Fast GPU Implementation of a Scan-Specific Deep Learning Reconstruction for Accelerated Magnetic Resonance Imaging

Chi Zhang*,†, Sebastian Weingärtner*,†,‡, Steen Moeller†, Kâmil Uğurbil† and Mehmet Akçakaya*,†
* Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN
† Center for Magnetic Resonance Research, University of Minnesota, Minneapolis, MN
‡ Computer Assisted Clinical Medicine, University Hospital Mannheim, Heidelberg University, Heidelberg, Germany
Emails: {zhan4906, sweingae, moell018, ugurb001, akcakaya}@umn.edu

***Abstract*—RAKI is a novel fast MRI image reconstruction algorithm that has been recently proposed, which gives satisfying results for highly accelerated MRI. However, due to RAKI reconstruction depends on multiple convolutional neural networks, implementing RAKI reconstruction is a time-consuming task. In this study, we present accelerate strategies for RAKI implementation aided by GPU parallel programming. Aiming at the characteristics of RAKI, we limited the iteration number of solving optimization problems in the network training stage, while maintaining the reconstruction results are visually satisfying. Further more, according to the independence between multiple networks, we parallelized the training tasks by CPU multiprocessing, which maximizes the performance by fully utilizing GPU resources. According to our experiments, these efforts gave more than 60x speed up compared with conventional, sequential implementation. With the ability of completing RAKI reconstruction in minutes, we are able to bring RAKI into practical applications.**

I. Introduction

Magnetic Resonance Imaging (MRI) is one of the most popular medical imaging technique in modern medicine. Compared with other common imaging modalities such as X-ray imaging and computed tomography (CT), MRI is known as a superior technique that not only gives better image quality for body tissues but also free from any exposure to radiation [1]. Unlike conventional imaging, MRI does not acquire grayscale image directly. Instead, it gives a discrete-sampled Fourier transform of the object image (so-called k-space). The final image is obtained after applying an inverse Fourier transform of the k-space data. This nature of MRI suggests a high-resolution MRI image requires plenty of samples in k-space. However, massive sampling in k-space is a time-consuming task. Blurring and other distortions might occur with even tiny movements of the object during the scanning. Thus in practice, it is impossible to simply extend the scanning time as long as we need. To obtain satisfying resolution within limited scanning time, i.e. limited sample amount, several fast MRI image reconstruction algorithms based on parallel imaging [2] were proposed, such as SMASH [3], SENSE [4], GRAPPA [5], SPIRiT [6]. In parallel imaging, several independent receiver coils scan the same object from different positions simultaneously, which brings information redundancy that could be exploited by the algorithms to eliminate image aliasing artifact caused by under-sampling in k-space. Thus full-sampling in k-space is not necessary anymore to achieve certain resolution. Depending on much fewer samples in k-space, people can reconstruct a high-resolution MRI image with an acceptable trade-off between signal-to-noise-ratio (SNR) and sample amount in k-space.

The reconstruction can be implemented in either image domain (SENSE) or in k-space directly (SMASH, GRAPPA and SPIRiT). RAKI [7] is an improved GRAPPA-like method that reconstructs under-sampled k-space with estimated point-spread functions (kernel) in k-space. In conventional GRAPPA reconstructions, kernels are calibrated by a linear approximation based on certain preserved calibration data (so-called auto-calibration-signal, ACS [8]). However, the hypothesis of linearity limits the quality of GRAPPA reconstruction. In RAKI, kernels are trained by multi-layer convolutional neural networks (CNN), which does not assume any linearity. It is capable of estimating the complex point-wise correlations in k-space. According to M. Akçakaya et al, RAKI gives significant improvement both on SNR and dealiasing in high accelerated cases.

However, training multi-layer CNN is a time consuming task. With conventional implementation, RAKI costs hours to reconstruct a single 2D image. To bring RAKI into practical applications, we accelerated RAKI implementation aided by graphic processing units (GPUs) and TensorFlowa popular deep learning framework that is widely used in machine learning field. In CNN training stage, we compared several optimization approaches and set a stopping criteria that limits the iteration amount while ensuring a satisfying result. Exploiting the independence between multiple coil images, we parallelized the multiple kernel training tasks on GPUs by CPU multiprocessing, in order to fully utilize GPU resources and achieve further acceleration. These efforts gave a 60x speed up in our experiment that complete the RAKI reconstruction in minutes.

II. Robust Artificial-neural-networks for k-space Interpolation (RAKI)

The basic idea of RAKI is first to find a set of shift-invariant convolution kernels in k-space, then estimate the missing data in sub-sampled k-space by performing convolution of its' nearby sampled data with the kernels. The whole k-space is reconstructed in a point-wise manner. In this study, a three-layer CNN is employed to estimate the kernel. For an imaging coil c, let $k_{x,y,c}$ denotes its' k-space sample in 2D position *(x,y)* in ACS region, $w_{c,n}$ represents the convolution kernel of layer n, kernel calibration stage for coil c can be described as solving an optimization problem:

$$\min_{w_{c,1},w_{c,2},w_{c,3},} \sum_{x,y} \|k_{x,y,c} - I_{x,y} * w_{c,1} * w_{c,2} * w_{c,3}\|^2 \quad (1)$$

where * denotes convolution, ACS is a limited region of fully sampled k-space for calibration usage, $I_{x,y}$ is a collection of sampled points around position *(x,y)* that been used to estimate $k_{x,y,c}$. Convolution kernels $w_{c,n}$ are obtained by CNN training depends on ACS data as training sets. The missing data in sub-sampled k-space is then reconstructed by:

$$\hat{k}_{x,y,c} = I_{x,y} * w_{c,1} * w_{c,2} * w_{c,3} \quad (2)$$

where $\hat{k}_{x,y,c}$ is the estimated value for position *(x,y)*, coil c in k-space. To minimize reconstruction error, sampled data will be put back to corresponding position to replace the estimated values. A reconstructed image that free from aliasing is then obtained after an inverse Fourier transform.

III. Fast RAKI Implementation

A. GPU Acceleration and TensorFlow

RAKI trains multiple CNNs during the reconstruction. However, due to CNN training is a time-consuming task, it is necessary to accelerate RAKI in order to put it into practical use. Nowadays, GPUs are widely used to serve the purpose of accelerating deep learning applications. Different from CPU that process multiple tasks in a sequential manner, GPU launches multiple threads handling with multiple tasks in the same time, which is known as parallel computing model. Although CPU has better single thread performance, GPU works better than CPU when it is possible to solve massive independent tasks in parallel manner (Figure 1). Typical computations involved in CNN training such as convolution is highly parallelizable, thus they could be processed much faster by GPU than CPU. TensorFlow [9] is a popular deep learning framework that provides various essential deep learning functions, which includes forward/inverse convolution, activation functions, optimization solvers, etc. Furthermore, TensorFlow supports GPU acceleration aided by CUDA (GPU computing frame work developed by NVIDIA Co.) and CuDNN [10], a CUDA-based deep learning library, which give the edge-cutting performance of deep learning applications.

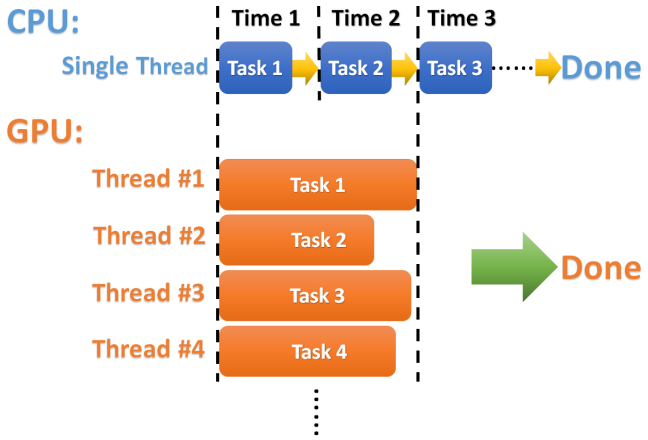

Fig. 1. Sequential processing pattern of CPU (up) and parallel processing model of GPU (down). Although CPU has faster single thread performance, GPU can still faster than CPU when handling with massive parallelizable tasks.

B. Optimization Solver

Choosing a proper optimizer plays significant role in RAKI reconstruction and its' acceleration. There are various optimizers available for particular cases. Once proper optimizer is been chosen, there is only trade-offs between accuracy and iteration time. Conventional deep learning applications require as many iterations as it could to reach higher accuracy of regression. However, in RAKI, our aim is not tiny increase of regression accuracy but visual improvement. Thus the ideal optimizer for RAKI should be the fastest one achieves a visually satisfying standard, rather than optimizers that able to converge to a high accuracy after numerous iterations. However, the trade-offs between visual quality and iteration number is difficult to observe. Insufficient iteration leads to blurring and even aliasing in final result, while massive iterations take a long time that is impractical in medical applications, and it does not promise a better visual quality. A straightforward way to handle with such trade-off is setting a stop criteria that stops the iteration when improvement is sufficiently small, which does not effect the visual experience. For coil c, layer n, we set the stopping criteria as:

$$stop \quad if: \quad \|\hat{w}_{c,n,i} - \hat{w}_{c,n,i-1}\|^2 \leq threshold \quad (3)$$

where $\hat{w}_{c,n,i}$ denotes the weight calculated by the ith iteration for coil c, layer n. In practice, this way is capable of achieving the same visual experience as using a large, fixed iteration number for training. Figure 2 lists the reconstruction result by different optimizers and iterations. In our experiment, Adam optimizer is appeared to be the best choice for RAKI. RAKI with Adam optimizer gives a visually satisfying reconstruction result within the least time cost and iterations. However, it is also obvious in the figure that in our case, running 2500 iterations is not necessary, for it did not give any noticeable improvement compared with the result of 1000 iterations. To determine an exact optimal iteration number is difficult, for the situation could be various coil to coil, and we

employed stochastic starting point at the CNN training stage, which has been proved to be beneficial by existing works [11]. Indeed, the optimal iteration number should be controlled by an adaptive mechanism, which limits the iteration number for the sake of performance, and considering the diversity between coils, it should also allow more iterations when it is necessary. A proper threshold will be the key to reach the balance between reconstruction quality and time cost. Moreover, using a this strategy of adaptive iteration number gives a uniformed error among the multiple kernels, for they will all reach a similar training error controlled by the threshold. Uniformed training error indicates uniformed contrast, sharpness, and noise level in every individual coil images. Considering the final image is combined by all individual coil images without any bias, setting a stopping criteria in the kernel training stage is also helpful to final image combination.

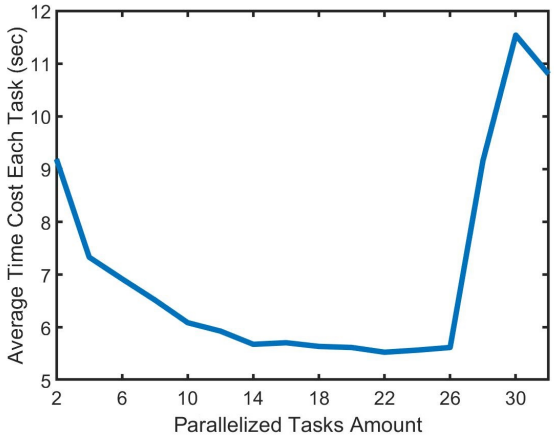

Fig. 2. Comparison between different optimizers, and trade-offs between iteration number and visual quality.

C. Parallelizing Training Tasks by CPU Multiprocessing

In RAKI reconstruction, multiple kernels are trained correspond to multiple coils. GPUs with strong processing power and large memory can handle with single training task easily. However, considering each individual training task is depends on limited data set, it does not require a big amount of GPU resources. Consequently, handling the kernel training tasks in a serial manner is a waste of GPU resources, which limits the benefit of GPU acceleration. Noticing the fact that in RAKI, convolution kernels are trained independently, thus training multiple kernels is also parallelizable. Therefore, another possible way to accelerate RAKI reconstruction is parallelizing multiple kernel training tasks on GPU, which can be equivalently considered as increasing GPU utilization.

GPU operation is controlled by CPU callings. To active a GPU task, we need to first allocate GPU resources and transfer the dependence data by CPU. After the initiate jobs are settled, GPU begins operating by receive a launching command send by CPU. To achieve the parallelization of training tasks, we can launch multiple CPU processes that each of them serves one corresponding coil by doing all initiate jobs and send the GPU calling. As the result, GPU will receive multiple callings in the same time, and it begins to process all of the tasks simultaneously. This approach can give additional speed up by avoiding waste of GPU processing power. Theoretically speaking, as long as the GPU resources are fully utilized, the best performance restricted by the hardware is achieved. To guarantee a smooth operation and a correct result, we need to ensure every training task receive minimum necessary resources. Thus allocating GPU resource is another crucial factor that need to be considered. TensorFlow gives a recommended resource allocation for the best performance depends on several network training parameters, but it can still work with a reduced performance depended on fewer resources. Consequently, the performance can be still limited by GPU memory and bandwidth even they are fully utilized. For instance, limited memory resource leads to extra time costs by additional temporary data writing and erasing, which can be avoided by allocating more memory to the individual task and erase the temporary data after the reconstruction is completed. Thus for some cases with certain limited GPU resources, there exists an optimal number of parallelized tasks for the best performance. In figure 3, we compared several parallel task amounts on a NVIDIA GTX 745 GPU (Single precision 793.30 GFLOPS, 4GB memory), and calculated the average time of training a kernel. As is shown in figure 3, within limited amounts, parallelizing multiple tasks on GPU reduced the average time cost for single kernel training. In our case, the best performance is given when we parallelize 14 to 26 tasks to be processed in the same time. In practical, different image size and network parameters would lead to other optimal region. However, if we put too much load to the GPU, the performance would drop dramatically. The result suggests that with limited GPU resources, we need to carefully design the parallelization by choosing an appropriate number of parallelized tasks. Parallelizing as much as we could might not be the best way for the sake of performance.

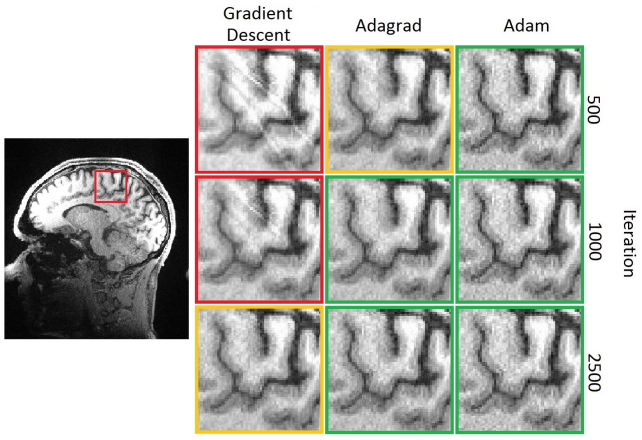

Fig. 3. Average time cost for training a kernel with different parallelized task amount on GTX 745.

IV. Experiments and Results

In our experiment, the brain imaging was performed at Siemens 7T MRI using a 32-channel head coil. 3D-MPRAGE was acquired on a patient, with the field of view 230250154mm , resolution is 0.60.60.6mm , with an acceleration factor = 3. The data is then sub-sampled to reach an high acceleration = 6 to test RAKI reconstruction. RAKI is implemented by python 3.6.2 and TensorFlow 1.3.0, supported by CUDA 8.0 and CuDNN 7.0.5. Python environment is created by Anaconda 3.8.3. All programs were run on the server which has two Intel E5-2643 CPUs (6 cores each, 3.7 GHz), 256 GB memory, and a NVIDIA Tesla K80 GPU (Single precision 8.74 TFLOPS, 24 GB memory). The sever runs Linux 3.10.0 OS with GCC 4.8.5.

We chose Adam optimizer with study rate 0.001 in our experiments. It is been proved by various experiments that Adam is the most effective optimizer for RAKI reconstruction. The results of different strategies are shown below in figure 4 to 5. Figure 4 showed the results of Adam optimizer with 2500 iterations (fig. 4-a), and alternatively with a stopping criteria shown by equation 3 (fig. 4-b), where the threshold is 0.0001. We can see that visually speaking, there is no conspicuous difference between the two results. Figure 4-c shows the difference image between 4-a and 4-b, which is basically a random noise map with very low amplitude. This fact indicates these two images have nearly the same reconstruction quality, no blur or aliasing is remained while using the stopping criteria. However in this experiment, with proper stopping strategy, we were able to obtain such satisfying result by 469 iterations in average (averaged by 10 repeats) instead of using fixed 2500 iterations, which means a four fifth of time saving. Meanwhile, the average of minimum iteration was about 247 iterations, and the average of maximum iteration was 672. For some channels that converged slower than others, they were allowed to run more iterations until they reach a uniformed training error.

We parallelized 64 training tasks on the GPU we have to achieve the best performance. Every individual training task got sufficient resources for a proper operation. The training time of different strategies are shown in figure 5, where we can learn that beginning with a sequential implementation based on Matlab and Matconvnet, it took nearly one and half hour to complete RAKI reconstruction in our case. Simply employ parallel programming with high-efficiency framework gave a nearly 8x speed up but it still take about 12 minutes. Aided by the strategies we have talked in this paper, it cost 1.57 minutes to complete the whole process, which is nearly 60x speed up compared with the sequential version. Promising a satisfying visual experience, we are able to complete RAKI in minutes even handling with such complex case with 32 coils (which needs 64 convolutional neural networks).

V. Conclusion

In this paper, we present strategies aided by parallel programming that accelerate RAKI reconstruction depend on its' own characteristics. In addition to conventional CNN acceleration based on GPU, we shrink the training time by limiting the iteration number with proper criteria, while ensuring a satisfying visual image quality. We also parallelized the individual CNN training tasks to have a further performance improvement. These efforts made considerable contribution to bring the newly proposed RAKI reconstruction algorithm into practical applications.

References

[1] J. Barentsz, S. Takahashi, W. Oyen, R. Mus, P. De Mulder, R. Reznek, M. Oudkerk, and W. Mali, "Commonly Used Imaging Techniques for Diagnosis and Staging," *J Clin Oncol.*, vol. 20, no. 24, pp.:3234-44, 2016.

[2] M. Hutchinson, and U. Raff, "Fast MRI data acquisition using multiple detectors," *Magn. Reson. Med.*, vol. 6, no. 1, pp.87-91, 1988.

[3] D. K. Sodickson, and W. J. Manning. "Simultaneous acquisition of spatial harmonics (SMASH): fast imaging with radiofrequency coil arrays," *Magn. Reson. Med.*, vol. 38, no. 4, pp. 591-603, 1997.

[4] K. P. Pruessmann, M. Weiger, M. B. Scheidegger and P. Boesiger, "SENSE: sensitivity encoding for fast MRI," *Magn. Reson. Med.*, vol. 42, no. 5, pp. 952-962, 1999.

[5] M. A. Griswold, P. M. Jakob, R. M. Heidemann, M. Nittka, V. Jellus, J. Wang, B. Kiefer, and A. Haase, Generalized autocalibrating partially parallel acquisitions (GRAPPA). *Magn. Reson. Med.*, vol. 47, no. 6, pp. 1202-1210, 2002.

[6] M. Lustig, and J. M. Pauly, "SPIRiT: Iterative selfconsistent parallel imaging reconstruction from arbitrary kspace," *Magn. Reson. Med.*, vol. 64, no. 2, pp. 457-471, 2010.

[7] M. Akçakaya, S. Moeller, S. Weingrtner and K. Uurbil, Scan-specific Robust Artificial-neural-networks for k-space Interpolation-based (RAKI) Reconstruction: Database-free Deep Learning for Fast Imaging, *Annual Meeting of the International Society of Magnetic Resonance in Medicine*, Paris, June 2018.

[8] P. M. Jakob, M. A. Griswold, R. R. Edelman, and D. K. Sodickson, "AUTO-SMASH: a self-calibrating technique for SMASH imaging," *Magnetic Resonance Materials in Physics, Biology and Medicine*, vol. 7, no. 1, pp. 42-54, 1998.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "TensorFlow: A System for Large-Scale Machine Learning," in *OSDI*, vol. 16, pp. 265-283, 2016.

[10] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "CuDNN: Efficient primitives for deep learning," *arXiv preprint*, arXiv:1410.0759, 2014.

[11] L. Bottou, "Large-scale machine learning with stochastic gradient descent," proceedings of *COMPSTAT'2010*, pp. 177-186, Physica-Verlag HD, 2010.

[12] D. P. Kingma, and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, arXiv:1412.6980. 2014.

[13] J. Sanders, and E. Kandrot, "CUDA by example: an introduction to general-purpose GPU programming," Addison-Wesley Professional, 2010.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097-1105, 2012.

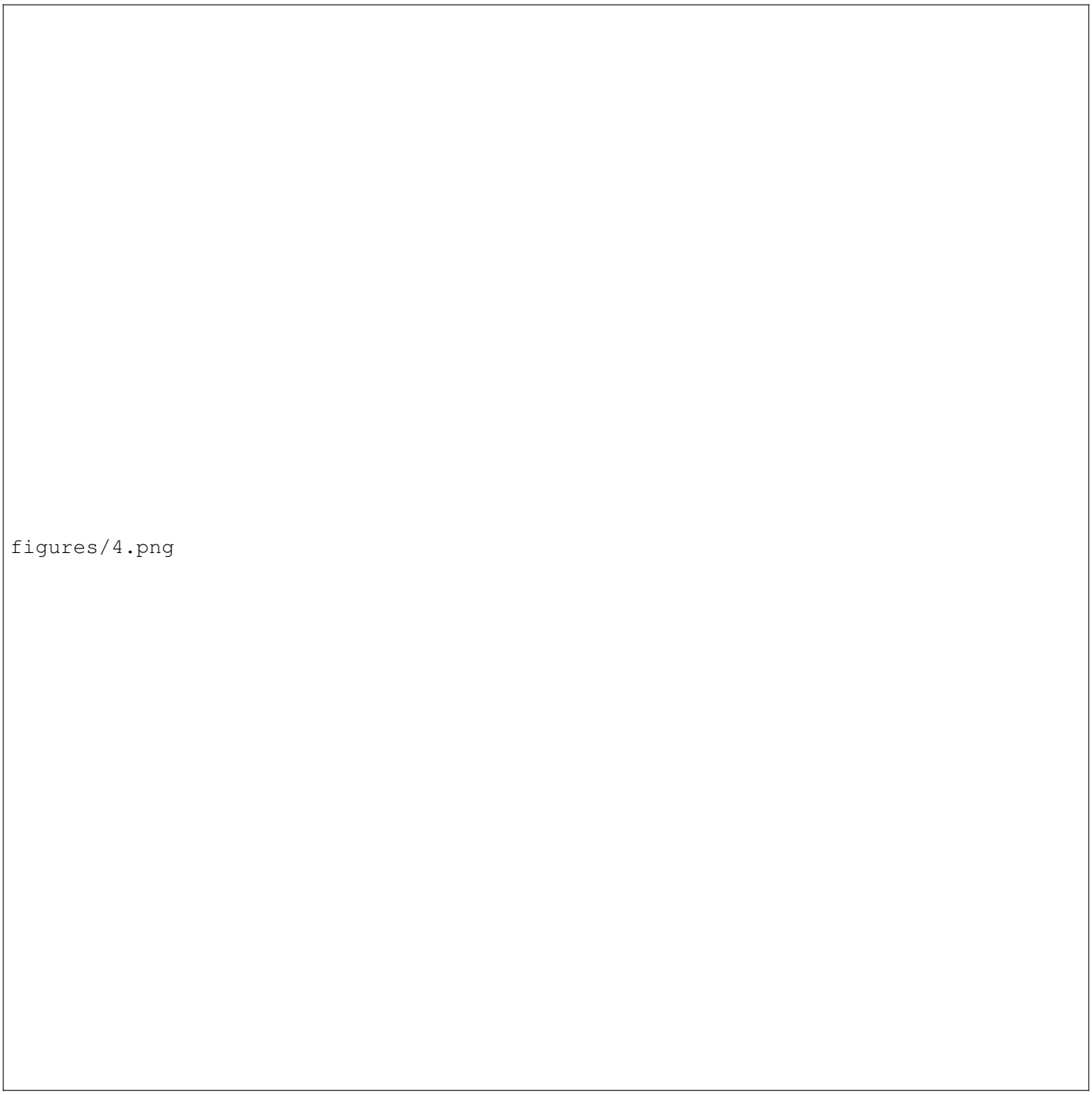

Fig. 4. Reconstruction results with Adam optimizer and different training strategies. (a): Adam optimizer runs 2500 iterations; (b): Adam optimizer with stopping criteria; (c): the difference of the two results. Notice that (a) and (b) has an intensity range from 0 to 1, but c has only 0 to 0.022, which indicates the noise level is small.

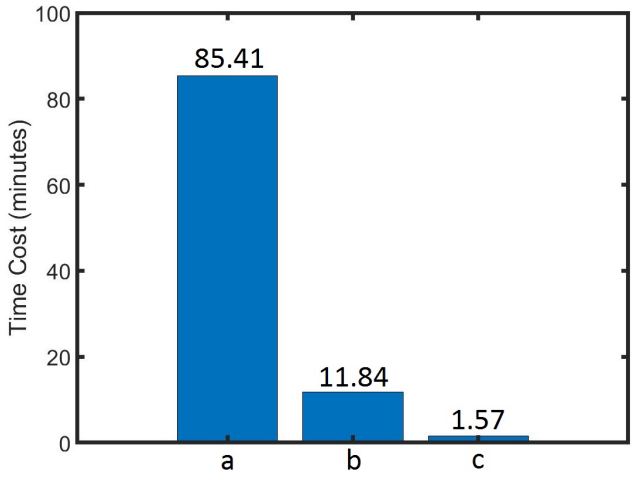

Fig. 5. The time cost of different RAKI implemetation strategies, (a): sequential programming based on Matlab and MatConvnet; (b): parallel programming with TensorFlow and Python; (c): based on the conditions of b, apply a stopping criteria on iteration; (d): based on the condition of c, parallelize multiple tasks on the GPU