Accelerating the Reconstruction of Magnetic Resonance Imaging by Three-Dimensional Dual-Dictionary Learning Using CUDA

Jiansen Li, Jianqi Sun, Ying Song, Yanran Xu and Jun Zhao*, *IEEE Member*

***Abstract*— An effective way to improve the data acquisition speed of magnetic resonance imaging (MRI) is using undersampled k-space data, and dictionary learning method can be used to maintain the reconstruction quality. Three-dimensional dictionary trains the atoms in dictionary in the form of blocks, which can utilize the spatial correlation among slices. Dual-dictionary learning method includes a low-resolution dictionary and a high-resolution dictionary, for sparse coding and image updating respectively. However, the amount of data is huge for three-dimensional reconstruction, especially when the number of slices is large. Thus, the procedure is time-consuming.**

In this paper, we first utilize the NVIDIA Corporation's compute unified device architecture (CUDA) programming model to design the parallel algorithms on graphics processing unit (GPU) to accelerate the reconstruction procedure. The main optimizations operate in the dictionary learning algorithm and the image updating part, such as the orthogonal matching pursuit (OMP) algorithm and the k-singular value decomposition (K-SVD) algorithm. Then we develop another version of CUDA code with algorithmic optimization. Experimental results show that more than 324 times of speedup is achieved compared with the CPU-only codes when the number of MRI slices is 24.

I. INTRODUCTION

The magnetic resonance imaging (MRI) modality is safe and can achieve tomography in any direction with high soft tissue contrast. Therefore, it has been clinically widely used. However, the sampling time of traditional MRI is long, which may lead to discomfort of patients and motion artifacts in the reconstruction images, limiting its use in many hot and focus areas, such as cardiac imaging.

Fortunately, performing reconstruction from undersampled k-space data can improve the data acquisition speed by sampling less data. The compressed sensing (CS) theory [1]-[2] suggests that a sparse signal can be reconstructed from its sparse representation under certain conditions. Therefore, it is possible for us to perform reconstruction of MRI images using undersampling k-space data. Dictionary learning method [3]-[5] is a very effective way to establish adaptive dictionaries with good sparsity, then the dictionary can be used to train the sparse representation and reconstruct the images. In our work, we utilize the k-singular value decomposition (K-SVD) [6] and orthogonal matching pursuit (OMP) [7]-[8] algorithm to train dictionaries and obtain the sparse representation.

Research supported by National Natural Science Foundation of China (No. 813716234), National Basic Research Program of China (2010CB834302), and Shanghai Jiao Tong University Medical Engineering Cross Research Funds (YG2013MS30 and YG2011MS51). *Asterisk indicates corresponding author.*

Jiansen Li, Jianqi Sun, Ying Song, Yanran Xu and *Jun Zhao are with the School of Biomedical Engineering, Shanghai Jiao Tong University, China (corresponding author Jun Zhao: junzhao@sjtu.edu.cn).

Ying Song et al. [3] proposed a new algorithm for the reconstruction of undersampled k-space data. They used three-dimensional dictionary and performed reconstruction of multi-slice MRI images. In their method, data is divided into blocks, and the spatial correlation among slices can be used when training dictionaries, as well as updating the resulting images. Furthermore, they use a new dictionary learning scheme – dual-dictionary learning, with a low-resolution dictionary D^{low} and a high-resolution dictionary D^{high}, for sparse coding and image updating respectively. Their work indicates that dual-dictionary scheme is better than the single dictionary scheme.

However, when it comes to three-dimensional reconstruction, the amount of data is huge and will increase with the number of slices increasing. In addition, K-SVD and OMP are both iterative algorithm, and they require more time for execution, and the situation will get even worse when the amount of data becoming larger. Therefore, accelerating the reconstruction procedure is needed. We first design the parallel algorithm on graphics processing unit (GPU) directly under the scheme of compute unified device architecture (CUDA) [9] (we call this version of CUDA code as "original CUDA"), utilizing GPU's strong computing power and high performance in parallel computing. Then we carry out algorithmic optimization proposed in Reference [10], and on this basis, we develop another version of CUDA code (we call it CUDA after Algorithmic Optimization, i.e. CUDA-AO). In both of the two versions of CUDA code, we emphasize on the optimization of the K-SVD and OMP algorithm, and implement their parallel version codes on GPU.

CUDA is a programming model and a general purpose parallel computing platform introduced by the NVIDIA Corporation. It allows the programmers to easily develop programs on GPU without much knowledge of the GPU internal structure and the parallelization mechanism of computing in threads. Due to its powerful computing capability, CUDA is being increasingly used in the scientific computing areas. In our work, the CUDA runtime API and the CUDA Basic Linear Algebra Subroutines (CUBLAS) library [11], which is a GPU-accelerated version of the complete standard BLAS library, are used to develop the CUDA programs.

II. THE ALGORITHM SCHEME

A. Overview of the algorithm

The formulation of the multi-slice MRI reconstruction [3] is shown in (1), where $\mathbf{X}$ is the unknown images to be reconstructed, $\mathbf{R}_{i,j,t}$ is the extraction operator, indexed by the location of the lowest top-left point, (i, j, t), in

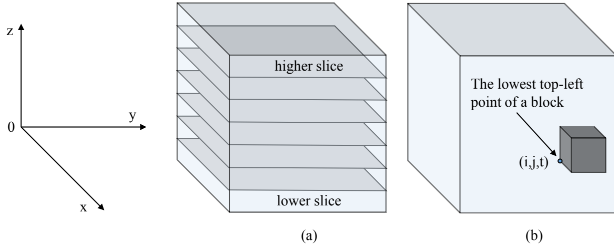


Fig. 1: (a) Three-dimensional MRI series volume. The multi-slice MRI images are concerned. (b) The multi-slice MRI images are divided into blocks. The lowest top-left point of a block is shown.

the image block, which is shown in Fig.1. $\boldsymbol{\alpha}_{i,j,t}$ is the sparse representation of $\mathbf{X}$ under the dictionary D, F_u is the undersampling Fourier matrix, y is the undersampled k-space measurements, ρ is the sparsity level, and ν is defined by $\nu = \lambda/\sigma$, where λ is a positive constant and σ is the standard deviation of the noise.

$$\min_{\mathbf{X},\boldsymbol{\alpha}_{i,j,t}} \sum_{i,j,t} \| \mathbf{R}_{i,j,t}\mathbf{X} - \mathbf{D}\boldsymbol{\alpha}_{i,j,t} \|_2^2 + \nu \| \mathbf{F}_u\mathbf{X} - \mathbf{y} \|_2^2 \quad s.t. \ \| \boldsymbol{\alpha}_{i,j,t} \|_2^2 \leqslant \rho \quad \forall i,j,t \tag{1}$$

The reconstruction scheme consists of three steps:

- (1) Dictionary learning step: In this step, the K-SVD algorithm is used to train the two dictionaries D^{low} and D^{high}, then they can be used in the next step.
- (2) Sparse coding step: In this step, the image $\mathbf{X}$ is assumed to be fixed, and the dictionary D^{low} is used to get the sparse representation $\boldsymbol{\alpha}$.
- (3) Image updating step: The sparse representation attained in the last step and the high-resolution dictionary D^{high} are used to reconstruction the final images.

The execution time of the OMP and K-SVD algorithm accounts for the most largest proportion of the total time consumed of the reconstruction process, thus we focus on OMP and K-SVD and develop their parallel algorithms and codes on CUDA to accelerate the whole reconstruction process.

B. Implementing OMP on CUDA

OMP is a greedy algorithm and at each step it chooses an atom from the dictionary, making sure that the atom is the most closest to the residual signal. Then the signal is orthogonally projected to the selected atoms to achieve the approximation. The detail of OMP is shown in Algorithm 1. $\mathbf{X}$ is the input signal matrix, and its columns correspond to the blocks shown in Fig.1. The data in one block is rearranged to form a column vector and then stored in one column of $\mathbf{X}$. Each column in $\mathbf{X}$ needs a cycle of loop.

The OMP algorithm uses iterative mechanism, requiring very large amount of computation. OMP is an important part in the dictionary learning step [6] and the sparse coding step [3], [12]. Reduction of the time consuming by OMP is significant. In the original CUDA method, we just parallelize the codes inside the iteration in Algorithm 1. Rubinstein et al [10] introduced a method that using Cholesky factorization to avoid the computation of the pseudo-inversion of matrices in the OMP algorithm, and OMP can be accelerated significantly in this way. We use this method in our codes and then transfer it to CUDA. The modified OMP algorithm is more suitable for implementation on CUDA.

Algorithm 1 Orthogonal Matching Pursuit (OMP)

Input:

$\mathbf{D}$: dictionary. ρ: sparse level. $\boldsymbol{\alpha}$: sparse representation. $\mathbf{X}$: signal. $\mathbf{r}$: residual signal. $\boldsymbol{\Lambda}$: selected atoms. $\mathbf{d}_j$: the jth column of $\mathbf{D}$.

Main Procedure:

For each column in $\mathbf{X}$, repeat the following operations until the sparse level is reached:

1: $j \leftarrow \arg\max_i |\mathbf{d}_i^T\mathbf{r}|$
2: $\boldsymbol{\Lambda} \leftarrow \boldsymbol{\Lambda} \cup \mathbf{d}_j$
3: $\boldsymbol{\alpha} \leftarrow \arg\min_{\boldsymbol{\alpha}} \|\mathbf{X} - \boldsymbol{\Lambda}\boldsymbol{\alpha}\|_2^2$
4: $\mathbf{r} = \mathbf{X} - \boldsymbol{\Lambda}\boldsymbol{\alpha}$

Output: $\boldsymbol{\alpha}$

In CUDA, we assign tens of thousands of threads to compute the main procedure in Algorithm 1, thus the loops can be eliminated, and all the operations for each column of the signal can run simultaneously. The OMP algorithm in the original CUDA method is shown in Algorithm 2. Please notice that the iteration inside the manipulation of each column of $\mathbf{X}$ for reaching the sparse level still exists, and this can ensure the convergence for each signal component.

Algorithm 2 OMP implemented on CUDA

Input:

Assigning space for all the intermediate variables on the GPU memory and initialized them. Allocating proper size of threads, blocks, and shared memory in CUDA.

Main Procedure:

In the kernel function defined using the __global__ declaration specifier, the pseudo-codes in C++ is:

1: int tid = blockDim.x * blockIdx.x + threadIdx.x;
2: **if** tid is less than the number of columns of the signal **then**
3: Compute the increasing factor for each intermediate variable.
4: For the tidth column of the signal, do the same operations shown in Algorithm 1 from line 1 to 4.
5: **end if**

Output: $\boldsymbol{\alpha}$

It should be noticed that many intermediate variables should be allocated space and initialized on the GPU memory at the beginning. Since all the threads are executed concurrently and the intermediate variable should be assigned for each thread, we ought to compute the increasing factor for their address pointers in the codes as shown in Algorithm 2.

The main part of the kernel function for CUDA is mostly the same as that in Algorithm 1. However, all the function called should be modified properly and defined by the __device__ declaration specifier, which, like __global__, is part of the C extensions of CUDA [9].

C. Implementing K-SVD on CUDA

The K-SVD algorithm is utilized to train both the low-resolution dictionary and the high-resolution dictionary. K-SVD updates only one column of the dictionary at each iteration, and involves only the signals that use the current atom. The detail of K-SVD is shown in Algorithm 3. In the original CUDA method, we parallelize the codes, and use the CUDA version OMP algorithm shown in Algorithm 2. We also use the method of Rubinstein et al [10] to first accelerate the K-SVD algorithmically, and then develop the CUDA version K-SVD for the CUDA-AO method.

Algorithm 3 k-singular value decomposition (K-SVD)

Input:

$\mathbf{D}$: normalized dictionary obtained from the training sets.
$\boldsymbol{\alpha}$: sparse representation. $\mathbf{X}$: signal.
$\mathbf{x}^j$: the jth row of $\mathbf{X}$. $\mathbf{d}_k$: the kth column of $\mathbf{D}$.

Main Procedure:

Firstly, do the sparse coding step:
Using the OMP algorithm to get the sparse representation $\boldsymbol{\alpha}$.
Secondly, do the dictionary updating step:

1: **for** the kth column of the dictionary **do**
2: **if** the sparse level is not reached **then**
3: Find the indices of the signals whose representations use $\mathbf{d}_k$.
4: Compute the overall representation err matrix: $\mathbf{E}_k = \mathbf{x} - \sum_{j \neq k} \mathbf{d}_j \mathbf{x}^j$.
5: Update the current column of the dictionary: $\mathbf{d}_k = \mathbf{E}_k / \left\| \mathbf{E}_k \right\|_2$.
6: Update the corresponding row in $\boldsymbol{\alpha}$ with $\mathbf{E}_k^{\mathrm{T}} \mathbf{d}_k$.
7: **end if**
8: **end for**

Output: $\mathbf{D}$

When implementing the K-SVD algorithm on CUDA, the sparse coding step can utilize the accelerated OMP algorithm in Algorithm 2. However, since the current loop should use the results of the last loop in the dictionary updating step, the algorithm cannot be parallelized. But we can still accelerate it by parallelizing each of the operations on CUDA. We also use the CUBLAS library to accelerate the execution of some linear algebra operations. The experiments show very effective results to deal with K-SVD in this way.

D. Reconstructing images using CUDA

As mentioned in Section II-A, the reconstruction scheme consists of three steps. When both the low-resolution dictionary and the high-resolution dictionary have been trained, they can be used to achieve the final image. We employ the method proposed in [3] and rewrite the codes using CUDA. Firstly, we use the low-resolution dictionary to obtain the sparse representation, where the OMP algorithm is used. The OMP is implemented on CUDA as mentioned in the above Section II-B.

TABLE I: Execution time and speed-up obtained with the original CUDA method and CUDA-AO method

Number of slices	Time consumed (seconds)			Speed-up	
	Original CPU	Original CUDA	CUDA-AO	Original CUDA	CUDA-AO
4	1466.73	73.275	19.14	20.02	76.63
8	7251.59	146.44	45.13	49.52	237.64
12	14361.8	187.35	71.62	76.66	200.53
16	25952.9	303.65	92.59	85.47	280.31
20	29214.5	310.51	95.19	94.09	306.91
24	35789.8	377.33	110.29	94.85	324.51

When updating the reconstruction results, experiments show that this procedure consumes very little time, which is less than 0.01 seconds. Therefore, CUDA has no advantage in this step, and we just leave the codes unchanged.

III. RESULTS

We present the performance and the acceleration effects of the original CUDA method and the CUDA-AO method. Furthermore, to verify that the two versions of CUDA codes have similar quality of the result images, we compare the peak signal-to-noise ratio (PSNR) and the reconstruction error of the results. The formulas we use are shown in (2) and (3), where MSE is the mean squared error, and MAX_I is the maximum possible pixel value of the image.

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}\right) \tag{2}$$

$$\mathrm{error} = \frac{\left\|\mathrm{result} - \mathrm{origin}\right\|_2^2}{\left\|\mathrm{origin}\right\|_2^2} \tag{3}$$

We use CUDA 5.5 to develop our codes on GPU, and all the computations are performed with an Intel Xeon E5640 CPU and an NVIDIA GeForce GTX 780 TI GPU, under a Windows Server 2008 operating system. All the codes are written with C++ and CUDA C language. We test the acceleration effect with different number of slices of MRI images. Table I shows the results of original CPU code that has no optimization and that of the original CUDA method and the CUDA-AO method. The execution time of the original CPU codes are also shown.

From Table I, we can see that the reconstruction procedure of MRI can get more than 20 times of speedup using the original CUDA method, and the acceleration effect is even better when using CUDA-AO method, which holds about 324 times of speedup when the number of slices is 24. It is sure that better effect of acceleration can be obtained when the number of slices becomes larger. Fig.2 shows the reconstruction results for a 32-slice MRI dataset, the results of the original CUDA method and the CUDA-AO method are shown. Meanwhile, the corresponding difference images with the original images are also displayed. The size of

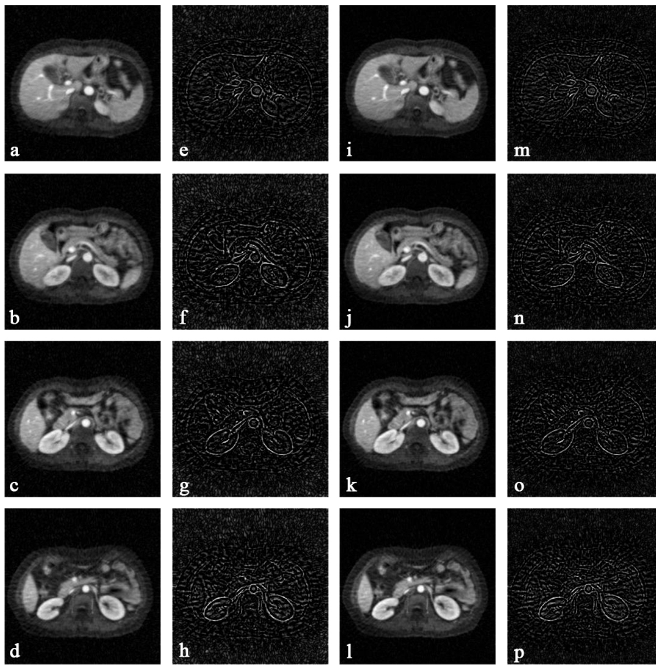


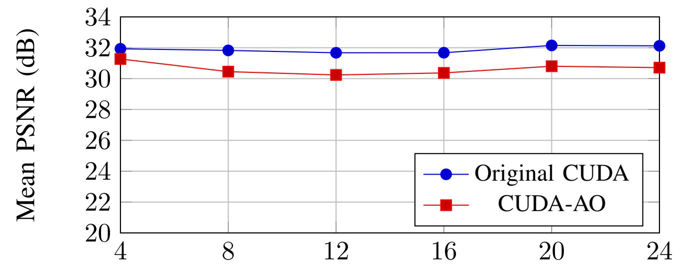
Fig. 2: Reconstruction results and difference images. (a), (b), (c), and (d) are slice 1, 5, 9, and 13 of the reconstruction results respectively obtained with original CUDA method. (e), (f), (g), and (h) are corresponding difference images of (a), (b), (c), and (d) respectively. (i), (j), (k), and (l) are slice 1, 5, 9, and 13 of the reconstruction results respectively obtained with the CUDA-AO method. (m), (n), (o), and (p) are corresponding difference images of (i), (j), (k), and (l) respectively.

each image is 256×256. From Fig.2, we can conclude that the reconstruction results of original CUDA and CUDA-AO are nearly the same. The reconstruction results and the corresponding difference images are similar.

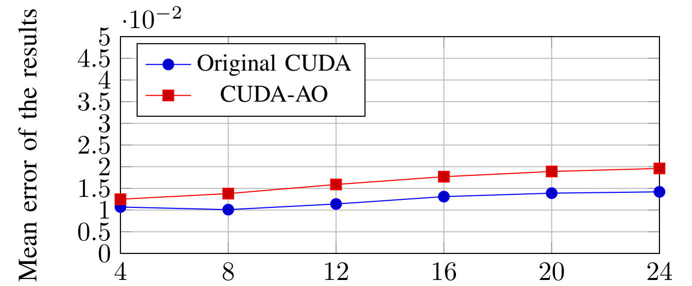
Fig.3 shows the PSNR and reconstruction error compared with the original images computed according to (2) and (3). The difference between the PSNR of the original CUDA method and that of the CUDA-AO method is within 2dB, and the difference of reconstruction error changes in an interval less than 0.005, indicating that the CUDA-AO method has almost the same reconstruction results with the original CUDA method while CUDA-AO is more than 3 times as fast as the original CUDA method.

IV. Conclusions

In this work, we accelerate the reconstruction of MRI by three-dimensional dual-dictionary learning using CUDA. The parallel algorithm on GPU and the acceleration performance are investigated. In addition, we develop two version of CUDA codes: (1) the original CUDA method just directly transfer the original CPU codes to CUDA; (2) the CUDA-AO method first improves the original CPU codes with algorithmic optimization, then implements the codes on CUDA. Experiments show that about 94 times of speedup is achieved using the original CUDA method when the number



(a) Mean PSNR of different number of slices



(b) Reconstruction error of different number of slices

Fig. 3: Mean PSNR and reconstruction error of the original CUDA method and the CUDA-AO method respectively compared with the original images.

of MRI slices is 24, while about 324 times of speedup is obtained with the CUDA-AO method.

This work shows that CUDA together with algorithmic optimization has great advantages in accelerating the reconstruction of MRI.

References

[1] David L Donoho. Compressed sensing. Information Theory, IEEE Transactions on. 2006:52(4):1289-1306.
[2] Jasper van de Gronde and Erald Vuini. Compressed Sensing Overview. 2008.
[3] Ying Song, Zhen Zhu, Yang Lu, Qiegen Liu, and Jun Zhao. Reconstruction of magnetic resonance imaging by three-dimensional dual-dictionary learning. Magnetic Resonance in Medicine. 2013.
[4] I. Tosic, P. Frossard, Dictionary learning, Signal Processing Magazine, IEEE 28 (2) (2011) 27-38.
[5] Kenneth Kreutz-Delgado, Joseph F Murray, Bhaskar D Rao, Kjersti Engan, Te-Won Lee, and Terrence J Sejnowski. Dictionary learning algorithms for sparse representation. Neural computation. 2003:15(2):349-396.
[6] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. Signal Processing, IEEE Transactions on. 2006:54(11):4311-4322.
[7] J.A. Tropp. Greed is good: Algorithmic results for sparse approximation. Information Theory, IEEE Transactions on. 2004:50(10):2231-2242.
[8] Geoff Davis, Stephane Mallat, and Marco Avellaneda. Adaptive greedy approximations. Constructive approximation. 1997:13(1):57-98.
[9] CUDA C Programming Guide. NVIDIA Corporation (2012). Version 5.0. Available at http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.
[10] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit. CS Technion. 2008.
[11] CUBLAS library. NVIDIA Corporation (2012). Version 5.0. Available at http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf.
[12] Saiprasad Ravishankar and Yoram Bresler. MR image reconstruction from highly undersampled k-space data by dictionary learning. Medical Imaging, IEEE Transactions on. 2011:30(5):1028-1041.