Efficient Computation of Joint Histograms and Normalized Mutual Information on CUDA Compatible Devices

Christian Ledig[1,2] and Christophe Chefd'hotel[1]

[1] Siemens Corporate Research, Princeton, NJ, U.S.A.
[2] Friedrich-Alexander University, Erlangen-Nuremberg, Germany

Abstract. We present new strategies for a highly optimized joint histogram computation of large datasets on NVIDIA's *compute unified device architecture* (CUDA) compatible *graphics processor units* (GPUs). By applying novel techniques to an algorithm proposed in [1] and adapting it for joint histogram computation, a very efficient calculation of the Normalized Mutual Information (NMI) is possible. This similarity measure plays a major role in multimodal image registration. Since the computation time of NMI is dominated by the calculation of the joint histogram, the availability of a fast histogram computation is a key factor in designing computationally efficient registration methods. We show that the applied optimizations result in a reduction of computation time of up to 35% and of required memory by 50% for the calculation of NMI compared to previously published GPU implementations. We also demonstrate the impact of high performance joint histogram computation on practical registration tasks.

1 Introduction

The alignment of two images from different modalities is a fundamental task in a variety of applications in interventional and diagnostic imaging.

For example, registration methods are used to combine functional Positron Emission Tomography (PET) data with morphological Computed Tomography (CT) images in oncology and cancer staging [2].

Image registration can be described as the process of estimating a geometric transformation (rigid or nonrigid) that aligns a "moving" image to a fixed "reference" image. This task can be formulated as an optimization problem with respect to a given cost function which measures the quality of the alignment. The selection of the transformation model and the cost function is application dependent and has a strong impact on the registration results as well as the computation time.

For multimodal image registration tasks, cost functions based on statistical similarity measures such as Mutual Information (MI) [2] and Normalized Mutual Information (NMI) [3] have been proposed. These metrics are derived from information theory, and are typically computed from joint histograms of image intensity pairs.

Many clinical applications require fast, robust and highly accurate registration methods. When statistical similarity measures are used, it is then critical to optimize histogram calculations.

One widely pursued approach to efficient (joint) histogram computation is to employ GPUs. Although histogram computation is simple and trivial to implement on CPU, its efficient parallelization on a GPU [4] is a challenging task. As discussed in [4], previously proposed approaches based on shader programming rely on expensive preprocessing steps, such as sorting pixels by intensity value. With NVIDIA CUDA [5] a new general purpose parallel computing architecture was introduced in 2006. Its shared memory capability enables efficient solutions on the GPU for more complex problems. Algorithms for histogram computation on CUDA compatible devices were presented in [1], [4], [6], [7], [8], [9] and [10].

In [4], a very efficient 64 bin histogram computation was proposed and extended to 256 bins by simulating atomic intra-warp updates to the shared memory. In order to compute thousands of bins, this update mechanism was also used in the first method presented in [1] (labeled *Method1*). This algorithm lacks data independence, but it has the key advantage of low memory requirement and very good performance on real data. To overcome data dependence, the second method described in [1] (*Method2*) ensures collision free updates. This method requires the allocation of a full histogram for every thread in the global memory, which results in additional memory requirement. Dependent on the data, it also underperforms *Method1* for high bin ranges [1], which often occur in joint histogram computations.

A new algorithm – named "sort and count" – was recently proposed in [6]. This algorithm possesses very good scaling with high bin numbers. For reasonable joint histogram sizes around 75×75 bins this approach performs similar to *Method1*.

A self-optimizing histogram algorithm was introduced in [7]. Because of the very expensive preprocessing time of several seconds, this data dependent optimization does not meet our needs. In [8] an algorithm which sorts the image by intensity values was presented. With sorted data, faster histogram computations are possible. However, in addition to high memory requirements because of additional stored coordinates, the expensive preprocessing step (sorting) is a major drawback.

In addition to these exact computations, very fast algorithms were proposed in [9] and [10] to approximate histograms and MI. Our goal in this paper is to use CUDA to provide an exact calculation that combines high accuracy with significant speed ups.

In this paper, we present three new optimization strategies for histogram computation in Section 2. We then present experimental results and evaluate the impact of our novel optimizations on practical registration tasks in Section 3. We conclude in Section 4.

2 Method

2.1 Problem Formulation

The choice of a suitable similarity measure is a critical part in image registration. The measure MI, proposed in [2], is particularly robust and data independent. This metric does not require a strong a priori knowledge on the content of the registered images. MI can be defined as

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \tag{1}$$

where $H(X) = -\sum_x p_X(x) \log p_X(x)$ is the marginal entropy and
$H(X,Y) = -\sum_{x,y} p_{XY}(x,y) \log p_{XY}(x,y)$ is the joint entropy.

This metric captures the statistical dependence between the intensity values of an image pair. A normalized version of MI, called NMI, was introduced in [3]. It tends to be more robust to variations of the overlap region between images and is defined as

$$NMI(X,Y) = \frac{H(X) + H(Y)}{H(X,Y)} \ . \tag{2}$$

To evaluate Eq. 1 or 2 an estimation of the marginal probability distributions $p_X(x)$ and $p_Y(y)$ and the joint probability distribution $p_{XY}(x,y)$ is necessary. This is done by computing a joint histogram of the intensity pairs.

For two normalized images $I_\mathrm{r}(\cdot)$ (reference) and $I_\mathrm{m}(\cdot)$ (moving) with intensity values in $[0.0, 1.0]$ the joint histogram with $\mathrm{B_r} \times \mathrm{B_m}$ bins can be described as:

$$J(i,j) = \sum_{x \in I_\mathrm{r}} \delta(x) \tag{3}$$

$$\text{with} \qquad \delta(x) = \begin{cases} 1, \text{ if } I_\mathrm{r}(x) \in [\frac{i}{\mathrm{B_r}}, \frac{i+1}{\mathrm{B_r}}] \text{ and } I_\mathrm{m}(\phi(x)) \in [\frac{j}{\mathrm{B_m}}, \frac{j+1}{\mathrm{B_m}}] \\ 0, \text{ otherwise} \end{cases}$$

where $\phi(x)$ is a geometric transformation applied to the moving image.

As shown in [9], joint histogram computation of $\mathrm{B_r} \times \mathrm{B_m}$ bins is equivalent to the computation of a single histogram with $\mathrm{B_j} = \mathrm{B_r B_m}$ bins on I_c by combining the intensity values of the two original images such that,

$$I_\mathrm{c}(\phi(\cdot), x) = \frac{\mathrm{B_m}((\mathrm{B_r} - 1) I_\mathrm{r}(x) + I_\mathrm{m}(\phi(x)))}{\mathrm{B_j} - 1}, \mathrm{B_j} > 1 \ . \tag{4}$$

This new image, usually created in a preprocessing step, requires additional memory. This can be problematic for the registration of large datasets.

2.2 Base Algorithm

Method1 discussed in [1] is a very efficient histogram computation method. There are neither limitations of histogram bins nor a loss of accuracy by approximation. In this paper we describe new optimization techniques for this algorithm.

When parallelizing histogram computations on GPU, the main bottleneck is that different threads might compete by updating the same histogram bin. Even if there is in general no "mutex" mechanism available in CUDA[3], a solution was presented in [4] and employed in *Method1* for an efficient parallel histogram computation.

This approach is based on the fact that 32 threads (one warp) execute the same instructions in parallel. Within a warp, an atomic update of a particular histogram bin can be simulated by tagging data with the ID of the updating thread. If more than one thread update this data, exactly one thread will succeed and data tagged with this thread ID will remain. Thus it is possible to check which thread actually succeeded and redo the update for all not successful threads [4, 5]. This is why this algorithm is data dependent and performs worse for images with constant intensity values, where all threads of a warp collide while updating the same bin and hence need to be serialized.

Since global memory is slow, shared memory is used to hold temporary histograms. In *Method1*, an unsigned integer is used in shared memory to hold the tag in the 5 ($log_2 32$) most significant bits and the bin counter in the remaining 27 bits for each bin. The limitations for this approach arise because shared memory is limited to 16 kB per block and in order to occupy the GPU, several warps need to be run within one execution block. As a result, a partial joint histogram has to be allocated in shared memory for every warp.

In order to apply this algorithm to joint histogram computation, a new image is first created according to Eq. 4. This operation is expensive since it increases the memory requirement by a full image size. If no additional memory is available Eq. 4 has to be evaluated within each kernel call.

The computation of a reasonable joint histogram of size 80×80 with 4 warps (128 threads) per block would exceed the available shared memory by far (with a memory requirement of 100 kB $= 4$ warps $\times$ 6400 bins $\times$ 4 Bytes). For this reason, joint histograms usually cannot be computed with one kernel call. In fact, several kernel calls are necessary to compute step by step the complete joint histogram by computing a certain range of bins within each call.

In the following we propose to extend this algorithm with three major optimizations, which focus on execution time as well as on memory requirement.

2.3 Multiple Bin Encoding (MBE)

In order to use the shared memory more efficiently, we encode two bins in 26 of the 27 less significant bits of an unsigned integer. This concept is shown in

[3] As described in [5], this functionality depends on the *compute capability* of the device. Potential improvements based on atomic operations remain to be investigated.

Fig. 1. Of course this makes collisions more likely to happen, since even updates to two different bins may conflict because they are encoded in the same integer. Nevertheless *MBE* allows processing twice the number of bins per kernel call. This results in fewer overall kernel calls and reduces global memory access. It is especially important for complex interpolation schemes when no spare memory is available. With this optimization the data of a bin is encoded with 13 bits. Overflows are avoided by regularly updating the block histograms in the global memory. This optimization is always applicable and has no influence on memory requirements.

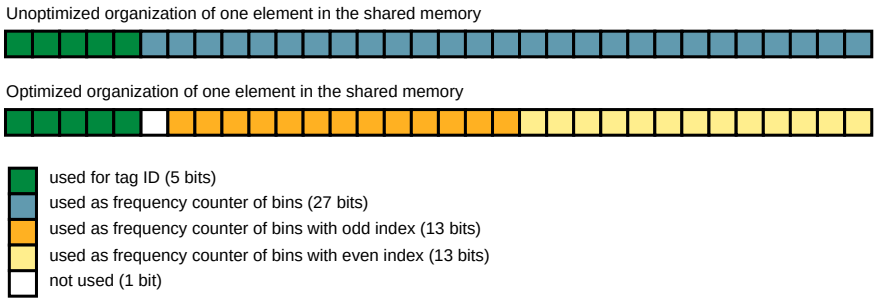

Fig. 1. Example of MBE using an unsigned integer to encode two bins instead of one. Bin1 and Bin2 are encoded in the first word, Bin3 and Bin4 in the second word, etc.

2.4 Bin Caching (BC)

If spare global memory is available, this optimization should be used in addition to *MBE*. Instead of performing a preprocessing step to create I_{c}, we use the first kernel call to compute the resulting bin number for a certain voxel pair. This is important since we avoid an unnecessary reread of the preprocessed image. It is also more efficient to store the resulting bin number instead of a combined intensity value. During the first kernel call the resulting bin number $[\mathrm{B_m}((\mathrm{B_r} - 1)I_{\mathrm{r}}(x) + I_{\mathrm{m}}(\phi(x)))]$ is stored to additional global memory. Joint histograms of more than 256×256 bins are of limited interest. In this case a resulting bin will never exceed 16 bits. We propose to pack two bin numbers into an unsigned integer which saves 50% of the additional required memory compared to the commonly used preprocessing. This results not just in an optimization in terms of memory demand, but also in a reduced global memory access. Subsequent kernels just read from the global memory the bin of the joint histogram that needs to be incremented.

This is a very valuable optimization, especially in cases where several kernel calls are necessary or expensive interpolation schemes are used. However, this approach requires additional memory of 50% of an image size. This is an improvement compared to the regular preprocessing step, but still too much

to claim general applicability for joint histogram computation. See Fig. 2 for a schematic flowchart of this optimization.

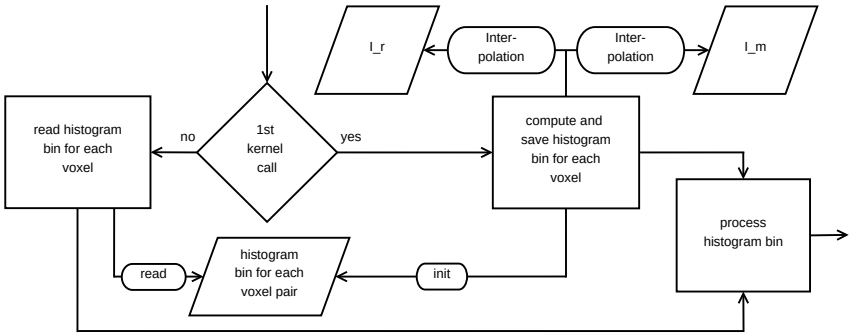

Fig. 2. Schematic flowchart of the kernel optimized with BC

2.5 Smart Texture Lookup (STL)

When there is not enough global memory temporarily available to apply *BC* or to use the regular preprocessing according to Eq. 4 the resulting bin number $[\mathrm{B_m}((\mathrm{B_r}-1)I_\mathrm{r}(x)+I_\mathrm{m}(\phi(x)))]$ needs to be computed within each kernel call. For this case we introduce an important optimization in addition to *MBE*. As already mentioned, the computation of a joint histogram with $\mathrm{B_r}\times\mathrm{B_m}$ bins is equivalent to the computation of a histogram of size $\mathrm{B_j}=\mathrm{B_r}\mathrm{B_m}$.

Whether a kernel call processes a certain voxel pair depends on the intensity values of the voxels in I_r and I_m. Often the intensity value in I_r is sufficient to rule out a voxel pair in the kernel call. Therefore the texture lookup in I_m needs to be skipped to avoid unnecessary and expensive texture lookups and interpolations in the global memory.

Within the k-th kernel call (each kernel computes $\mathrm{B_{kernel}}$ bins) we only consider combinations of intensity values for which the following equation holds:

$$(k-1)\mathrm{B_{kernel}} \leq \mathrm{B_m}((\mathrm{B_r}-1)I_\mathrm{r}(x)+I_\mathrm{m}(\phi(x))) < k\mathrm{B_{kernel}} \tag{5}$$

By reformulating Eq. 5 and the fact that $0.0 \leq I_\mathrm{m}(\phi(x)) \leq 1.0$ we get

$$(k-1)\mathrm{B_{kernel}} - \mathrm{B_m} \leq \mathrm{B_m}(\mathrm{B_r}-1)I_\mathrm{r}(x) < k\mathrm{B_{kernel}}\ . \tag{6}$$

If Eq. 6 is not true for a $I_\mathrm{r}(x)$, the texture lookup of $I_\mathrm{m}(\phi(x))$ is skipped.

3 Experiments & Benchmarks

For the experiments we used two different systems that are shown in Tab. 1. In 3.1 we generated and used data with either constant (HOMO_DATA) or uniformly distributed random (RAND_DATA) intensity values of different resolutions. We used different interpolation schemes (trilinear or tricubic) and applied

an affine transformation to the moving volume. We then compared the performance of (I): *MBE & BC* to *Method1* using a preprocessing step for Eq. 4 and (II): *MBE & STL* to *Method1* without the use of additional memory.
Note that computation times depend on the number of warps as observed in [1]. Here, 6 warps were used for comparison (I) and 4 warps for comparison (II).

In 3.2 we employed standard optimization schemes to tackle the registration tasks shown in Tab. 2.

Table 1. Test Systems

	System A	System B
CPU	2 Intel Xeon@3,2 GHz; 3 GB	2 Intel Xeon QuadCore@2,5 Ghz; 3 GB
GPU	NVIDIA GeForce 8800GTX; 768 MB	NVIDIA Tesla C1060; 4096 MB

Table 2. Registration tasks (REG), reference (R) and moving (M) images

Task	REG_BRAIN		REG_LUNG128	REG_LUNG256
	R_BRAIN	M_BRAIN	R/M_LUNG128	R/M_LUNG256
Resolution	$44 \times 512 \times 512$	$52 \times 256 \times 256$	$128 \times 128 \times 128$	$256 \times 256 \times 256$
Modality	CT	MRI (T1)	CT	CT
Remark	from the Vanderbilt database		lung in inspiration/expiration (R/M)	

3.1 NMI Computation Times

We calculate NMI in three steps: joint histogram computation, normalization of the histogram ($\sum_{i,j} J(i,j) = 1$) and the actual computation of NMI by Eq. 2. For the computation of the marginal entropies $H(X)$ and $H(Y)$ an adapted implementation of NVIDIA's high efficient reduction algorithm [11] is employed. The normalization together with the computation of NMI takes 0.19/0.14 ms (on System A/B resp.) for a joint histogram with 100×100 bins. Its cost is negligible compared to the joint histogram calculation.

Figure 3 illustrates the improvements achieved with the proposed optimizations for different bin numbers on the REG_BRAIN dataset. Table 3 shows the average reduction in computation time.

In comparison (I) on System B, we observed a significant reduction in runtime of up to 35% for real images from the Vanderbilt database [12].

Experiments on REG_DATA revealed that for bin ranges above 140×140 the optimized algorithm with *tricubic* interpolation is faster than the unoptimized algorithm with *trilinear* interpolation (if additional memory is available). This could allow for a more accurate registration by using higher order interpolation schemes *while* keeping computation times reasonable.

With *MBE*, due to its higher computational complexity and more frequent global memory updates (to avoid overflows), no further improvements were observed by encoding more than two bins in the 27 less significant bits.

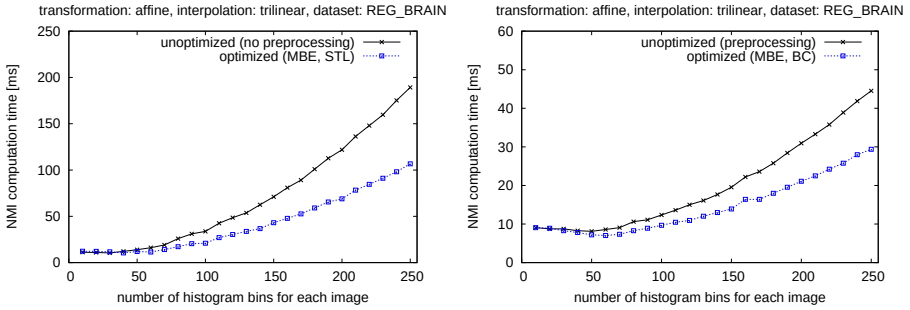

Fig. 3. NMI computation times on System B using different methods on dataset REG_BRAIN without (left) and with (right) additional memory available.

Table 3. Comparison of NMI computation times with optimized/unoptimized algorithms on RAND_DATA/HOMO_DATA and realistic REG_BRAIN datasets. Average computation time reduction on System A/B. Joint histogram size: bins×bins, Transformation: affine

	MBE & BC, (I)		MBE & STL, (II)	
	(trilinear)	(tricubic)	(trilinear)	(tricubic)
side length of volumes: 256 bins: 10 to 250, RAND_DATA	19.2/24.4%	15.4/28.5%	28.8/28.6%	40.9/41.9%
side length of volumes: 16 to 256 bins: 100, RAND_DATA	19.8/21.7%	14.1/24.1%	35.1/32.8%	44.7/43.5%
side length of volumes: 16 to 256 bins: 100, HOMO_DATA	10.9/13.6%	10.7/19.6%	33.0/31.6%	53.0/52.1%
REG_BRAIN datasets bins: 10 to 250	17.3/22.7%	16.0/28.3%	31.4/30.9%	45.9/46.6%

3.2 Applications to Rigid and Nonrigid Registration Tasks

Now we demonstrate the impact of the presented results on practical registration tasks. In addition to similarity measures, we also implemented all the building blocks of two registration algorithms (rigid and nonrigid) with CUDA. Their performance was evaluated on System B.

For *rigid registration* tasks we employ NMI together with a simple Hill Climbing algorithm to register multimodal head scans.

The *nonrigid registration* method is based on an algorithm proposed in [13]. The registration process is driven by forces derived from MI. A fast recursive Gaussian filter is used as regularizer. This method is tested with two CT lung datasets in different breathing phases.

The registration results are shown in Fig. 4. Computation times are listed in Tab. 4. We achieved a very low registration time with *MBE* & *BC* compared to the CPU implementation. Even if optimizing with *STL* instead of *BC* performs notably slower for the rigid registration task, it is an important optimization when no spare memory is available. This effect is not as noticeable for the nonrigid registration task because of the significant cost of the filtering operations.

The CPU code is highly optimized and parallelized. The GPU code is optimized by applying basic techniques presented in [14] and using either *MBE* & *BC* or *MBE* & *STL* for the joint histogram computation (100×100 bins).

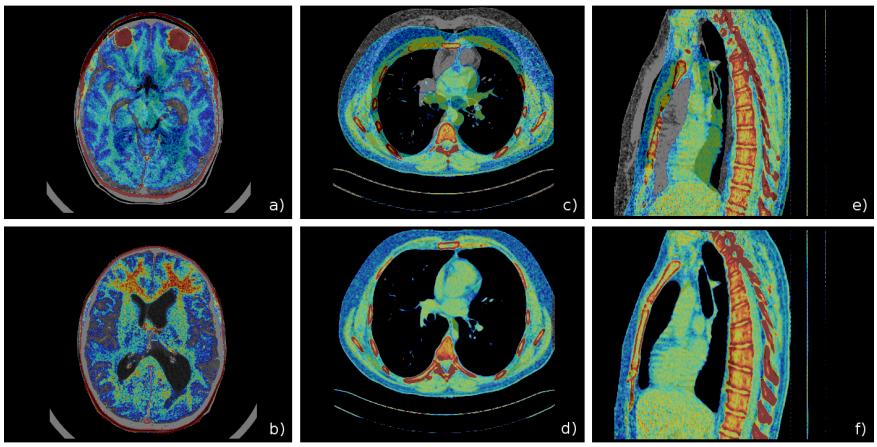

Fig. 4. REG_BRAIN unregistered: a and registered (rigid): b; REG_LUNG256 unregistered: c, e and registered (nonrigid): d, f

Table 4. Rigid and nonrigid registration performance on System B

	rigid		nonrigid	
	REG_BRAIN		REG_LUNG128	REG_LUNG256
	(trilinear)	(tricubic)	(trilinear)	
CPU	1.5s	15.9s	9.3s	127.4s
GPU (MBE & BC)	1.0s	1.6s	2.8s	23.1s
GPU (MBE & STL)	1.6s	3.2s	3.1s	25.0s

4 Conclusion

We presented three novel optimization strategies which allow for faster computation of joint histograms, even if higher order interpolation schemes are used or memory is limited. While *Bin Caching* requires slightly more memory, optimizations based on *Multiple Bin Encoding* and *Smart Texture Lookup* are always possible. The combination of *MBE* and *BC* allows for up to a 35% decrease in computation time with a reduced memory overhead. Even if *STL* is significantly slower than *BC*, this optimization can still be useful when no additional memory is available. None of the proposed optimization techniques require preprocessing steps. The performance gains observed in our experiments open the door to new applications for advanced registration methods, such as solving alignment problems in interventional imaging under real-time constraints.

References

1. R. Shams and R. A. Kennedy, Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices, Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS), Gold Coast, Australia, Dec. 2007, pp. 418–422.
2. F. Maes, A. Collignon, D. Vandermeulen, P. Suetens and G. Marchal, Multimodality Image Registration by Maximization of Mutual Information, IEEE Trans. Medical Imaging, 16(2), Apr. 1997, pp. 187–198
3. C. Studholme, D. L. G. Hill and D. J. Hawkes, An overlap invariant entropy measure of 3D medical image alignment, Pattern Recognition, 32(1), 1999, pp. 71–86
4. V. Podlozhnyuk, Histogram calculation in CUDA, NVIDIA Tech. Rep., 2007
5. NVIDIA, CUDA Programming Guide, 2010
6. R. Shams, et al., Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images, Comput. Methods Programs Biomed., 2009
7. T. Brosch and R. Tam, A Self-Optimizing Histogram Algorithm for Graphics Card Accelerated Image Registration, Proc. MICCAI Grid Workshop, 2009, pp. 35–44
8. S. Chen, J. Qin, Y. Xie, W. Pang and P. Heng, CUDA-based Acceleration and Algorithm Refinement for Volume Image Registration, Int. Conf. on Future BioMedical Information Engineering (FBIE), 2009
9. R. Shams and N. Barnes, Speeding up Mutual Information Computation Using NVIDIA CUDA Hardware, IEEE Computer Society, 2007, pp. 555–560
10. Y. Lin and G. Medioni, Mutual Information Computation and Maximization Using GPU, IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, 2008, pp. 1–6
11. M. Harris, Optimizing Parallel Reduction in CUDA, NVIDIA Tech. Rep., 2007
12. J. West, J. M. Fitzpatrick, M. Y. Wang, et al., Comparison and evaluation of retrospective intermodality image registration techniques, J. Comput. Assist. Tomogr., 21(4), 1997, pp. 554–566
13. C. Chefd'hotel, H. Faugeras, G. Hermosillo and O. Faugeras, Flows Of Diffeomorphisms For Multimodal Image Registration, Proc. of IEEE Int. Symposium on Biomedical Imaging, 2002, pp. 21–28
14. NVIDIA, NVIDIA CUDA C Programming, Best Practices Guide, 2010