Bio-Medical Materials and Engineering 26 (2015) S983–S989
DOI 10.3233/BME-151393
IOS Press

CUDA accelerated uniform re-sampling for non-Cartesian MR reconstruction

Chaolu Feng [a,b,*] and Dazhe Zhao [a,b,c]

[a] *College of Information Science and Engineering, Northeastern University, Shenyang, Liaoning, 110819, China*
[b] *Institute of Medical Information Computing and Network Information Serving, Northeastern University, Shenyang, Liaoning, 110819, China*
[c] *Key Laboratory of Medical Image Computing of Ministry of Education, Northeastern University, Shenyang, Liaoning, 110819, China*

Abstract. A grid-driven gridding (GDG) method is proposed to uniformly re-sample non-Cartesian raw data acquired in PROPELLER, in which a trajectory window for each Cartesian grid is first computed. The intensity of the reconstructed image at this grid is the weighted average of raw data in this window. Taking consider of the single instruction multiple data (SIMD) property of the proposed GDG, a CUDA accelerated method is then proposed to improve the performance of the proposed GDG. Two groups of raw data sampled by PROPELLER in two resolutions are reconstructed by the proposed method. To balance computation resources of the GPU and obtain the best performance improvement, four thread-block strategies are adopted. Experimental results demonstrate that although the proposed GDG is more time consuming than traditional DDG, the CUDA accelerated GDG is almost 10 times faster than traditional DDG.

Keywords: Gridding, magnetic resonance reconstruction, uniform re-sampling, CUDA acceleration

1. Introduction

Since the concept and principle of magnetic resonance imaging (MRI) was presented by Lauterbur, researchers in this field have typically focused their efforts on acquiring data under time-varying gradients [1]. Although data acquired uniformly under a on-off gradient can be reconstructed straightforwardly by Fast Fourier Transform (FFT), fast gradient switch is not feasible in practice [2]. Therefore, acquiring data with non-Cartesian trajectories under smoothly switched gradients have become the focus of intensive investigation by diverse groups of researchers in this field [3]. As signals from central part of the k-space determine the contrast and signal-to-noise ratio (SNR), the multi-strip central oversampling method, namely PROPELLER, is able to correct physiological and motion artifacts and therefore has been successfully applied in high-end MR equipments [4, 5].

To reconstruct images from raw data acquired with non-Cartesian trajectories, Direct Fourier transform (DFT) and data-driven gridding (DDG) are two most important methods [6]. In DFT, intensities

*Corresponding author. Chaolu Feng, College of Information Science and Engineering, Northeastern University, Shenyang, Liaoning, 110819, China. Tel.: +8602483665418; Fax: +860248363446; E-mail: fengchl@ise.neu.edu.cn

of the reconstructed image are weighted sums of raw data where the weightings are estimated by a predefined density compensation function (DCF) [7, 8]. Although images reconstructed by DFT are usually used as a standard to evaluate other reconstruction methods, its high computational requirement is the greatest obstacle for its application in practice. Therefore, DDG methods are used to re-sample the non-uniform raw data onto Cartesian grids. The uniform result is then transferred by FFT to reconstruct the MR image. As one of the most popular methods, convolution interpolation gridding has been widely studied due to its effectiveness and robustness [9]. In convolution interpolation gridding, raw data are convoluted with a predefined diffuse kernel, which is equivalent to re-sample values of the raw data onto the related grids. The interpolation weightings are used to compensate the non-uniform sampling density [8]. Although infinite *sinc* function is considered as the optimal choice of convolution kernel, it has to be replaced by a finite one in practice [10]. Therefore, an investigation on finite convolution functions was developed by Jackson *et al.* where the Kaiser-Bessel function is asseverated as the best choice [11]. In DDG, a convolution window is first computed for each trajectory point. Raw data at this point will then be convoluted to distribute its energy to Cartesian grids in the window with convolution kernel and DCF as the weightings. Therefore, it is time consuming for DDG to uniformly re-sample the raw data.

Due to its ever-increasing in computing power and powerful capabilities in general purpose computing power, the graphics processing unit (GPU) has been widely used in medical image processing [12]. Nevertheless, only few researches concerning acceleration of MR reconstruction using GPU. A GPU accelerated method was proposed by Stone *et al.* for advanced three-dimensional reconstruction [13]. Yang *et al.* tried to accelerate traditional DDG using CUDA (which is the paralell computing architecture of NVIDIA) and found out an inconsistent problem in the CUDA accelerated [14]. Then, a reverse gridding algorithm was proposed and was accelerated using CUDA [14].

In this paper, a grid-driven gridding (GDG) method is first proposed. The proposed method creates a trajectory window for each Cartesian grid (which means grid-driven). The intensity of the reconstructed image at this grid is the weighted average of raw data in this window, which is accomplished by convolution. To improve performance of the proposed GDG, a CUDA accelerated method is then proposed. Experimental results show that the proposed GDG is almost 10 times faster than traditional DDG. The rest of this paper is organized as follows. In section 2, a grid-driven gridding method is presented in PROPELLER trajectory. The CUDA acceleration of the proposed GDG method is presented in Section 3. Experimental results are given in section 4. This paper is discussed and concluded in section 5 and 6.

2. Grid-driven gridding

Let M, M_{RS}, R, and C be the raw data, the gridding result, the equal spaced Cartesian grids, and the convolution kernel, respectively. The DDG can be then expressed as $M_{RS} = [M(u,v) \bullet D * C(u,v)] \bullet R(u,v)$ where (u,v) are the k-space coordinates of sampling trajectory S, D is the density compensation function defined by $D = S(u,v)/(S(u,v) * C(u,v))$, and $*$ is the convolution operator.

2.1. Data acquisition

In PROPELLER, raw data are acquired in k-space with N concentric rotated strips. Each strip consists of L phase encoded lines, which are at the center of M_y parallel linear trajectories in phase encoding direction. Each trajectory line is filled with M_x sampling points in frequency-encoding direction. The overlap of neighbouring strips emerges into a central circle in k-space. Let K be the size of field of view (FOV) and $Re(\star)$ and $Im(\star)$ are the components of $\star$ in frequency-encoding direction and phase

encoding direction, respectively. The first trajectory strip S_1 can then be computed using $Re(S_1(u,v)) = -Re(K)/2 + Re(K)/M_x \times u$ and $Im(S_1(u,v)) = -Im(K)/2 + Im(K)/M_y \times (M_y - L/2 + v)$ where $0 \leq u < M_x$ and $0 \leq v < L$. The next strip S_{i+1} is achieved by rotating S_i anticlockwise with θ radian, in which θ is the inner angle between two adjacent strips given by π/N.

2.2. Uniform re-sampling

To uniformly re-sample the raw data acquired with PROPELLER, a convolution window with size W for each grid in the first strip is first computed in the proposed GDG method. The convolution window is clockwise rotated with an angle γ until the final convolution window in trajectory matrix S is achieved where γ is the inner angle between the i-th strip and the first one given by $\gamma = i \times (\pi/N)$. For grid point $G(u,v)$, the lower and upper bounds of the convolution window can be computed by $irmin = ((G(u,v) - G_c - W)) \oslash G \otimes K$ and $irmax = ((G(u,v) - G_c + W)) \oslash G \otimes K$. Rotation of the convolution window with an angle γ clockwise is equal to anticlockwise rotate it with $-\gamma$, which can be accomplished by right multiplying the window with a unit matrix consisting of *cos* and *sin* values of $-\gamma$. Bounds of the convolution window are updated by the lower and upper bounds of the rotation result. The bounds $irmin$ and $irmax$ are then used to computed a coordinate range of the convolution window in frequency-encoding direction $\lfloor umin, umax \rfloor$ and a coordinate range in phase-encoding direction in the form of $[vmin, vmax]$ where $umin = (Re(irmin) + Re(K)/2) \times M_x/Re(K)$, $umax = (Re(irmax) + Re(K)/2) \times M_x/Re(K)$, $vmin = (Im(irmin) + Im(K)/2) \times M_y/Im(K) - M_y - L/2$, and $vmax = (Im(irmax) + Im(K)/2) \times M_y/Im(K) - M_y - L/2$. For any element (m,n) in $[(umin, vmin), (umax, vmax)]$, the corresponding coordinates (p,q) which are in the trajectory matrix S can be given by $p = m + i \times L$ and $q = n$, respectively. In the proposed GDG, the Euler distance between trajectory $S(p,q)$ and grid position $G(u,v)$ is then computed using $dk = K \otimes (G(u,v) - G_c) \oslash G - S(p,q)$. If $|dk|$ is less than $|W_k|$ where $|W_k| = K \otimes W \oslash G$, raw data $M(p,q)$ is convolution interpolated to accumulate the contribution to the grid point $G(u,v)$. It is obvious that there might be an overlap among different convolution windows which may result in a read-read conflict. But this will not influence the consistency of the re-sampling result.

3. Acceleration with CUDA

It is obvious that the proposed GDG method is more complicated than traditional DDG. Therefore, a CUDA accelerated method is proposed to improve the performance of the proposed GDG in this section. The accelerated method takes full advantage of the data parallel property of uniform sampling. It also resolves the inconsistency pointed by Yang *et al.* when traditional DDG is directly accelerated using multithreading [14]. The pseudo-code of the CUDA accelerated GDG can be given in Algorithm 1.

Different from traditional DDG, the proposed CUDA accelerated GDG creates thread-block according to the Cartesian grids. Therefore, it is possible for thread-block scheduler to create only one FIFO (First In First Out) queue. One block in this queue at a time is transferred to processors. The processors corresponds to streaming multi-processors (SMs) of the GPU. Processors directly operate data M, S, and D, which is marked with the bold blue arrow in the lower right corner of Figure 1.

As SMs are relevant to thread-blocks and there is a thread-block scheduler, threads in each block are controlled by a thread scheduler. Convolution interpolation is implemented by streaming processors (SPs) of the GPU through scheduling the thread queue as shown in Figure 2. In fact, SPs are the ultimate executive units for a CUDA enabled GPU. The proposed CUDA accelerated GDG guarantees that SPs

Algorithm 1 CUDA accelerated grid-driven gridding

Input: M, S, D, C, W
Output: M_{RS}

- function __global *GriddingKernel*(M_{RSdev}, M_{dev}, S_{dev}, D_{dev}, C_{dev}, W_{dev})

1. Compute the index of current thread in the Cartesian grids from the block index and the block inside index.
2. For this grid point, compute the convolution window.
3. Accumulate the contribution of raw data in the convolution window with D and C as the weightings.

- function *CudaGridDrivenGrid*(M_{RS}, M, S, D, C, W)

1. Detect the number and performance of GPUs performance and configure CUDA run-time parameters
2. Allocate global memory on the GPU for M, S, D, C, M_{RS} and load them (except M_{RS}) from system memory to GPU memory via *cudaMemcpy* function with macro *cudaMemcpyHostToDeive*.
3. Create thread-blocks by dividing M_{RS} with thread-block size. Note that M_{RS} should be expanded to integral multiples of thread-block size and filled the expansion with 0.
4. Invoke *GriddingKernel* with parametric configuration $<<< B_n, T_n, M_n, S_n >>>$, where B_n is the block number, T_n indicates thread number inside each block, M_n represents shared memory size required inside each block, and S_n is the processing stream number.
5. Copy re-sampling result M_{RS} back to system memory via *cudaMemcpy* function with macro *cudaMemcpyDeviceToHost*.

in one SM write the same element of M_{RS}. However, if traditional DDG was accelerated with CUDA directly, it would like to write a number of different elements of M_{RS}.

4. Results

We implemented the proposed method on a general PC with CUDA 1.1, where the CPU is Intel ®Core(TM)2 E6550 2.33GHz and the GPU is NVIDIA GeForce 8800 GT with 112 CUDA cores, 512MB 256-bit memory, and 64 GB/sec bandwidth. The reason to accelerate the proposed GDG on Geforece 8800 GT with CUDA 1.1 is that they are standard configuration of the first release of the Neusoft Ltd. Superstar 1.5T MRI system besides that CUDA is downward compatible and we do not need to update our hardware. In this section, CPU implementations are all written in language C and run on single core straightforward. To perform a fair compare, single precision are used both in CPU and GPU implementations due to the non-support of double precision in CUDA 1.1. Two groups of k-space data are used to validate the proposed GDG. These data are both acquired by Neusoft Superstar 1.5T MRI system using PROPELLER trajectory with $N = 17$ and $L = 24$. The parameter M_x is set to be 960

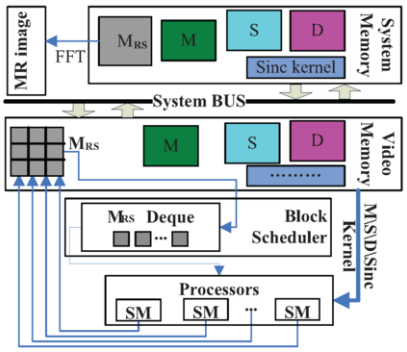

Fig. 1. The CUDA acceleration of the proposed GDG.

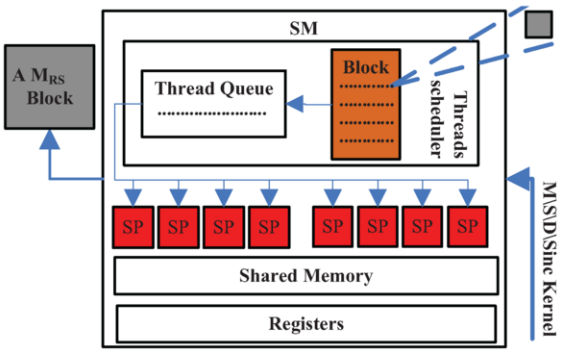

Fig. 2. The thread model inside each thread-block.

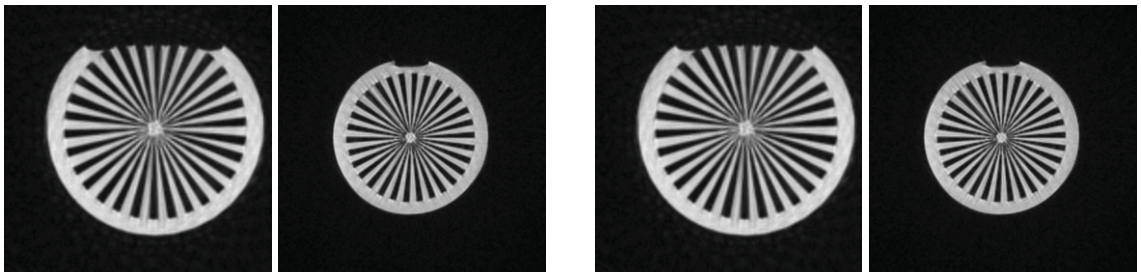

Fig. 3. Final reconstruction results of water phantom 1 and water phantom 2 reconstructed by traditional DDG (left) and the proposed CUDA accelerated GDG (right).

and 1024, respectively. While the lower one is named water phantom 1, the higher one is called water phantom 2. To compare conveniently, 512×512 is considered as the re-sampling resolution for both of the data. Figure 3 shows final reconstructed by transferring the re-sampling results with FFT.

In CUDA, the minimal execution unit is thread and a number of threads form a block. Threads in the same block share the memory and are able to be synchronized. Therefore, it is necessary to find out the best threads allocation strategy to utilize the hardware resource reasonably. Although more threads in the same block can make use of the latent read-write efficiency of the shared memory, too many threads allocated in one block will cost too much time for thread scheduling. Moreover, the number of threads in the same block is also limited by specific hardware. On the contrary, thread reduction can improve efficiency of each block. But this might not take full advantage of the computational resources and will bring additional pressure to SMs in resource scheduling. A performance analysis on the proposed CUDA accelerated GDG is taken with thread-block being set to be 16×16, 12×12, 8×8, and 4×4, respectively. From Figure 4, we can see that DDG and GDG achieved their best performance when the thread-block is 16×16 and 8×8, respectively. As mentioned earlier, the proposed GDG generates a convolution window for each Cartesian grid. Thus, the CUDA accelerated GDG has to create $512 \times 512 = 262144$ threads or more (if there is a filling operation as mentioned earlier). And there are at least $408 \times 960 = 391680$ threads in CUDA accelerated traditional DDG. Due to the rotation operation to compute the final convolution window, the CUDA kernel function of the proposed CUDA accelerated GDG is much more time consuming than CUDA accelerated DDG. Hence, the proposed CUDA accelerated GDG is more easily influenced by executive efficiency of each thread rather than scheduling time. Thus, the smaller the thread-block is, the less the re-sampling time is. On the other hand, also restrained by computational resource and increase of block scheduling, performance of the proposed CUDA accelerated GDG will decline when thread-block size achieves a certain degree (8×8 in Figure 4). However, CUDA accelerated traditional DDG is more easily influenced by thread-block scheduling than the proposed CUDA accelerated GDG. The smaller the block is, the more blocks there are. Therefore, scheduling cost more time and results in re-sampling time increase as shown in Figure 4.

As mentioned earlier, resolutions of the experimental data are 408×960 and 408×1024, respectively. High resolution data results in more threads than low resolution data in CUDA accelerated traditional DDG. Thus, uniformly re-sampling the high resolution data is more time consuming as show in Figure 5. Moreover, as more easily affected by block scheduling than computation commands existing in each thread as mentioned earlier, time performance of CUDA accelerated traditional DDG becomes much worse while the number of thread-block increases as shown in Figure 5. However, the proposed CUDA accelerated GDG creates threads based on the resolution of the sampling grid. Thus, the total thread number is always equal to 512×512 except for the filling zero operation at block 12×12. But the filling

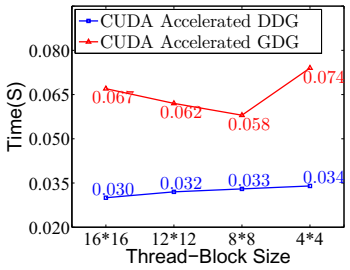

Fig. 4. Comparison of CUDA accelerated GDG with CUDA accelerated traditional DDG for water phantom 1.

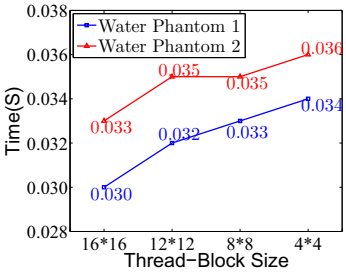

Fig. 5. Performance of CUDA accelerated traditional DDG.

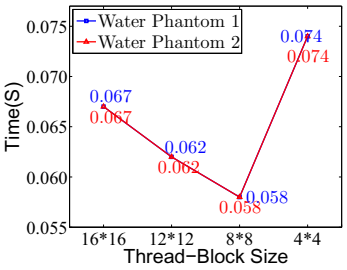

Fig. 6. Performance of the proposed CUDA accelerated GDG.

is the same for both of the data. Therefore, time performance of the proposed CUDA accelerated GDG is almost the same in re-sampling these two data with the same thread-block strategy as shown in Figure 6.

Performance comparison between the proposed GDG and traditional DDG is given in Figure 7. As traditional DDG is much more easily influenced by the resolution of raw data, it cost 0.500 seconds to re-sample the 408960 data onto a 512×512 grid but 0.533 seconds for the other data. As the proposed GDG is much more complicated, it needs 1.313 seconds to re-sample the low resolution data and 1.401 seconds for the high resolution one. However, CUDA accelerated traditional DDG receives its best efficiency 0.030 seconds for the low resolution data at block size 16×16 and 0.033 seconds for the higher one. Whereas the proposed CUDA accelerated GDG is not influenced by the resolution of raw data but the re-sampling grid. Thus, it achieves the best performance 0.058 seconds at block size 8×8 for both of these two data. For the low resolution data, speedup ratio of traditional DDG is 16.667 and grid-driven gridding reaches up to 22.638. But for the high resolution data, the numbers are 16.152 and 24.155, respectively. As CUDA accelerated traditional DDG will results in an inconsistency, performance comparison of the proposed CUDA accelerated GDG with the traditional DDG is meaningful. As shown in Figure 8, the ultimate speedup ratio is nearest to 10 (9.190 in fact) for water phantom 2.

5. Discussions

In the Neusoft Superstar 1.5T MRI system, images are reconstructed slice by slice. Although re-sampling can be accelerated using streaming SIMD extensions (SSE) in which the CPU is able to perform multiple operations at the same time on each core, we accelerated it in another way in this paper. The re-sampling time of the CUDA accelerated DDG and GDG are all recorded from the beginning to the end, which includes data transferring between the CPU and GPU. Hence, if newer GPUs with higher memory transferring efficiency and computing power are used, the CUDA implementation can run without modification to give a greater speedup.

6. Conclusion

A grid-driven gridding algorithm has been proposed to uniformly re-sample the non-Cartesian data for magnetic resonance image reconstruction. The proposed method computes a trajectory window for each Cartesian grid. The intensity of the reconstructed image at this grid is the weighted average of raw data in this window, which is accomplished by convolution. To improve performance of the proposed GDG

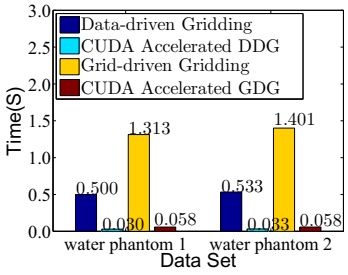

Fig. 7. Comparison of the best run time among traditional DDG, GDG, and CUDA accelerated DDG and GDG.

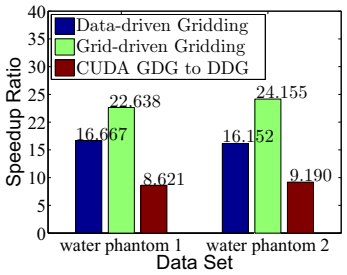

Fig. 8. Speedup ratio of traditional DDG, GDG, and CUDA accelerated GDG to traditional DDG.

method, a CUDA accelerated method is then proposed. Experimental results show that the proposed GDG is correct and is almost 10 times faster than traditional DDG.

Acknowledgment

This work was supported by the Fundamental Research Funds for the Central Universities of China under grant N140403006, N140407001, and N140402003, the Chinese National Natural Science Foundation under grant No. 61172002, the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under grant 2014BAI17B01.

References

[1] N. O. Addy, H. H. Wu, D. G. Nishimura, Simple method for MR gradient system characterization and k-space trajectory estimation, Magnetic Resonance in Medicine **68** (2012) 120-129.
[2] V. Parot, C. Sing-Long, C. Lizama, C. Tejos, S. Uribe, P. Irarrazaval, Application of the fractional Fourier transform to image reconstruction in MRI, Magnetic Resonance in Medicine **68** (2012) 17-29.
[3] K. L. Wright, J. I. Hamilton, M. A. Griswold, V. Gulani, N. Seiberlich, Non-Cartesian parallel imaging reconstruction, Journal of Magnetic Resonance Imaging **40** (2014) 1022-1040.
[4] C. Feng, J. Yang, D. Zhao, J. Liu, CUDA accelerated method for motion correction in MR PROPELLER imaging, Magnetic resonance imaging **31** (2013) 1390-1398.
[5] J. G. Pipe, W. N. Gibbs, Z. Li, J. P. Karis, M. Schar, N. R. Zwart, Revised motion estimation algorithm for PROPELLER MRI, Magnetic Resonance in Medicine **72** (2014) 430-437.
[6] K. P. Pruessmann, Encoding and reconstruction in parallel MRI, NMR in Biomedicine **19** (2006) 288-299.
[7] G. E. Sarty, R. Bennett, R. W. Cox, Direct reconstruction of non-Cartesian k-space data using a nonuniform fast Fourier transform, Magnetic Resonance in Medicine **45** (2001) 908-915.
[8] N. R. Zwart, K. O. Johnson, J. G. Pipe, Efficient sample density estimation by combining gridding and an optimized kernel, Magnetic Resonance in Medicine **67** (2012) 701-710.
[9] Y. Feng, Y. Song, C. Wang, X. Xin, Q. Feng, W. Chen, Fast direct fourier reconstruction of radial and PROPELLER MRI data using the chirp transform algorithm on graphics hardware, Magnetic Resonance in Medicine **70** (2013) 1087-1094.
[10] M. Uecker, S. Zhang, D. Voit, A. Karaus, K.-D. Merboldt, J. Frahm, Real-time MRI at a resolution of 20 ms, NMR in Biomedicine **23** (2010) 986-994.
[11] J. I. Jackson, C. H. Meyer, D. G. Nishimura, A. Macovski, Selection of a convolution function for Fourier inversion using gridding [computerised tomography application], IEEE Transactions on Medical Imaging **10** (1991) 473-478.
[12] A. Eklund, P.Dufort, D. Forsberg, S.M. LaConte, Medical image processing on the GPUâĂŞPast, present and future,Medical image analysis **17** (2013) 1073-1094.
[13] S. S. Stone, J. P. Haldar, S. C. Tsao, B. Sutton, Z.-P. Liang, et al., Accelerating advanced MRI reconstructions on GPUs, Journal of Parallel and Distributed Computing **68** (2008) 1307-1318.
[14] J. Yang, C. Feng, D. Zhao, A CUDA-based reverse gridding algorithm for MR reconstruction, Magnetic resonance imaging **31** (2013) 313-323.