Computation of Mutual Information Metric for Image Registration on Multiple GPUs

Andrew Adinetz[1], Jiri Kraus[2], Markus Axer[3], Marcel Huysegoms[3], Stefan Köhnen[3], and Dirk Pleiter[1]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
[2] NVIDIA GmbH, Germany
[3] Institute of Neuroscience and Medicine (INM-1), Forschungszentrum Jülich, 52425 Jülich, Germany

Abstract. Because of their computational power, GPUs are widely used in the field of image processing. Registration of brain images has already been successfully accelerated with GPUs, but registration of high-resolution human brain images presents new challenges due to large amounts of data and images not fitting in the memory of a single device.

In this paper, we address this issue with two approaches. The first approach replicates image data in system memory of each node and distributes only a part of the data over multiple GPUs. The second approach splits image data between multiple GPUs, and overlaps computation and communication to hide latency. For both approaches, we present a performance analysis and comparison.

1 Introduction

Unraveling the basic organization principles of the human brain represents one of the major challenges in modern neuroscience. The human brain's heterogeneous topology poses a particular challenge to any neuro-imaging technique and prevented the neuroscientists from understanding its complexity so far.

Therefore, the Institute of Neuroscience and Medicine (INM-1) focusses on the assembly of a realistic 3D model of the human brain [1, 2, 3, 4]. This brain model will be referred to as JuBrain [5]. The data set will not only help to reveal the neuro-biological basics of mental capacities, but will also enable characterization of their individual facets and underlying mechanisms. Recent developments in the field of post mortem microscopic neuro-imaging allow researchers to obtain information about the brain that were not available so far, but they also pushed the manageability of data sets to the limits.

The post mortem studies require brains to be cut into thousands of histological sections. Typical section thickness cover the range between 10 microns and 60 microns, which translates into 15.000 and 2.500 sections per human brain that need to be measured one by one. To construct a coherent 3D description (cf. Fig. 1), advanced image registration techniques have to be applied to all sections [6, 7, 8, 9]. However, the assembly of one cellular model of the human brain scanned with the resolution of 1 micron will add up to data sizes of 300 TB per

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 208–217, 2014.

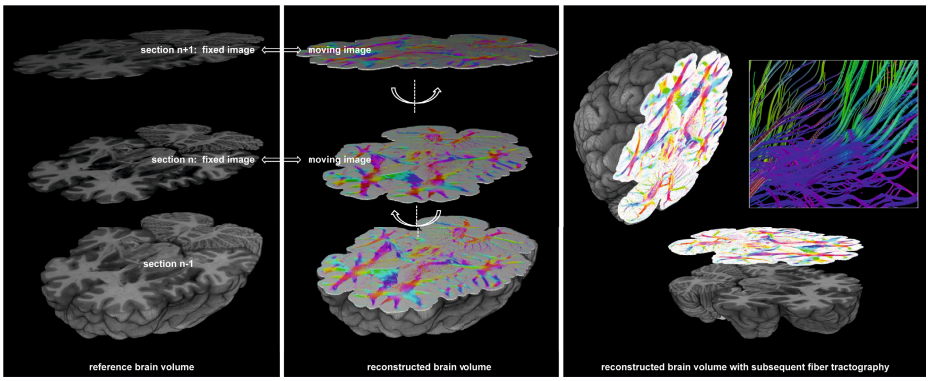
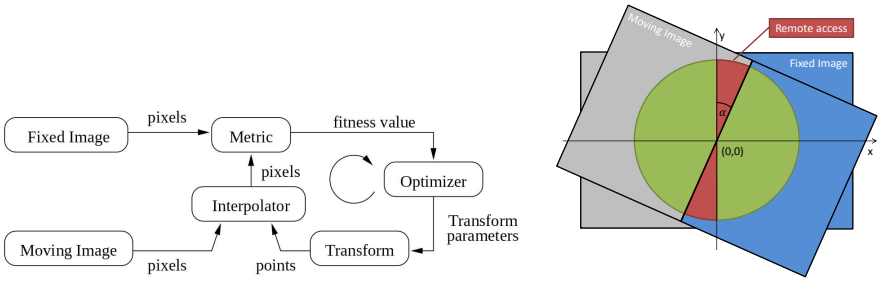


Fig. 1. Schematic of a 3D reconstruction of a human brain: Each registration approach requires a moving image (middle) that is being aligned with a fixed image (left)



(a) ITK work-flow [10]. (b) Benchmark setup.

Fig. 2. Image registration workflow and benchmark setup

brain considering only the imaging data. The fiber architecture in post mortem brains acquired with polarized light imaging [4] easily reach sizes of 500 GB for a single whole brain section. This means, to deal with the high resolution images and large data sets highly advanced computational concepts are required.

A typical image registration process is shown in Fig. 2a. The process is a feedback loop in which the *Metric* determines the fitness value by comparing the transformed moving image with the fixed image. While iterating the loop, the *Optimizer* selects the transformation parameter. Fig. 2b shows a simple registration problem, where only rotation is used as transformation.

The computationally most demanding step is the metric evaluation, where we use the mutual information metric. In this paper we make several contributions to solve the problem of computing the metric efficiently on multiple GPUs. Firstly, we discuss different strategies for evaluating mutual information metric on GPU clusters for large images. Secondly, we present a performance analysis of prototype implementations for different GPU architectures.

2 Related Work

A major problem of the existing image registration approaches is the time required for the calculations. Since image resolutions and file sizes are continually growing, new approaches to speed up image registration and other image processing algorithms, are needed. Consequently, there is need to take advantage of the high performance provided by modern GPU architectures. Previous studies could already demonstrate that the parallelization of specific registration approaches on the GPU allows fo significant speed ups [11, 12, 13]. Also surveys show that there is still a lot potential to reduce computation time and handle even larger data sets [14, 15].

3 Metric Evaluation

The mutual information metric determines the shared information between two images [6]. Shared information is defined via mutual dependence between the two images. To measure mutual dependence, a joint probability distribution is calculated. In the case of two images, this distribution is defined as a joint histogram. To measure the similarity from the joint histogram, the Shannon entropy [16] is used. The similarity is thus defined as the uncertainty of the joint distribution.

As metric evaluation is the most time-consuming part of the optimization, we offload it to the GPU. For sufficiently small images we can copy both fixed and moving images are copied to the memory of a single GPU at the start of the optimization procedure, and keep them there during the computations. The joint histogram is evaluated on the GPU using atomic operations. It turns out that on Fermi GPUs, atomics are the limiting factor. So we evaluate histogram of individual images from the joint histogram on the host, which reduces the number of atomics by a factor of 3.

We also experimented with the number of pixels processed by each GPU thread. And while for Fermi 1 pixel per thread gives the best results, for Kepler the best single-GPU performance is achieved when each thread processes a grid of 32×32 pixels.

4 Parallelization Strategies

Because high resolution images are too large to fit in device memory of a single GPU, we perform a 2D domain decomposition to use the memory of multiple GPUs. We use block distribution of both fixed and moving image, i.e. we divide both into n equally sized rectangular domains and assign each domain to one GPU. Each GPU is then responsible for calculating the contribution to the joint histogram of the domain it owns. The individual histograms are combined into a single histogram in the second step.

To calculate individual histogram contributions, each GPU reads pixels from the part of the fixed image it owns and applies the given transformation to each

of these. This will give coordinates of the moving image pixel for each fixed image pixel. Some of these coordinates fall into the locally stored part of the moving image, and the value of the moving image pixel can then be directly fetched. If the transformed pixel coordinates are stored on a remote GPU, the histogram contribution of that pixel has to be computed differently. We have developed two different approaches for evaluating contributions of remote pixels.

4.1 Replication in System Memory

In this approach we assume the system memory, which compared to the device memory has a much larger capacity, can hold the entire moving image. The device memory holds the fixed image and a local portion of the moving image. If the device needs to access the non-local portion of the image, it has to read from the (pinned) host memory. We call this approach *replication in system memory*, or just *system-memory approach*.

The advantages of this appoach are its simplicity and that it only requires a single kernel launch per GPU for processing. Its disadvantages are that accesses to the non-local parts of the moving image are very slow compared to device memory, and that the interpolation features of the texture unit cannot be used as in the single GPU case. Although the low precision texture interpolation on the moving image data is sufficient to calculate the mutual information metric, we faced accuracy issues when using software interpolation to the non local parts of the moving image and texture interpolation to the local part. We therefore use software interpolation only. Since the kernel is never limited by arithmetic throughput, this has only minor effects on the overall performance.

4.2 List-Update Approach

In another approach we distribute both fixed and moving images. When remote data is required to compute the joint histogram, the local fixed image data as well as the moving image coordinates are sent to the remote node that holds the required moving image data. The computations are performed on the remote node. This reduces communication and synchronization costs, as there is no need for a response message. Since the message size for each pixel is small, many messages are aggregated in a device memory buffer before being sent. Since in this approach for each remote node a list of updates is collected, we call it *list-update approach*.

Since the number of pixels processed inside a single kernel invocation is large, and potentially each pixel can send a message, the number of messages sent during the kernel execution cannot fit into a buffer of reasonable size. Flushing this buffer while the kernel is running to make place for new messages is currently not possible to implement reliably. We therefore decided to split the processed portion of the image into smaller-sized chunks, with one kernel invocation per chunk. As no more than one message is sent per pixel, the size of the chunk can be chosen in such a way that the buffer size is reasonable. Moreover, processing

multiple chunks in different streams allows us to overlap computation, i.e. processing pixels and handling list-update messages, with communication to and from GPU as well as between nodes.

We considered two ways of collecting list-update messages within the buffer. One is to allocate a chunk-sized buffer which holds a message for a particular pixel at a predefined position. "compression" kernel then groups messages by the destination node before sending. Alternatively, one can allocate one buffer per destination node and use atomic operations to fill these buffers contiguously. The compression approach has the obvious advantage of using less memory, while the atomic write-out approach does not require any additional passes before sending the buffers.

For both approaches, we apply additional optimizations. For the compression approach, a sent-flag was introduced which indicates whether a message had been sent to avoid unnecessary compressions. For atomic write-out, we exploit synchronous execution of warp threads, which allows doing only one atomic increment per warp per sender, increasing write-out throughput.

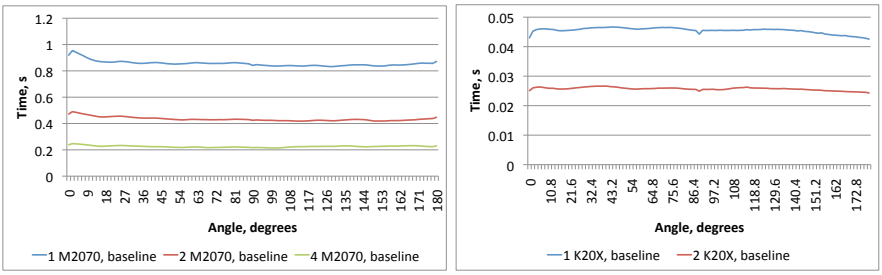
Also, for both approaches, AoS (array of structures) or SoA (structure of arrays) message buffer layouts can be used. And while AoS is simpler to implement, SoA can consume less memory per message (9 bytes vs. 12 bytes) since no alignment is required. We also experimented with the number of pixels processed by each thread.

5 Performance Evaluation and Discussion

The benchmark setup employed in our experiments is depicted in Fig. 2b. Both fixed and moving images have the same size. The centers of both are aligned. A circular mask centered at the center of the images is used, and its radius is chosen as 0.9 of the smaller image half-axis. The transformation applied to the moving image is rotation by angle α around the center. This simplified setup has the advantage that we can control the number of remote accesses via angle α.

We start 1 MPI process per GPU and investigate parallelization over up to 4 GPUs. Block distribution of the image among processes is applied, first along x axis, and then along y. The choice of transformation and the mask ensures that for a fixed image pixel inside the mask, the moving image pixel will also fall inside the mask, which reduces load imbalance. The fraction of pixels with remote access is $\beta = \min(\frac{\alpha N}{2\pi}, 1)$, where $N \geq 2$ is the total number of MPI processes. We typically measured the execution times for 101 different values of α with $0^\circ \leq \alpha \leq 180^\circ$ and report average values obtained from 5 measurements.

All experiments presented here were run at Jülich Supercomputing Centre (JSC) using up to 2 nodes of the JUDGE cluster equipped with Fermi GPUs, as well as a Kepler test system. Each node of the JUDGE cluster is equipped with 2 6-core 2-way hyper-threaded Intel X5650 CPUs running at 2.67 GHz, 96 GBytes RAM, and 2 NVIDIA M2070 GPUs with activated ECC and 5.25 GBytes of device memory. The test system comprised 2 8-core, 2-way hyper-threaded Intel E5-2650 CPUs running at 2 GHz, 64 GBytes RAM and 2 NVIDIA K20X GPUs with ECC enabled and 5.6 GBytes of device memory.



(a) M2070 (b) K20X

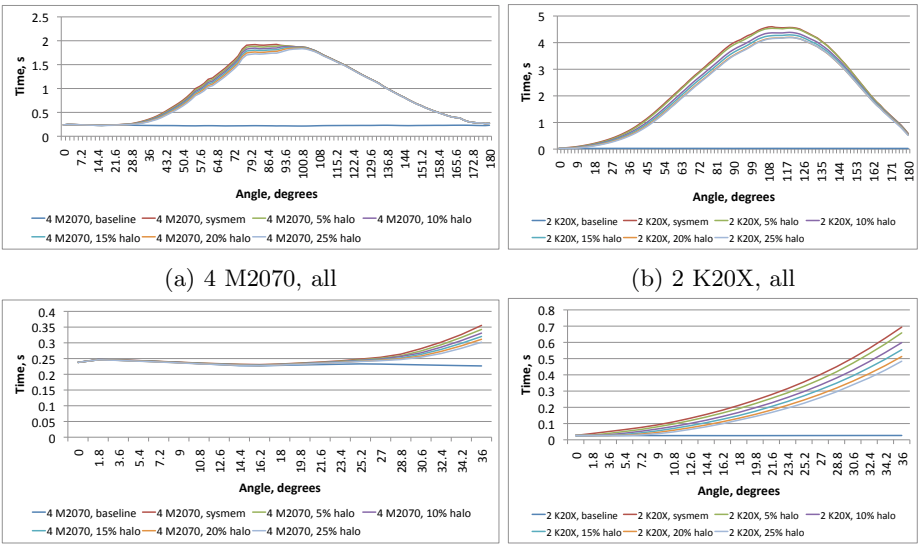
Fig. 3. Baseline performance

To obtain baseline performance, we replicated images to all GPUs such that parallel evaluation of the metric requires almost no communications. We used 1 and 32×32 pixels per thread in case of Fermi and Kepler, respectively. The results are presented in Fig. 3. As expected, the scaling is close to ideal, as only the final histogram reduction involves communication. It is also noticeable that due to improved atomics, performance on Kepler GPUs is more than $20\times$ higher than for Fermi.[1].

We then performed experiments with system memory replication approach on different GPUs. We also varied the percentage of GPU memory allocated for storing a moving image halo. The results for different GPU architectures are shown in Fig. 4. As Fermi performance is limited by atomics, a significant fraction of system memory accesses can be tolerated. When the rotation angle α becomes larger, the amount of data being read from system memory increases, and execution times get larger. In Fig. 5 we show both execution time and the number of system memory accesses as a function of α. The number of memory accesses does not only depend on the amount of read data but also on the access pattern. As is shown in Fig. 5b, for small angle α adjacent system memory addresses are read which allows coalesced accesses, i.e. a small amount of data is read efficiently. For $\alpha \simeq 90°$ this is not possible anymore and a result the number of memory accesses increases significantly, i.e. an increased amount of data is fetched in an inefficient way. For $\alpha > 90°$ the amount of data read from system memory increases but now accesses can again be coalesced and thus the number of access reduces.

As Kepler has much higher atomic throughput, the increasing fraction of system memory accesses causes a performance drop also for small angles α. When the coalescing is restored at $\alpha = 180°$, execution time is nevertheless 10-times larger compared to $\alpha = 0$ when all data is read from device memory. Experimentally we found that on Kepler using 4×4 pixels per thread results in best performance. For both GPUs, increasing the halo size provides some performance improvement, but the overall picture remains the same.

[1] Optimization is still on-going so performance improvements are possible for both.



(a) 4 M2070, all

(b) 2 K20X, all

(c) 4 M2070, close-up

(d) 2 K20X, close-up

Fig. 4. System memory replication performance with and without halo

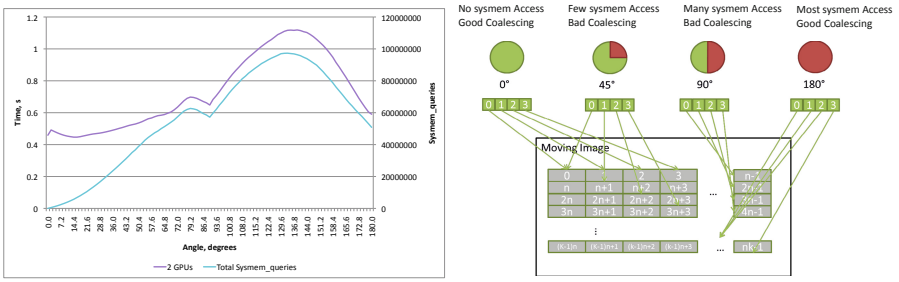
Tests based on the list-update approach were run with 4 streams, a chunk size of 1536×1536, 1 pixel per thread on Fermi and 4×2 for Kepler. We first compared array-of-structures (AoS) and structure-of-arrays (SoA) approaches, and compared atomic and compression write-out. Note that due to the benchmark setup, processing chunks in the same order in all processes will lead to high load imbalance. To lower imbalance, we use a "reflected" order of chunk processing.

Results are presented in Fig. 6. SoA is always better than AoS, since it allows reducing the amount of data transferred. On Kepler message buffer compression results in lower performance, while for Fermi we observe similar performance for either using buffer compression or atomic buffer updates. The latter is consistent with the observation that atomic operations are a bottleneck on Fermi.

We also did performance tracing of our list-update implementation. It turns out that the main limiting factor, at least for Fermi GPUs, is not communication but load imbalance between chunks. It is partially caused by the sequence of atomic operations, making performance data-dependent. Furthermore, the numbers of list-update messages per chunk are different.

5.1 Comparison of Parallelization Approaches

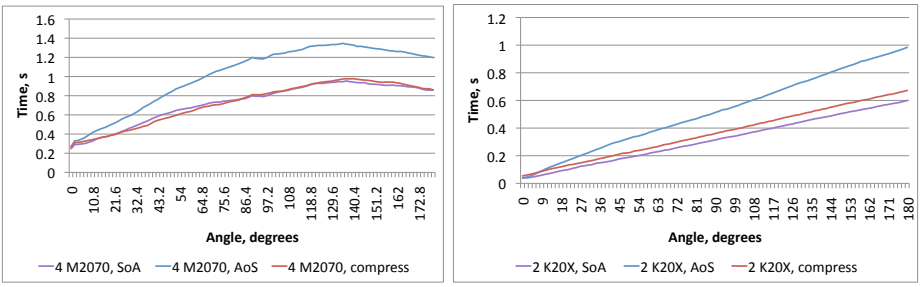
Comparison of performance of both parallelization approaches is presented in Fig. 7. For Fermi GPUs, as using system memory does not compromise performance, system-memory approach is clearly superior for small angle values. As we expect coalescing to be better in real applications, we actually expect the system-memory approach to be better even when the fraction of remote accesses is high.



(a) Execution time and number of system memory accesses using 2 M2070 GPUs.

(b) System memory access patterns for different angle α.

Fig. 5. Performance analysis for the system-memory approach



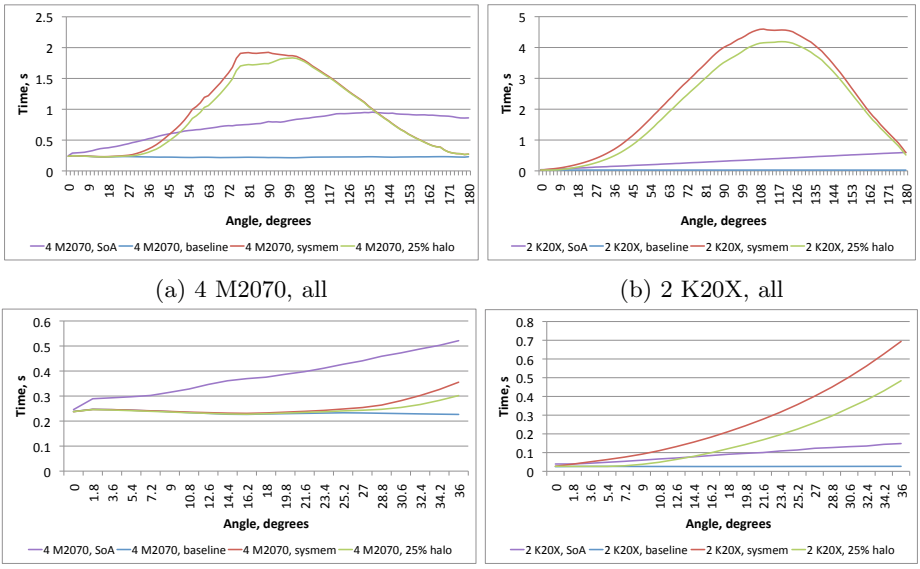
(a) 4 M2070

(b) 2 K20X

Fig. 6. Performance of list-update variants

The situation is quite different for Kepler. System memory accesses are no longer cheap compared to atomics, and the list-update approach provides clearly better execution times, which scale linearly with the fraction of remote accesses, and does not depend on coalescing. And while we expect system-memory approach to do better for real applications, list-update approach will still have higher performance.

There are other concerns in choosing the preferred approach, apart from performance. As we expect major changes of the application to continue, application developer productivity is a major concern. System-memory approach has a clear advantage here, as it required significant less effort to implement (about 50 lines of code, 1 person-day) compared to the list-update approach (about 1000 lines of code, 2 person-weeks).



(a) 4 M2070, all

(b) 2 K20X, all

(c) 4 M2070, close-up

(d) 2 K20X, close-up

Fig. 7. Performance of different parallelization approaches

6 Conclusions

In this paper, we presented two approaches to compute the mutual information metric for large images on multiple GPUs. The first approach keeps a full copy of the moving image in system memory, which is much larger than device memory, and only partitions the data hold in device memory. The second approach partitions all image data among multiple GPUs, aggregates remote accesses and overlaps computation and communication to hide latencies. For both approaches, we discussed a number of optimizations and provided a performance analysis.

Surprisingly, it is hard to find a winner. On the one hand, the system-memory approach is clearly superior on Fermi GPUs, but it may suffer from non-coalesced system memory accesses. It does not show load imbalance issues and is clearly simpler in terms of implementation. On the other hand, the list-update approach is better on K20X GPUs but is more complicated. Its performance is mostly independent of the degree of coalescing. In case where more than 2 images needs to be processed, this can also be a better approach.

Acknowledgements. This work was performed in the scope of NVIDIA Application Lab at Jülich which is jointly funded by Forschungszentrum Jülich and NVIDIA.

References

[1] Toga, A.W., Thompson, P.M., Mori, S., Amunts, K., Zilles, K.: Towards multimodal atlases of the human brain. Nature Reviews Neuroscience 7(12), 952–966 (2006)

[2] Zilles, K., Schleicher, A., Palomero-Gallagher, N., Amunts, K.: Quantitative analysis of cyto-and receptor architecture of the human brain. Brain Mapping: The Methods 2, 573–602 (2002)

[3] Zilles, K., Amunts, K.: Segregation and wiring in the brain. Science 335(6076), 1582–1584 (2012)

[4] Axer, M., Amunts, K., Grässel, D., Palm, C., Dammers, J., Axer, H., Pietrzyk, U., Zilles, K.: A novel approach to the human connectome: ultra-high resolution mapping of fiber tracts in the brain. Neuroimage 54(2), 1091–1101 (2011)

[5] `http://www.jubrain.fzjuelich.de`

[6] Modersitzki, J.: Numerical methods for image registration. Oxford University Press, USA (2004)

[7] Studholme, C., Hill, D.L., Hawkes, D.J., et al.: An overlap invariant entropy measure of 3D medical image alignment. Pattern Recognition 32(1), 71–86 (1999)

[8] Palm, C., Axer, M., Gräßel, D., Dammers, J., Lindemeyer, J., Zilles, K., Pietrzyk, U., Amunts, K.: Towards ultra-high resolution fibre tract mapping of the human brain–registration of polarised light images and reorientation of fibre vectors. Frontiers in Human Neuroscience 4 (2010)

[9] Eiben, B., Palm, C., Pietrzyk, U., Davatzikos, C., Amunts, K.: Perspective error correction using registration for blockface volume reconstruction of serial histological sections of the human brain. In: Bildverarbeitung für die Medizin, pp. 301–305 (2010)

[10] Ibanez, L., Schroeder, W., Ng, L., Cates, J.: The ITK Software Guide, 1st edn. Kitware, Inc. (2003), `http://www.itk.org/ItkSoftwareGuide.pdf`, ISBN 1-930934-10-6

[11] Modat, M., Ridgway, G.R., Taylor, Z.A., Lehmann, M., Barnes, J., Hawkes, D.J., Fox, N.C., Ourselin, S.: Fast free-form deformation using graphics processing units. Computer Methods and Programs in Biomedicine 98(3), 278–284 (2010)

[12] Köhn, A., Drexl, J., Ritter, F., König, M., Peitgen, H.O.: GPU accelerated image registration in two and three dimensions. In: Bildverarbeitung für die Medizin 2006, pp. 261–265. Springer (2006)

[13] Köhnen, S., Ehrhardt, J., Schmidt-Richberg, A., Handels, H.: CUDA Optimierung von nicht-linearer oberflächen-und intensitätsbasierter Registrierung. In: Bildverarbeitung für die Medizin 2011, pp. 99–103. Springer (2011)

[14] Shams, R., Sadeghi, P., Kennedy, R., Hartley, R.: A survey of medical image registration on multicore and the GPU. IEEE Signal Processing Magazine 27(2), 50–60 (2010)

[15] Eklund, A., Dufort, P., Forsberg, D., LaConte, S.M.: Medical image processing on the GPU - past, present and future. Medical Image Analysis (2013)

[16] Shannon, C.E.: A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review 5(1), 3–55 (2001)