

# Efficient Computation of Autocalibrating Parallel Imaging Reconstructions

A. C. Brau<sup>1</sup>, P. J. Beatty<sup>1,2</sup>, S. Skare<sup>3</sup>, R. Bammer<sup>3</sup>

<sup>1</sup>Global Applied Science Lab, GE Healthcare, Menlo Park, CA, United States, <sup>2</sup>Department of Electrical Engineering, Stanford University, Stanford, CA, United States, <sup>3</sup>Department of Radiology, Stanford University, Stanford, CA, United States

**Introduction:** Among the various parallel imaging reconstruction techniques available, autocalibrating methods such as GRAPPA have proven advantageous because they require no coil sensitivity estimation and exhibit relatively benign artifacts, especially at reduced fields-of-view [1,2]. Recently, many advances have improved upon the original published methods. Several authors have shown that the accuracy of autocalibrating techniques can be improved by using a 2D k-space kernel [3-5]; however, this improved accuracy comes at the expense of an increase in computation time. It has also been shown that the advantages of a 2D k-space kernel can be realized by 1D kernels in hybrid ( $x, k_x$ ) space, where a unique 1D kernel is used at each  $x$  location [6]. Acquired data is transformed into hybrid space by applying a 1D Fourier Transform (FT) in the readout direction. However, finding the weights in hybrid space is computationally intensive. In addition, the computational advantage of transforming k-space kernel weights into image space and reconstructing the image in the image domain has been shown for 1D kernels [7].

In this work, we calculate the most computationally efficient method of performing autocalibration reconstructions with a 2D kernel. We separate the reconstruction process into two steps: 1) a “training phase” in which the kernel weights are calculated, and 2) an “application phase” in which the kernel weights are applied to accelerated data, and we show that the most computationally efficient means of obtaining the accuracy of a 2D k-space kernel is to find the kernel weights in k-space and then apply the weights in either hybrid or image space, depending on the application.

**Theory & Methods:** The determination of the kernel weights can be performed in either k-space or hybrid space, while the application of the kernel weights can be performed as a 2D convolution in k-space, a 1D convolution in hybrid space, or a point-by-point multiplication in image space. The computation expense is calculated separately for each of these processing steps. First, the total number of complex-valued multiplications needed to find the weights in k-space vs. hybrid space is determined. Second, the total number of complex-valued multiplications required to reconstruct accelerated data in k-space, image space, or hybrid space is calculated. In addition, the number of complex-valued multiplications required to convert k-space weights to equivalent hybrid space and image space weights are calculated.

**Results:** For the training phase, the number of weights used to fill in a missing data point in k-space is  $N_c W_x W_y$ , where  $N_c$  is the number of coils and  $W_x$  and  $W_y$  are the kernel widths, as shown in Fig. 1. The most significant term in computing the weights in k-space requires  $F_k = N_c (N_c W_x W_y)^2$  complex-valued multiplications, where  $N_f$  is the number of fits performed in the calibration region. For  $N_x$  values in the readout direction, modeling the weights by a cosine basis in hybrid space requires an additional  $(N_x/W_x)F_k$  multiplications, making it significantly more expensive.

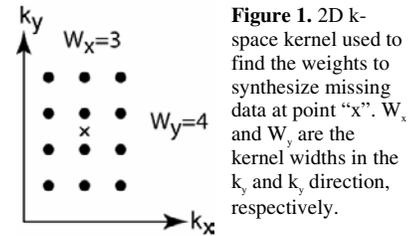
For the application phase, applying the weights in k-space requires  $R_k = N_c W_x W_y N_x N_y$  complex-valued multiplications, where  $N_x$  is the number of phase encode lines that must be synthesized. With a 1D kernel in k-space,  $R_k/W_x$  multiplications are needed, while in image space only  $(N_y/N_x)R_k/(W_x W_y)$  multiplications are needed, where  $N_y$  is the total number of lines in the phase encode direction in the reconstructed image. The number of multiplications for converting the weights from k-space can also be expressed in terms of  $R_k$ , as shown in Fig. 2.

To get a sense of the magnitude of these numbers, we can plug the following reasonable values into the equations above:  $W_x=5$ ,  $W_y=4$ ,  $N_c=8$ ,  $N_x=256$ ,  $N_y=256$ ,  $N_f=118$ ,  $N_f=2000$ . In this case, calculating the weights in hybrid space requires about 50 times more computation than the equivalent operation in k-space. However, performing the application phase in hybrid space takes only 1/5<sup>th</sup> the computation as the equivalent operation in k-space. In this case, and in most cases, the computation to convert the weights from k-space to hybrid space is negligible. While removing the aliasing artifact in image space can be accomplished in about 1/9<sup>th</sup> the time required in k-space, the computation required to convert the weights to image space is not negligible, requiring nearly half as much time as it takes to simply apply the weights in k-space. Furthermore, removing the aliasing in image space requires a uniform k-space sampling density, a condition that can only be achieved with regularly undersampled data from which the calibration lines have been removed, resulting in reduced SNR and the inability to achieve flexible sampling patterns.

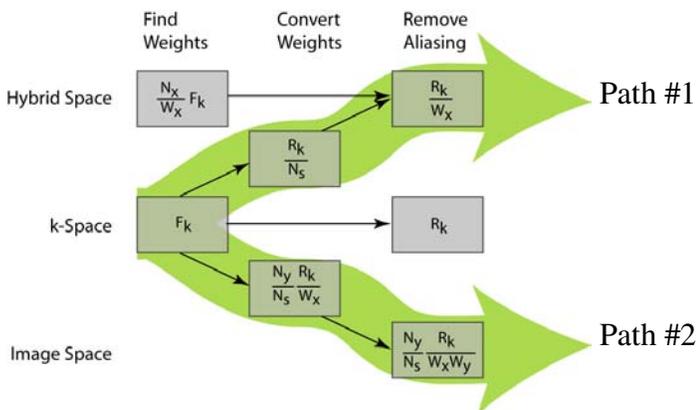
**Discussion & Conclusion:** In all cases, the training phase is most efficiently performed in k-space. For the application phase, when aliasing artifacts need to be removed from only one image, it is computationally more efficient to apply the weights in hybrid space (Path #1 in Fig. 2). Performing this task in hybrid space also has the advantage over image space that the calibration lines do not need to be removed. Figure 3 provides a schematic for how Path #2 can be implemented in practice, using a simplified example with data from just 2 coils. In the case of time-series imaging, however, where the calibration lines are acquired once at the beginning of imaging and used to remove the aliasing artifacts on a number of subsequent images, then removing the aliasing artifacts in image space becomes the most attractive method from a computation standpoint (Path #2 in Fig. 2).

These results show that the autocalibrating reconstruction path should be tailored to the specific imaging application to minimize total reconstruction time. While this work examined 1D-accelerated parallel imaging implementations only, the conclusions also have important implications for 2D-accelerated parallel imaging reconstructions where the computation burden increases significantly.

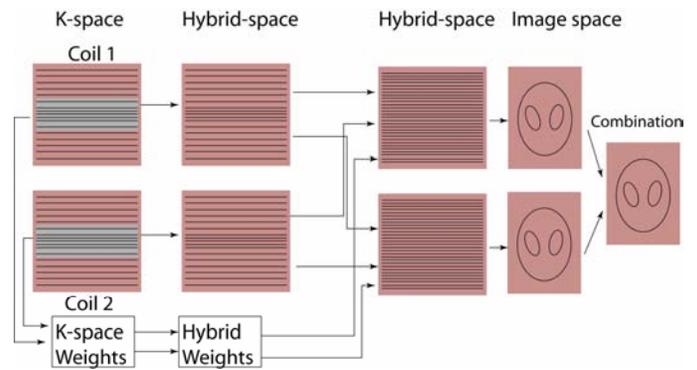
**References:** [1] Griswold MA *et al.* MRM 47:1202-1210. [2] Griswold MA *et al.* MRM 52:1118-1126. [3] Kholmovski EG *et al.* ISMRM 2005, 2672. [4] Qu P *et al.* ISMRM 2005, 2667. [5] Wang Z *et al.* MRM 54:738-742, 2005. [6] Skare S *et al.* ISMRM 2005, 2422. [7] Wang J *et al.* ISMRM 2005, 2428.



**Figure 1.** 2D k-space kernel used to find the weights to synthesize missing data at point “x”.  $W_x$  and  $W_y$  are the kernel widths in the  $k_x$  and  $k_y$  direction, respectively.



**Figure 2.** Block diagram of the possible processing pathways to reconstruct autocalibrated data with a 2D kernel. The relative computational requirement for each step is indicated within each block. The most computationally efficient pathway for most applications is indicated by Path #1. Path #2 is also more efficient than a reconstruction performed entirely in k-space, but is less flexible and thus restricted in its applicability.



**Figure 3.** Practical implementation of Path #1, using a simplified example with data from 2 coils. Kernel weights are determined from calibration lines in k-space, converted to hybrid space via 1D FT, and then applied to data that has also been transformed into hybrid space to synthesize missing lines. Performing another 1D FT on the reconstructed hybrid space data produces unaliased images on a per coil basis, which can then be combined (e.g. using sum-of-squares) to create the final image.