PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications

Parag Bhosale, Marius Staring, Zaid Al-Ars, Floris F. Berendsen

Parag Bhosale, Marius Staring, Zaid Al-Ars, Floris F. Berendsen, "GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications," Proc. SPIE 10574, Medical Imaging 2018: Image Processing, 105740R (2 March 2018); doi: 10.1117/12.2293098

SPIE.

Event: SPIE Medical Imaging, 2018, Houston, Texas, United States

GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications

Parag Bhosale[a,b], Marius Staring[b,c], Zaid Al-Ars[a], and Floris F. Berendsen[b]

[a]Computer Engineering, Delft University of Technology, Delft, The Netherlands
[b]Division of Image Processing, Leiden University Medical Center, Leiden, The Netherlands
[c]Department of Intelligent Systems, Delft University of Technology, Delft, The Netherlands

ABSTRACT

Currently, non-rigid image registration algorithms are too computationally intensive to use in time-critical applications. Existing implementations that focus on speed typically address this by either parallelization on GPU-hardware, or by introducing methodically novel techniques into CPU-oriented algorithms. Stochastic gradient descent (SGD) optimization and variations thereof have proven to drastically reduce the computational burden for CPU-based image registration, but have not been successfully applied in GPU hardware due to its stochastic nature. This paper proposes 1) NiftyRegSGD, a SGD optimization for the GPU-based image registration tool NiftyReg, 2) random chunk sampler, a new random sampling strategy that better utilizes the memory bandwidth of GPU hardware. Experiments have been performed on 3D lung CT data of 19 patients, which compared NiftyRegSGD (with and without random chunk sampler) with CPU-based elastix Fast Adaptive SGD (FASGD) and NiftyReg. The registration runtime was 21.5s, 4.4s and 2.8s for elastix-FASGD, NiftyRegSGD without, and NiftyRegSGD with random chunk sampling, respectively, while similar accuracy was obtained. Our method is publicly available at https://github.com/SuperElastix/NiftyRegSGD.

Keywords: Non-rigid image registration, stochastic gradient descent, GPGPU, memory access optimization, random chunk sampling

1. DESCRIPTION OF PURPOSE

Image registration is widely used in clinical applications. The high number of parameters in non-rigid registration (up to 100k in some cases) makes this approach computationally intensive, resulting in a rather high computation time. This limits non-rigid registration from being used in real-time critical applications, such as image-guided surgery or online adaptive radiotherapy.

To address this problem, FASGD[1] was proposed recently, which speeds up the adaptive[2] version of stochastic gradient descent optimization (SGD).[3] In SGD, the cost function gradients are computed using a random subset of samples (i.e. voxels) instead of the entire set of sample space (i.e. full image). A random subset is generated each iteration. Hence, this approach reduces data intensity which results in less computation time as compared to conventional gradient descent methods. High performance computation approaches such as NiftyReg[4] and Plastimatch[5] take advantages of the GPU architecture for parallel computing. These implementations are, however, based on gradient descent optimization using full image sampling. In Shamonin et al.,[6] CPU and GPU parallelization were implemented in `elastix`.[7] In the field of machine learning or neural networks, there exist implementations of SGD on the GPU.[8] However, these methods typically optimize over a large collection of data and the term stochastic refers to random batches of data instead of random voxels within one image, which is a fundamental difference from a registration point of view.

This paper presents the implementation of SGD on the GPU architecture for image registration to take advantage of both stochastic optimization as well as parallel computing. For the proposed work, coined NiftyRegSGD, NiftyReg was chosen as a foundation of our implementation for its public access and pre-existing use of the GPU. The accuracies and timings of lung CT registrations are compared with `elastix`-FASGD, which has been evaluated using the same data and is the fastest publicly available method to our knowledge.

Further author information: (Send correspondence to Parag Bhosale)
Parag Bhosale: E-mail: P.S.Bhosale@student.tudelft.nl, Telephone: +31 65 15 11725
Floris Berendsen: E-mail: F.Berendsen@lumc.nl, Telephone: +31 71 52 66206

Medical Imaging 2018: Image Processing, edited by Elsa D. Angelini, Bennett A. Landman, Proc. of SPIE
Vol. 10574, 105740R · © 2018 SPIE · CCC code: 1605-7422/18/$18 · doi: 10.1117/12.2293098

2. METHOD

Our implementation NiftyRegSGD is based on NiftyReg. In this section we discuss the several changes that have been carried out to implement a stochastic gradient descent method and we introduce the random chunk sampler. Image registration may be formulated as an iterative optimization problem, using:

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \gamma_k \mathbf{g}_k, \tag{1}$$

where $\boldsymbol{\mu}$ represents the transformation parameters at iteration k, $\mathbf{g}$ is the search direction and γ_k the stepsize.

The optimizer of NiftyReg uses a search direction $\mathbf{g}_k$ based on the gradient in combination with a line search strategy to select the optimal stepsize γ_k. The line search of NiftyReg inherently imposes a stopping criterion for k. For the search direction either the gradient $\mathbf{g}_k := \partial C/\partial \boldsymbol{\mu}$, or the conjugate gradient of the cost function C can be selected. The cost function in image registration typically takes the following form:

$$C(\mu) = \Psi\left(\frac{1}{|\Omega_F|} \sum_{x_i \in \Omega_F} \xi(F(x_i), M(\mathbf{T}(x_i, \mu)))\right), \tag{2}$$

where $\Psi(u)$ and $\xi(u, v)$ are continuous and differentiable functions, and where Ω_F is a discrete set of voxel coordinates from the fixed image. The key idea in SGD[3] is to compute a fast but noisy approximation of the search direction $\tilde{\mathbf{g}}_k := \partial \widetilde{C}/\partial \boldsymbol{\mu}$, by randomly selecting a small subset of all fixed image coordinates. In each iteration a new random subset is drawn. An exponentially decaying stepsize γ_k is typically used:

$$\gamma_k = \frac{\delta a}{(A + k)^\alpha}, \tag{3}$$

where δ is the maximum voxel spacing among the x, y and z axes. Parameters a, A and α are constants that typically need to be tuned for the type of data. Since stochastic gradient descent does not have a trivial stopping criterion the maximum number of iterations $k_{\max}$ needs to be set as well.

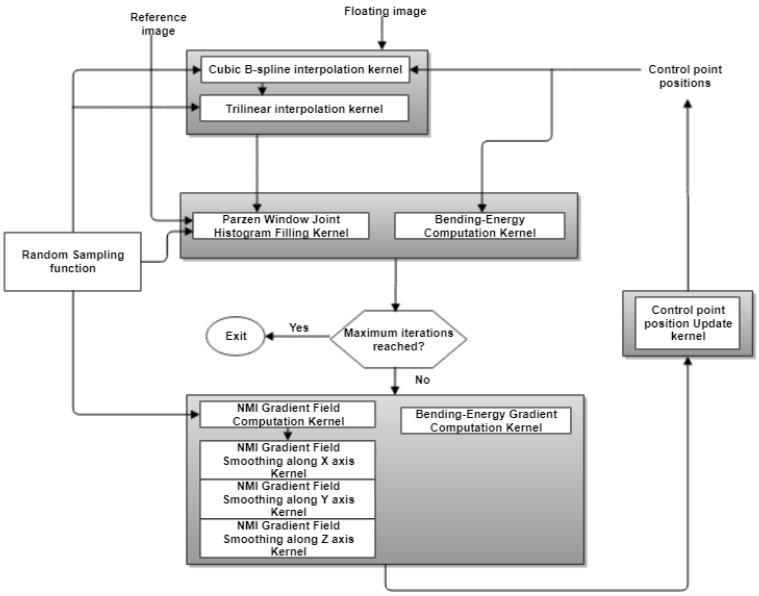

Figure 1: Block diagram of NiftyRegSGD, adding Random Sampling function to the original design of NiftyReg.

The architectural changes applied to NiftyReg allow for the implementation of a random sampler, enabling the calculation of the approximate gradient and replacing the line search strategy with a stopping criterion by the

decaying step size function using a fixed number of iterations. Figure 1 shows NiftyRegSGD architecture which is similar to the original NiftyReg.[4] We implemented a naive sampler that randomly picks samples from the fixed image. This randomness, however, prevents the GPU from accessing memory in parallel, forcing sequential global memory reads, which lead to higher memory access time and wastage of memory bandwidth. Therefore, we propose a new sampling strategy that is better tailored to GPU hardware, coined random chunk sampling. In this strategy, every first sample out of 32 samples is created randomly, followed by 31 samples adjacent to this first sample. This enables 32 threads on a GPU to have a coalesced memory access, which results in faster memory access and increased GPU throughput. Figure 2 shows the non-coalesced and coalesced memory access for the naive and chunk random sampler respectively. Left picture in Figure 2 shows the non-coalesced memory access for the naive sampler. When thread 0 in a GPU warp accesses a memory at address 0 from the global memory, the neighboring memory addresses (denoted by dots) are copied to the cache memory automatically. But, other threads in the same warp do not access these neighboring memory available in cache. Instead, these threads again access the global memory. Thus, there are 32 global memory accesses in a warp, which are slower than accessing the memory from cache. Using the random chunk sampling (right picture in Figure 2), there is only one global memory access. While, the other 31 threads will access the faster cache memory.

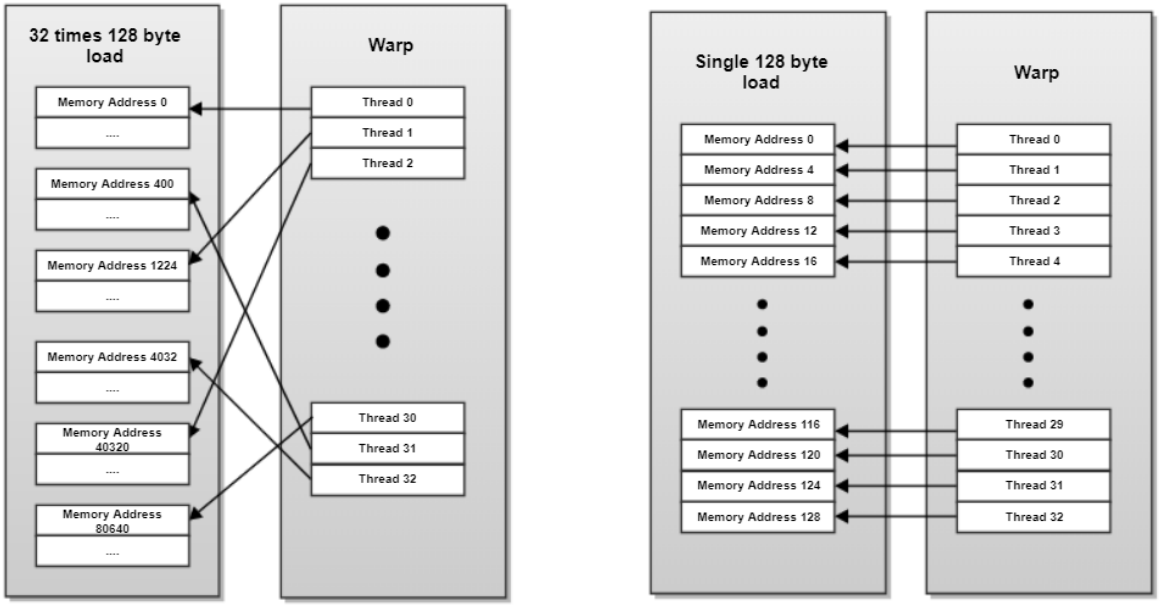

Figure 2: Non coalesced (left) and coalesced (right) memory access.

3. EXPERIMENTS AND RESULTS

The proposed NiftyRegSGD method with naive random sampling as well as with random chunk sampling is compared with `elastix`-FASGD and the original NiftyReg.

For the experiments, CT lung data from the SPREAD study[9] have been used. It consists of 19 patients with baseline (fixed) and follow-up (moving) images. Each patient has 100 corresponding landmarks that serve as a ground truth to evaluate the target registration error (TRE). The data was divided into a training set of 10 patients, to find the optimal registration settings, and a testing set of 9 patients to perform the final evaluation. All experiments were run on an Intel Xeon E5-1620 CPU with a Tesla K40c GPU.

Profiling the most time consuming GPU kernels, i.e. the B-spline transform and the resampler kernel, shows an improvement in the throughput by the random chunk sampler with respect to the the naive random sampler. In Figure 3, the GPU-throughputs are plotted for both samplers for different sampling percentages $s_{\%}$ and the different resolution levels of the registration procedure. For each sampling percentage and resolution level

the random chunk sampler has a better GPU-throughput than the naive random sampling for both the B-spline transform and resampler kernel. An important notion for this graph is that, although a higher sampling percentage might give a higher throughput, it still takes more time for one iteration in the registration procedure. And, although a higher sampling percentage improves the approximation of the gradient, this does not necessarily improve the convergence rate in such a way that the total registration time is smaller. Registration experiments were performed to assess this.

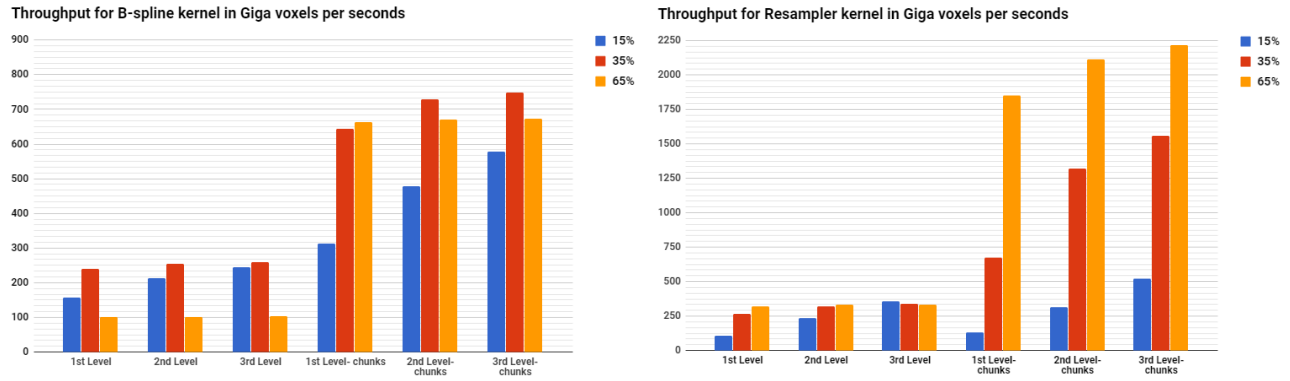

Figure 3: GPU-throughputs of the most time consuming kernels for different sampling percentages at different resolution levels of the registration procedure. The throughputs using the random chunk sampler are indicated with 'chunk' and those using the naive random sampler without.

The registration settings we used for `elastix`-FASGD where proposed by its authors[1] and were determined for the same SPREAD data set. To have a fair comparison based on speed we tuned NiftyRegSGD with the random chunk sampler to match the median TRE accuracy to that of `elastix`-FASGD. NiftyReg could not be tuned to achieve a sufficient median TRE, therefore we report using the default settings. For NiftyRegSGD we used the same B-spline grid spacing as NiftyReg. The values $\alpha = 0.90$ and $A = 20$ were chosen from the literature.[2] We optimized the parameters gain factor $a \in [0.05, 0.65]$, sampling percentage $s_{\%} \in [10, 80]$ and maximum number of iteration $k_{\max} \in [10, 300]$, over the training set. Keeping computation time in mind, the optimal value of a was found to be 0.25. For $a = 0.25$, optimal values for $s_{\%} = 15\%$ and $k_{\max} = 20$ were found, see Figure 4.

The accuracies of the four methods are compared in Figure 5 for both the training and the test set. For the training data, Wilcoxon signed rank tests indicated that there were no significant differences in median accuracies of both NiftyRegSGD methods with respect to `elastix`-FASGD ($p > 0.05$). For the test data, the differences with respect to `elastix`-FASGD were found to be just significant for the NiftyRegSGD with random chunk sampling and not for the NiftyRegSGD with naive random sampling. All settings are summarized in Table 1. The average timings of the four methods are plotted in figure 6.

	elastix-FASGD	**Original NiftyReg**	**NiftyRegSGD (both)**
Resolutions	3	3	3
Transform	B-spline	B-spline	B-spline
3D Parameters	$\approx 1k/6k/35k$	$\approx 6k/35k/230k$	$\approx 6k/35k/230k$
Metric	NMI	NMI	NMI
Optimizer	FASGD	Conjugate gradient	SGD
Step size	adaptive	line search	Equation (3)
Iterations	500/500/500	$\approx 54/64/169$	20/20/20
Sampler	random	full	random or random chunk
Samples	5000/5000/5000 (0.04%)	$\approx 4 \cdot 10^5/3 \cdot 10^6/2 \cdot 10^7$ (100%)	$\approx 6 \cdot 10^4/5 \cdot 10^5/4 \cdot 10^6$ (15%)

Table 1: Algorithmic and settings overview of the various registration methods.

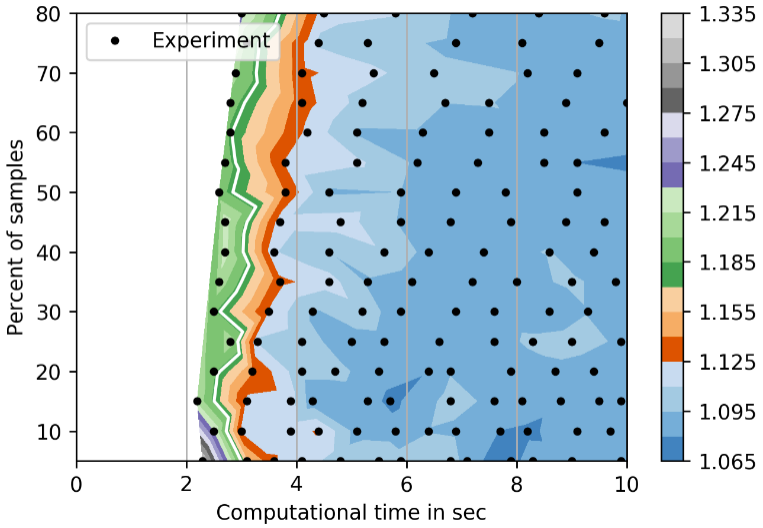

Figure 4: The median TRE for NiftyRegSGD with random chunk sampling is plotted against $s_{\%}$ in steps of 5% and $k_{\max}$ in steps of 10 for $a = 0.25$ for the training set. The background color indicates the median TRE. The white curve equals the median TRE of `elastix`-FASGD. The left-most point on this white curve indicates the fastest setting equally accurate as FASGD. The optimal settings are $s_{\%} = 15\%$ and $k_{\max} = 20$.

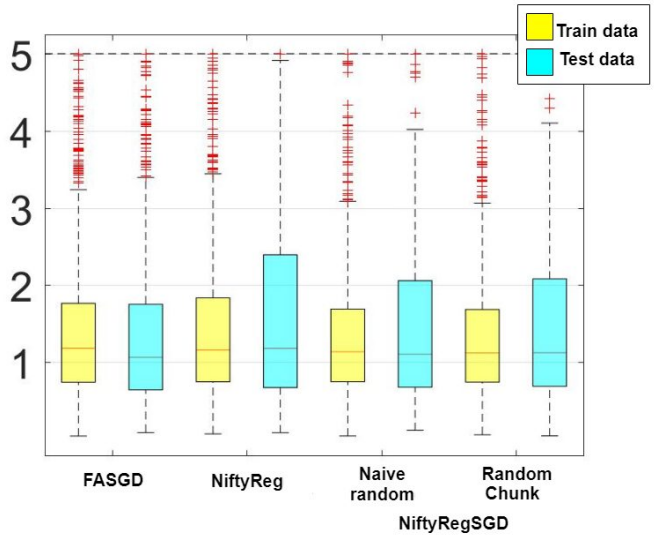

Figure 5: Target registration error (TRE) in mm using settings of Table 1 for the training and test data set. Proposed NiftyRegSGD with random chunk sampling performs fastest with better accuracy for the training data and less accuracy for the test data compared to `elastix`-FASGD.

4. DISCUSSION AND CONCLUSION

We presented our image registration method NiftyRegSGD which introduces stochastic gradient descent (SGD) optimization in a high performance GPU implementation. Experiments have been performed on follow-up lung CT scans. The use of SGD drastically speeds up registration time (to 2.8s) compared to the current GPU-based registration method (NiftyReg, 35.2s) that uses fully deterministic gradients. At an equal median target

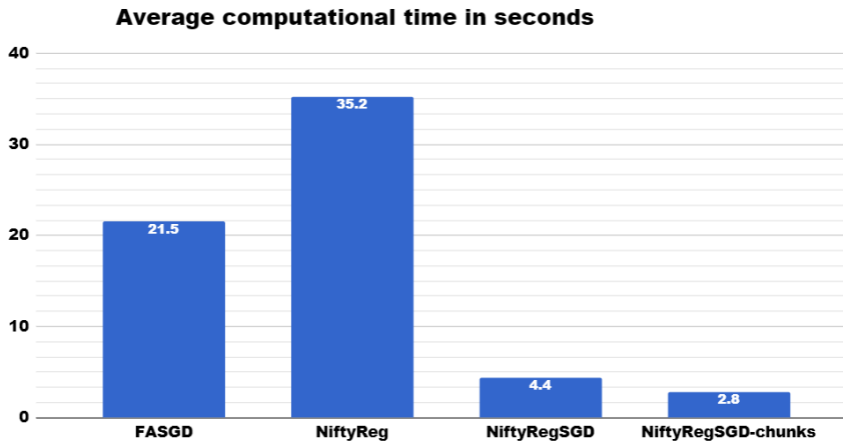

Figure 6: Run time of all methods, using an Intel Xeon E5-1620 CPU with a Tesla K40c GPU

registration error, our GPU implementation also outperforms elastix-FASGD (21.5s), to our knowledge currently the fastest method on this dataset.

We introduced a random chunk sampler as part of our SGD, which shows a speed improvement over a naive random sampler (4.4s). Due to its coalesce memory access, the throughput of GPU kernels is drastically increased. Sacrificing some of the randomness-properties only harmed the convergence rate a little (as observed in the test set), which might easily be compensated for by an extra iteration.

Experiments from Figure 4 have shown that the stochastic sub-sampling percentage of NiftyRegSGD was found to be optimal at 15%, in contrast to the CPU-based SGD of elastix-FASGD of around 0.04%. A smaller percentage of data reduces the computation time per iteration, but typically degrades the convergence rate. On the GPU, this trade-off favors a higher sampling percentage due to its parallel computation.

The small difference in accuracy for training and test data shows that accuracy depends on manual tuning, that is currently needed for NiftyRegSGD. For future work we may adopt the automatic parameter estimation methods from FASGD to remedy this. We may accelerate this estimation procedure on the GPU as well. The current results however still yield sub-voxel accuracy within 2.8s for a non-rigid registration procedure. We therefore conclude that the proposed methods open up possibilities to embed online registration in the clinical workflow, and consequently may benefit e.g. image-guided surgery and adaptive radiotherapy. Our fork of NiftyReg is publicly available at https://github.com/SuperElastix/NiftyRegSGD.

Acknowledgements

The Tesla K40c used for this research was donated by the NVIDIA Corporation.

REFERENCES

[1] Qiao, Y., van Lew, B., Lelieveldt, B. P. F., and Staring, M., "Fast automatic step size estimation for gradient descent optimization of image registration," *IEEE Transactions on Medical Imaging* **35**, 391–403 (Feb 2016).

[2] Klein, S., Pluim, J. P., Staring, M., and Viergever, M. A., "Adaptive stochastic gradient descent optimisation for image registration," *International journal of computer vision* **81**(3), 227 (2009).

[3] Robbins, H. and Monro, S., "A stochastic approximation method," *The annals of mathematical statistics* , 400–407 (1951).

[4] Modat, M., Ridgway, G. R., Taylor, Z. A., Lehmann, M., Barnes, J., Hawkes, D. J., Fox, N. C., and Ourselin, S., "Fast free-form deformation using graphics processing units," *Comput. Methods Prog. Biomed.* **98**, 278–284 (June 2010).

[5] Sharp, G. C., Kandasamy, N., Singh, H., and Folkert, M., "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," *Phys Med Biol* **52**, 5771–5783 (Oct 2007).

[6] Shamonin, D., Bron, E., Lelieveldt, B., Smits, M., Klein, S., and Staring, M., "Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease," *Frontiers in Neuroinformatics* **7**, 50 (2014).

[7] Klein, S., Staring, M., Murphy, K., Viergever, M. A., and Pluim, J. P., "Elastix: a toolbox for intensity-based medical image registration," *IEEE Transactions on Medical Imaging* **29**(1), 196–205 (2010).

[8] Zastrau, D. and Edelkamp, S., [*Stochastic Gradient Descent with GPGPU*], 193–204, Springer Berlin Heidelberg, Berlin, Heidelberg (2012).

[9] Stolk, J., Putter, H., Bakker, E. M., Shaker, S. B., Parr, D. G., Piitulainen, E., Russi, E. W., Grebski, E., Dirksen, A., Stockley, R. A., Reiber, J. H., and Stoel, B. C., "Progression parameters for emphysema: A clinical investigation," *Respiratory Medicine* **101**(9), 1924 – 1930 (2007).