired direction by displacement Δ, which smoothly pulls along its neighborhood of size σ. Assuming P moves in the z direction, the displacement field of the deformation model is

$$u_x = \Delta \frac{(x - P_x)(z - P_z)}{x^2 + y^2} \left[e^{-\frac{(z-P_z)^2}{2\sigma^2}} - K \right], \quad (10)$$

$$u_y = \Delta \frac{(y - P_y)(z - P_z)}{x^2 + y^2} \left[e^{-\frac{(z-P_z)^2}{2\sigma^2}} - K \right], \quad (11)$$

$$u_z = \Delta K, \quad (12)$$

where $K = e^{-\frac{(x-P_x)^2+(y-P_y)^2+(z-P_z)^2}{2\sigma^2}}$. We tested eight different situations: the movement of a liver point P in the x, y, z, and oblique ($\sqrt{\frac{1}{3}}, \sqrt{\frac{1}{3}}, \sqrt{\frac{1}{3}}$) directions, with $\Delta = 10\ mm$ and $\sigma = 30\ mm$ for local deformation, and $\Delta = 10\ mm$ and $\sigma = 60\ mm$ for global deformation. To account for the contrast-enhancement, we applied the following intensity transformation: $I_{simulated} = 1.4 \cdot I_{real,deformed} + 96$.



	σ=30, X=10	σ=30, Y=10	σ=30, Z=10	σ=30, O=10	σ=60, X=10	σ=60, Y=10	σ=60, Z=10	σ=60, O=10
Mean Error	0.194	0.224	0.225	0.229	0.228	0.217	0.245	0.241
Max Error	1.272	1.421	1.982	1.718	0.681	0.840	0.875	0.992

Fig. 2. Simulation results: the mean and max displacement errors for the eight cases. All units are millimeters.

Including our GPU, the mobile GPU models before the Fermi architecture of NVIDIA or 6xxx level of ATI do not support the double precision floating-point format (double). With only the single precision floating-point datatype available in our GPU, we validated the motion correction results against the simulated motion, which was generated using double-precision arithmetic.

3. RESULTS AND CONCLUSION

The laptop used in the experiments was ASUS Notebook, with Intel Core2 Duo CPU P8700 @ 2.53GHz and 6 GB of main memory, running Window 7. The NVIDIA GTX 260M mobile GPU supporting OpenCL 1.0 and compute capability 1.1 was integrated. The GPU has 14 compute units (multiprocessors) with 8 processing elements (scalar cores) each, or 112 processing elements in total and a performance of 462 GFLOPS. The programming tool was Microsoft Visual Studio 2008 in which C/C++ programs were built in a release mode.

Fig. 2 shows that the mean and max errors computed relative to the simulated displacement field are smaller than or equal to the average of voxel size, $1.98\ mm$, and the local deformation has a larger error since the parameters of the bi-LCC Demons algorithm were optimized for the global deformation. Fig. 3 shows comparison to the result using the CPU double datatype. The GPU implementation with float datatype and hardware interpolation exhibits slightly bigger errors than that the GPU implementation with software interpolation, but the errors are still negligible while the runtime is reduced by 16.1%.

Table 1 shows the speedup contribution of each technique used in the GPU implementation of the liver MRI motion correction obtained with real images. The largest speedup is achieved by the parallel reduction, in which all

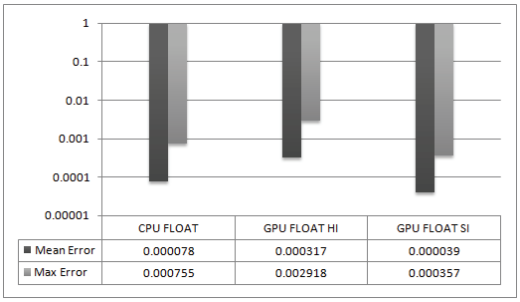


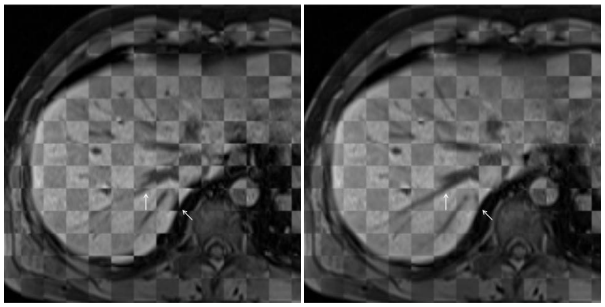
Fig. 3. Simulation results: the mean and max displacement errors for the method implementations in CPU, GPU with hardware and software interpolation, all using float precision, relative to the CPU implementation with double precision. All units are millimeters.

	Speedup Factor (%)
Pre-compilation	3.7
Hardware interpolation	16.1
Parallel reduction	51.2

Table 1. Speedup factors introduced by the three techniques of GPU programming

	CPU [s]	GPU [s]	SUC
Gradient	0.28	0.029	9.7
Incremental displacement	1.03	0.013	79.2
Three Gaussian filtering	21.79	0.36	60.5
Tri-linear interpolation	1.62	0.039	41.5
LCC\SLCC	0.32	0.0060	53.3
Misc (*, +, ...)	2.90	0.11	25.7
One iteration	27.94	0.56	49.9
100 iterations	2790	56	50.0
Translation-only	4.03	1.76	2.3

Table 2. The comparison of the CPU and GPU run time for each kernel, as well as the speed up factor (SUC).



(a) (b)

Fig. 4. Checkerboard display of a pre-contrast scan and post-contrast scan before (a) and after (b) motion correction. Note the improvement in alignment indicated by the white arrows.

work-items need to be busy and escape data conflict. Table 2 indicates that the GPU implementation outperforms the CPU implementation by approximately 50 times. In GPU implementation it took 1.76 sec for translation-only and 56 sec for bi-LCC demons registration, thus 57.76 sec for one post-contrast scan and 6.7 min for a sequence of seven post-contrast scans. In contrast, in CPU implementation it took 4.03 sec for translation-only and 46.5 min for bi-LCC demons registration, thus 46.5 min for one post-contrast scan and 5.4 hrs for a sequence of seven post-contrast scans. Note that the Gaussian filtering is the most expensive process because this kernel has the most GFLOP per iteration. Fig. 4 shows alignment improvement achieved by the motion correction of a pre-contrast and post-contrast liver MRI scan of a patient.

We showed that a GPU implementation of the motion correction of contrast-enhanced liver MRI scans can reduce the run-time by a factor of 50 without sacrificing accuracy. The approach does not require specialized hardware; in fact, the results were achieved on a consumer laptop. Since the method was implemented in OpenCL, it is not restricted to a specific vendor or hardware. The sub-minute run time allows for a regular clinical use of the liver motion correction.

4. REFERENCES

[1] Pickle LW, Mungiole M, Jones GK, and White AA, *Atlas of United States Mortality*, National Center of Health Statistics, 2010, http://www.cdc.gov/nchs/data/misc/atlasmet.pdf, accessed June 2010.

[2] J.P. Thirion, "Image matching as a diffusion process: an analogy with Maxwell's demons," *Medical Image Analysis*, vol. 2, no. 3, pp. 243–260, 1998.

[3] P. Cachier and X. Pennec, "{3D} Non-Rigid Registration by Gradient Descent on a Gaussian-Windowed Similarity Measure using Convolutions," *Proc. of MMBIA00*, pp. 182–189, 2000.

[4] S.S. Samant, J. Xia, P. Muyan-
"Ozçelik, and J.D. Owens, "High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy," *Medical Physics*, vol. 35, pp. 3546, 2008.

[5] P. Muyan-Ozcelik, J.D. Owens, J. Xia, and S.S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," in *International Conference on Computational Sciences and Its Applications*, 2008, pp. 223–233.

[6] A. Munshi, "OpenCL 1.0 Specification," *Khronos Group, May*, 2009.