Fast ℓ_1-SPIRiT Compressed Sensing Parallel Imaging MRI: Scalable Parallel Implementation and Clinically Feasible Runtime

Mark Murphy, Marcus Alley, James Demmel, Kurt Keutzer, Shreyas Vasanawala, and Michael Lustig

Abstract

We present ℓ_1-SPIRiT, a simple algorithm for auto calibrating parallel imaging (acPI) and compressed sensing (CS) that permits an efficient implementation with clinically-feasible runtimes. We propose a CS objective function that minimizes cross-channel joint sparsity in the Wavelet domain. Our reconstruction minimizes this objective via iterative soft-thresholding, and integrates naturally with iterative Self-Consistent Parallel Imaging (SPIRiT). Like many iterative MRI reconstructions, ℓ_1-SPIRiT's image quality comes at a high computational cost. Excessively long runtimes are a barrier to the clinical use of any reconstruction approach, and thus we discuss our approach to efficiently parallelizing ℓ_1-SPIRiT and to achieving clinically-feasible runtimes. We present parallelizations of ℓ_1-SPIRiT for both multi-GPU systems and multi-core CPUs, and discuss the software optimization and parallelization decisions made in our implementation. The performance of these alternatives depends on the processor architecture, the size of the image matrix, and the number of parallel imaging channels. Fundamentally, achieving fast runtime requires the correct trade-off between cache usage and parallelization overheads. We demonstrate image quality via a case from our clinical experimentation, using a custom 3DFT Spoiled Gradient Echo (SPGR) sequence with up to 8$\times$ acceleration via Poisson-disc undersampling in the two phase-encoded directions.

Key words: Autocalibrating Parallel Imaging, SPIRiT, Compressed Sensing, GPGPU, Parallel Computing

1 Introduction

Imaging speed is a major limitation of MR Imaging, especially in comparison to competing imaging modalities such as Computed Tomography (CT). MR allows much more flexible contrast-generation and does not expose patients to ionizing radiation, and hence does not increase risk of cancer. However, other imaging modalities are substantially more popular, as MR scans are slow, expensive, and in some cases less robust. Patient motion during long scans frequently causes image artifacts, and for uncooperative patients, like children, anesthesia is a frequent solution. Acquisition time in MRI can be reduced by faster scanning or by subsampling. Parallel imaging [1, 2, 3] is a well-established acceleration technique based on the spatial sensitivity of array receivers. Compressed sensing (CS) [4, 5, 6] is an emerging acceleration technique that is based on the compressibility of medical images. Attempts to combine the two have mostly focused on extensions of iterative SENSE [7] with SparseMRI [6]. In [8] Block *et al.*, added total-variation to a SENSE reconstruction from radial sampling, Liang *et al.*, in [9] showed improved acceleration by first performing CS on aliased images and then applying SENSE to unfold the aliasing, Otazo *et al.* used compressed sensing with SENSE to accelerate first-pass cardiac perfusion [10]. More recently [11, 12] have presented some improvements, again, using an extension of SENSE. The difficulty in estimating exact sensitivity maps in SENSE has created the need for autocalibrating techniques. One class of autocalibrating algorithms extends the SENSE model to joint estimation of the images and the sensitivity maps [13, 14]. Combination of these approaches with compressed sensing have also been proposed. Knoll *et al.* [15] proposed a combination with Uecker's non-linear inversion and Huang *et al.* [16] proposed a self-feeding SENSE combined with compressed sensing.

A different, yet very popular class of autocalibrating techniques are methods like GRAPPA [3] that do not use the sensitivity maps explicitly. In [17] we proposed an optimized iterative method, SPIRiT, and demonstrated the combination with non-linear regularization. In [18] we presented and extension, ℓ_1-SPIRiT, that synergistically combines SPIRiT with compressed sensing and in [19, 20] we presented more details and clinical results in pediatric patients.

The combination of compressed sensing with parallel imaging has the advantage of improved image quality, however it comes at a cost. These algorithms involve substantially more computation than direct or iterative linear reconstructions.

In this paper we discuss the ℓ_1-SPIRiT reconstruction. ℓ_1-SPIRiT solves a constrained non-linear optimization over the image matrix. The non-linearity of this optimization necessitates an iterative reconstruction, and we describe our simple and efficient POCS algorithm in Secion 3.

A recent trend in MRI has been to accelerate reconstructions by implementing and optimizing them for massively parallel processors. Silicon manufacturing technology has recently experienced the end of a trend that produced the incredible pace of comptuational speed during the 1990's [21]. In the past decade, all major microprocessor vendors have increased the computational throughput of their designs by introducing programmer-visible parallelism. Intel and AMD provide 4-16 CPU cores per socket, and GPGPUs typically have 16-32 massively multithreaded vector cores per socket. In each case, the computational throughput of the processor is proportional to the number of cores, and future designs will have larger numbers of cores.

This paper discusses the massively parallel implementation of ℓ_1-SPIRiT on these processors. The resulting sub-minute runtimes demonstrate that computational expense is not a substantial obstacle to clinical deployment of ℓ1-SPIRiT. Many previous works have demonstrated substantial improvement in reconstruction runtime using GPUs and multi-core CPUs as parallel execution platforms. Chang and Ji [22] demonstrated multi-channel acceleration by solving SparseMRI reconstruction separately for each channel and reporting 1.6-2.0 acceleration using 4 cores. More recently Kim *et al.* [23] present a high-performance implementation of a SENSE based compressive sensing reconstruction, describing many low-level optimizations that apply for both CPU and GPU architectures.

Stone *et al.* [24] describe the implementation of an iterative reconstruction using the Conjugate Gradient (CG) algorithm to solve regularized linear reconstructions for non-Cartesian trajectories. Their implementation relies on a highly optimized GPU implementation of a non-uniform Fourier transform (NDFT) to perform subminute non-Cartesian reconstructions. Wu *et al.* [25, 26] have generalized this work to model other acquisition effects in the NDFT, such as off-resonance and sensitivity encoding. Several other works have discussed the GPU implementation of Gridding [27], a highly accurate NDFT approximation. Sørensen *et al.* [28] describe an algorithm for obviating potentially expensive synchronization in a GPGPU implementation of Gridding. Obeid *et al.* [29] use a spatial-partitioning approach to optimize gridding interpolation, and report 1-30 second runtimes. Nam *et al.* [30] describe another gridding implementation achieving sub-second interpolations for highly undersampled data. Several other works have presented GPU implementations of Parallel Imaging (PI) reconstructions with clinically-feasible runtimes. Roujol *et al.* [31] describe GPU implementation of temporal sensitivity encoding (TSENSE) for 2D interventional imaging. Sørensen *et al.* [32] present a fast iterative SENSE implementation which performs 2D gridding on GPUs. Uecker [14] describes a GPU implementation of a non-linear approach to estimate PI coil sensitivity maps during image reconstruction.

This work presents the parallelization of an autocalibrating approach, ℓ1-SPIRiT, via multi-core CPUs and GPUs and the resulting clinically-feasible reconstruction runtimes. Moreover we discuss the approach taken to parallelizing the various operations within our reconstruction, and the performance trade-offs in different parallelization strategies. Additionally, we discuss the data-size dependence of performance-relevant implementation decisions. To our knowledge, no previous works have addressed this issue.

2 iTerative Self-Consistent Parallel Imaging Reconstruction (SPIRiT)

SPIRiT is a coil-by-coil autocalibrating parallel imaging method and is described in detail in [17]. SPIRiT is similar to the GRAPPA parallel imaging method in that it uses autocalibration lines to find linear weights to synthesize missing k-space. The SPIRiT model is based on self-consistency of the reconstructed data with the acquired k-space data and with the calibration.

SPIRiT is an iterative algorithm in which in each iteration non-acquired k-space values are estimated by performing a linear combination of nearby k-space values. The linear combination is performed using both acquired k-space samples as well as estimated values (from the previous iteration) for the non-acquired samples. If we denote x_i as the entire k-space grid of the i^{th} coil, then the consistency criterion has a form of a series of convolutions with the so called SPIRiT kernels g_{ij}. The SPIRiT kernels are obtained by calibration from auto calibration lines similarly to GRAPPA. If N_c is the total number of channels, the calibration consistency criterion can be written as

$$x_i = \sum_{j=1}^{Nc} g_{ij} * x_j.$$

The SPIRiT calibration consistency for all channels can be simply written in matrix form as

$$x = Gx,$$

where x is a vector containing the concatenated multi-coil data and G is an aggregated operator that performs the appropriate convolutions with the g_{ij} kernels and the appropriate summations. As discussed in [17], the G operator can be implemented as a convolution in k-space or as multiplication in image space.

In addition to consistency with the calibration, the reconstruction must also be consistent with the acquired data Y. This can be simply written as

$$y = Dx,$$

where D is an operator that select the acquired k-space out of the entire k-space grid. In [17] two methods were proposed to find the solution that satisfies the constraints. Here we would like to point out the projection over convex sets (POCS) approach which uses alternate projections that enforce the data consistency and calibration consistency. In this paper we extend the POCS approach to include sparsity constraints for combination with compressed sensing.

As previously mentioned, the convolution kernels $g_{i,j}$ are obtained via a calibration from the densely sampled auto-calibration region in the center of k-space, commonly referred to as the Auto-Calibration Signal or ACS lines. In the reconstruction we would like to find x that satisfies $x = Gx$. However in the calibration x is known and G is unknown. We compute the calibration in the same way as GRAPPA [3], by fitting the kernels g to the consistency criterion $x = Gx$ in the ACS lines. Due to noise, data corruption, and ill-conditionedness, we solve this fit in an ℓ_2-regularized least-squares sense. We compute the calibration once, prior to image reconstruction. Calibration could potentially be improved via a joint estimation of the kernels and images, as has been presented by Zhao *et al.* as Iterative GRAPPA [33] and in SENSE-like models by Ying *et al.* [13] and Uecker *et al* [14]. Joint estimation is more computationally expensive, but its runtime could be improved with techniques similar to those discussed in this papar.

3 $\ell 1$-SPIRiT Reconstruction

Variations of the $\ell 1$-SPIRiT reconstruction have been mentioned in several conference proceedings [18, 34, 35, 19]. More detailed descriptions are given in [17] and in [20]. But for the sake of completeness and clarity we include here a detailed description of the variant that is used in this paper.

$\ell 1$-SPIRiT is an approach for accelerated sampling and reconstruction that synergistically unifies compressive sensing with auto-calibrating Parallel imaging. The sampling is optimized to provide the incoherence that is required for compressed sensing yet compatible to parallel imaging. The reconstruction is an extension of the original SPIRiT algorithm that in addition to enforcing consistency constraints with the calibration and acquired data, enforces joint-sparsity of the coil images in the Wavelet domain. Let y be a the vector of acquired k-space measurements from all the coils, F a Fourier operator applied individually on each coil-data, D a subsampling operator that chooses only acquired k-space data out of the entire k-space grid, G an image-space SPIRiT operator that was obtained from auto-calibration lines, Ψ a wavelet transform that operates on each individual coil separately. ℓ_1-SPIRiT solves for the multi-coil images concatenated into the vector x which minimizes the following problem:

$$\begin{aligned} \text{minimize}_x \quad & \text{Joint}\ell_1(\mathbf{\Psi}x) & (1)\\ \text{subject to} \quad & \mathbf{DF}x = y & (2)\\ & \mathbf{G}x = x & (3) \end{aligned}$$

The function $\text{Joint}\ell_1(\cdot)$ is a joint ℓ_1-ℓ_2-norms convex functional and is described later in more detail. Minimizing the objective (1) enforces joint sparsity of wavelet coefficients between the coils. The constraint in (2), is a linear data-consistency constraint and in (3) is the SPIRiT parallel imaging consistency constraint. The Wavelet transform [36] $\mathbf{\Psi}$ is well-known to sparsify natural images, and thus used frequently in Compressive Sensing applications as a sparsifying basis. Just as the Fourier transform, it is a linear operation that can be computed via a fast $O(n \log n)$ algorithm.

As previously mentioned, in this work we solve the above problem via a an efficient POCS algorithm, shown in Figure 1. This algorithm does not solve the constrained minimization exactly, but instead minimizes the related Lagrangian objective function. The POCS algorithm converges to a fixed-point that satisfies the above constraints, often within 50-100 iterations.

3.1 Joint-Sparsity of Multiple Coils

We perform soft-thresholding on the Wavelet coefficients to minimize the $\ell 1$-objective function (1). The soft-thresholding function $\mathcal{S}_\lambda(x)$ is defined element-wise for $x \in \mathbb{C}$ as:

$$\mathcal{S}_\lambda(x) = \frac{x}{|x|} \cdot \max(0, |x| - \lambda)$$

where $|x|$ is the complex modulus of x. The parameter λ estimates the amplitude of noise and aliasing in the Wavelet basis, and the soft-thresholding operation is a well-understood component of many denoising [37] and compressive sensing algorithms [38]. We use a randomized shifting technique to approximate translation-invariant Wavelets [19, 39], and this has a negligible computational overhead.

The individual coil images are sensitivity weighted images of the original image of the magnetization. We assume that coil sensitivities are smooth and do not produce spatial shift of one coil's image relative to another.

```
 $x_k$ - image estimate after $k^{\text{th}}$ iteration
 $y$ - acquired data
 $F$ - multi-coil Fourier transform operator
 $G$ - SPIRiT operator
 $\Psi$ - multi-coil Wavelet transform operator
 $D$ - subsampling operator choosing acquired data
 $\mathcal{S}_\lambda$ - Joint Soft-thresholding


 $\mathbf{G} \leftarrow \text{AutoCalibrate}(y)$
 Initialize $x_0 \leftarrow \mathbf{F^{-1}} D^T y$
 for $k = 1, 2, \ldots$ until convergence:
(A)    $m_k \leftarrow \mathbf{G} x_{k-1}$
(B)    $w_k \leftarrow \mathbf{\Psi^{-1}} \mathcal{S}_\lambda \{\mathbf{\Psi} m_k\}$
(C)    $x_k \leftarrow \mathbf{F}^{-1} \left[(I - D^T D)(\mathbf{F} w_k) + D^T y\right]$
```

Figure 1: The POCS algorithm. Line (A) performs SPIRiT k-space interpolation, implemented as voxel-wise matrix-vector multiplications in the image domain. Line (B) performs Wavelet Soft-thresholding, computationally dominated by the forward/inverse Wavelet transforms. Line (C) performs the k-space consistency projection, dominated by inverse/forward Fourier transforms.

Thus, edges in these images appear in the same spatial position, and therefore coefficients of sparse transforms, such as wavelets, exhibit similar sparsity patterns. To exploit this, we use a joint-sparsity model citeISBI11,Vasanawala:2010fk. In compressed sensing, sparsity is enforced by minimizing the ℓ_1-norm of a transformed image. The usual definition of the ℓ_1-norm is the sum of absolute values of all the transform coefficients, $\sum_c \sum_r |w_{cr}| = \sum_c \sum_r \sqrt{|w_{rc}|^2}$, where c is the coil index and r is the spatial index. In a joint-sparsity model we would like to jointly penalize coefficients from different coils that are at the same spatial position. Therefore we define a joint ℓ_1 as:

$$\text{Joint}\ell_1(w) = \sum_r \sqrt{\sum_c |w_{rc}|^2}$$

In a joint ℓ_1-norm model, the existence of large coefficient in one of the coils, protects the coefficients in the rest of the coils from being suppressed by the non-linear reconstruction. In the POCS algorithm joint sparsity is enforced by soft-thresholding the magnitude of the wavelet coefficients across coils, at a particular position.

3.2 Computational Complexity

If n_c is the number of PI channels and v is the number of voxels per PI channel, the computational complexity of our algorithm is:

$$O\left(C_C \cdot n_c^3 + T \cdot ((C_W + C_F) \cdot n_c v \log v + C_S \cdot n_c^2 v)\right)$$

T is the number of iterations the POCS algorithm performs. The algorithm often converges with sufficient accuracy within 50-100 iterations. The constants C_W, C_F, C_S, and C_C indicate that the relative computational cost of the Wavelet transforms, Fourier transforms, and SPIRiT interpolation and calibration are heavily dependent on input data size. Section 6 presents more detailed runtime data.

The n_c^3 term represents the SPIRiT calibration, which performs a least-norm least-squares fit of the SPIRiT model to a densely sampled autocalibration region in the center of k-space. Solving each of these systems independently leads to an $O(n_c^4)$ algorithm, which is prohibitively expensive for large coil arrays. Appendix A describes an algorithm that reduces this complexity to $O(n_c^3)$ by re-using a single Cholesky factorization for all channels. This derivation can potentially be used to accelerate the computation of GRAPPA kernels as well.

The $n_c v \log v$ term represents the Fourier and Wavelet transforms, and the $n_c^2 v$ term represents the image-domain implementation of the k-space SPIRiT interpolation. This k-space convolution is implemented as multiplication in the image domain, hence the linearity in v of this term. Due to the $O(n_c^2 v)$ complexity, SPIRiT interpolation is asymptotically the bottleneck of the POCS algorithm. All other operations are linear in the number of PI channels, and at worst log-linear in the number of voxels per channel. There are several proposed approaches that potentially reduce the complexity of the SPIRiT interpolation without degrading image quality. For example ESPIRiT [35] performs an eigendecomposition of the **G** matrix, and uses a rank-one approximation during POCS iterations. Also, coil array compression [40, 41] can reduce the number of parallel imaging channels to a small constant number of *virtual* channels. Our software includes implementations of both of these approaches, and in practice the ℓ1-SPIRiT solver is rarely run with more than 8 channels.

One could solve the ℓ_1-SPIRiT reconstruction problem (Eqns 1-3) via an algorithm other than our POCS approach, for example non-linear Conjugate Gradients (NLCG). The computational complexity of alternate algorithmic approaches would differ only in constant factors. The same set of computations would still dominate runtime, but a different number of iterations would be performed and potentially a different number of these operations would be computed per iteration. End-to-end reconstruction times would differ, but much of the performance analysis in this work applies equally well to alternate algorithmic approaches.

4 Fast Implementation

The POCS algorithm is efficient: in practice, it converges rapidly and performs a minimal number of operations per iteration. Still, a massively parallel and well-optimized implementation is necessary to achieve clinically feasible runtimes. A sequential C++ implementation runs in about 10 minutes for the smallest reconstructions we discuss in this paper, and in about 3 hours for the largest. For clinical use, images must be available immediately after the scan completes in order to inform the next scan to be prescribed. Moreover, time with a patient in a scanner is limited and expensive: reconstructions requiring more than a few minutes of runtime are infeasible for on-line use.

In this section, we discuss the aspects of our reconstruction implementation pertinent to computational performance. While previous works have demonstrated the suitability of parallel processing for accelerating MRI reconstructions, we provide a more didactic and generalizable description intended to guide the implementation of other reconstructions as well as to explain our implementation choices.

4.1 Parallel Processors

Many of the concerns regarding efficient parallel implementation of MRI reconstructions are applicable to both CPU and GPU architectures. These two classes of systems are programmed using different languages and tools, however have much in common. Figure 2 establishes a four-level hierarchy that one can use to discuss parallelization decisions. In general, synchronization is more expensive and aggregate data access bandwidth is less at "higher" levels of the hierarchy (i.e. towards the top of Figure 2). For example, Cuda GPGPUs can synchronize threads within a core via a `__syncthreads()` instruction at a cost of a few processor cycles, but synchronizing all threads within a GPU requires ending a grid launch at a cost of $\approx 5\mu s$, or 7,500 cycles. CPU systems provide less elaborate hardware-level support for synchronization of parallel programs, but synchronization costs are similar at the corresponding levels of the processor hierarchy. On CPUs, programs synchronize via software barriers and task queues implemented on top of lightweight memory-system support. With respect to data access, typical systems have $\approx$ 10 TB/s (10^{13} bytes/s) aggregate register-file bandwidth, but only $\approx$ 100 GB/s (10^{11} bytes/s) aggregate DRAM bandwidth. Exploiting locality and data re-use is crucial to performance.

In this work, we do not further discuss cluster-scale parallelization (among Nodes in Figure 2). The CPU parallelization we'll describe in this section only leverages the parallelism among the multiple Sockets/Cores a single Node. As indicated by Figure 2, parallelization decisions at this level are analogous to decisions among the multiple GPUs in a single system, but we leave more detailed performance analysis of cluster-parallelization to future work.

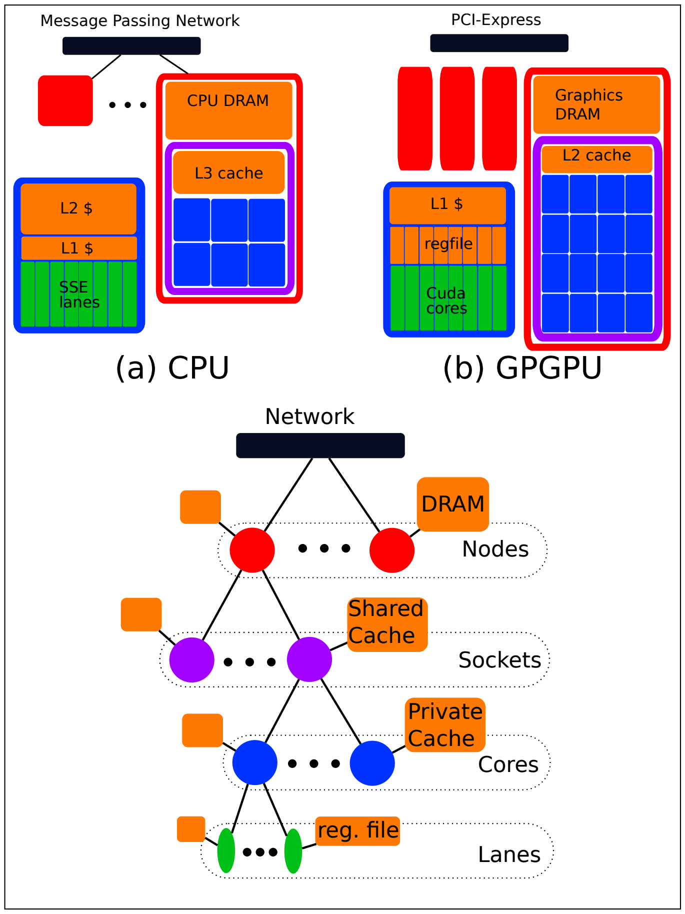

Figure 2: The four-level hierarchy of modern parallel systems. *Nodes* contain disjoint DRAM address spaces, and communicate over a message-passing network in the CPU case, or over a shared PCI-Express network in the GPU case. *Sockets* within a node (only one shown) share DRAM but have private caches – the L3 cache in CPU systems and the L2 cache in Fermi-class systems. Similarly *Cores* share access to the Socket-level cache, but have private caches (CPU L2, GPU L1/scratchpad). Vector-style parallelism within a core is leveraged via *Lanes* – SSE-style SIMD instructions on CPUs, or the SIMT-style execution of GPUs.

4.2 Data-Parallelism and Geometric Decomposition

The computationally intense operations in MRI reconstructions contain nested data parallelism. In particular, operations such as Fourier and Wavelet transforms

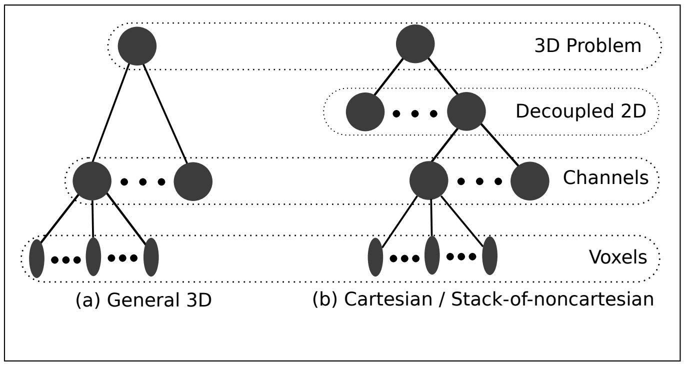

Figure 3: Hierarchical sources of parallelism in 3D MRI Reconstructions. For general reconstructions, operations are parallelizable both over channels and over image voxels. If there is a fully-sampled direction, for example the readout direction in Cartesian acquisitions, then decoupling along this dimension allows additional parallelization.

are performed over k-dimensional slices through the N-dimensional reconstruction volume, with $k < N$. In most cases the operations are performed for *all* k-dimensional (k-D) slices, providing another source of parallelism to be exploited for accelerating the reconstruction. The k-D operations themselves are parallelizable, but usually involve substantial synchronization and data-sharing. Whenever possible, it is very efficient to exploit this additional level of parallelism. For the purposes of software optimization, the size and shape of the N-dimensional (N-D) data are important. The Geometric Decomposition (GD) design pattern [42] discusses the design of parallel programs in which the data involved have geometric structure. GD suggests the parallelization should follow a division of the data that follows this structure, in order to achieve good caching and inter-thread communication behavior.

Recall from Figure 2 that modern processor architectures provide four levels at which to exploit parallelism. An efficient parallel implementation must decide at which levels of the processor hierarchy to exploit the levels of the nested parallelism in MRI reconstruction.

In volumetric MRI reconstructions, all of these operations are applied to the 4-D array representing the multi-channel 3D images. Figure 3 illustrates that the exploitable parallelism of operations over these arrays is two-level: operations like Fourier and Wavelet transforms applied to the individual channels' images exhibit massive voxel-wise parallelism and require frequent synchronization; but the transforms of the 4-32 channels can be performed independently and in parallel.

In Cartesian acquisitions, the readout direction is never randomly subsampled. Similarly in stack-of-spirals or stack-of-radial acquisitions, the same non-Cartesian sampling of $x - y$ slices is used for every z position. In these cases, the 3D reconstruction can be decoupled into independent 2D reconstructions for each undersampled slice. The resulting reconstruction is not SNR-optimal, since noise is only averaged over the samples within each 2D slice. Additionally, the compressive sensing reconstruction is unable to exploit cross-slice Wavelet-domain sparsity. However, decoupling can provide a substantial performance benefit. Parallelizing over independent 2D reconstructions is very efficient, as the decoupled 2D reconstructions require no synchronization. Our Cuda ℓ_1-SPIRiT solver is able to run multiple 2D problems simultaneously per GPU in batch mode. Large batch sizes require more GPU memory, but expose more parallelism and can more effectively utilize the GPU's compute resources.

4.3 Size-Dependence and Cache/Synchronization Trade-off

One can produce several functionally equivalent implementations by parallelizing at different levels of the hierarchy in Figure 2. These different implementations will produce identical results[1], but have very different performance characteristics. Moreover, the performance of a given implementation may differ substantially for different image matrix sizes and coil array sizes. In general, the optimal implementation is a trade-off between effective use of the cache/memory hierarchy and amortization of parallelization overheads.

For example, one may choose to exploit the voxel-wise parallelism in an operation only among the vector lanes within a single processor core. The implementation can then exploit parallelism over multiple channels and 2D slices over the multiple cores, sockets, and nodes in the system. Utilizing this additional parallelism will increase the memory footprint of the algorithm, as the working set of many 2D slices must be resident simultaneously. This consideration is particularly important for GPU systems which have substantially less DRAM capacity than CPU systems.

On the other hand, one may leverage voxel-wise parallelism among the multiple cores within a socket, the multiple sockets within the system, or among the multiple nodes. In doing so the implementation is able to exploit a larger slice of the system's processing and memory-system resources while simultaneously reducing memory footprint and working-set size. The favorable caching behavior of the smaller working set may result in a more efficient implementation. However it is more expensive to synchronize the higher levels of the processing hierarchy. Furthermore for problems with smaller matrix sizes,

[1]Identical up to round-off differences in floating point arithmetic, which is not always associative or commutative

voxel-wise parallelism may be insufficient to fully saturate the processing resources at higher levels. Even when caching behavior is more favorable, this over-subscription of resources may degrade performance.

Which implementation provides better performance depends both on the size of the input data and on the size of the processor system. When the image matrix is very high-resolution (i.e. has a large number of voxels) or the processing system is relatively small (i.e. a small number of processor cores), then one can expect a high degree of efficiency from exploiting voxel-wise parallelism at higher levels of the hierarchy. If the image matrix is relatively small or the processor system is very large, then one should expect that parallelism from the Channel and Decoupled-2D levels of Figure 3 is more important. As the number of processing cores per system and the amount of cache per core both continue to increase over time, we expect the latter case to become more common in the future.

4.4 Parallel Implementation of ℓ_1-SPIRiT

In the case of ℓ_1-SPIRiT, there are four operations which dominate runtime: SPIRiT auto-calibration, Fourier transforms during the k-space consistency projection, Wavelet transforms during the joint soft-thresholding, and the image-domain implementation of SPIRiT interpolation. Figure 4 depicts the overall flow of the iterative reconstruction. Note that PI calibration must be performed only once per reconstruction, and is not part of the iterative loop.

SPIRiT Auto-Calibration Our ℓ_1-SPIRiT implementation performs auto-calibration by fitting the SPIRiT consistency model to the densely sampled Auto-Calibration Signal (ACS), which requires solving a least-squares least-norm problem for each PI channel. Note that while we perform the POCS iterations over decoupled 2D slices, we perform calibration in 3D k-space. The SPIRiT interpolation kernels for the 2D problems are computed via an inverse Fourier transform in the readout direction.

As discussed in Appendix A, the auto-calibration is computationally dominated by two operations: the computation of a rank-k matrix product A^*A and a Cholesky factorization $A = LL^*$, which itself is dominated by rank-k products. Numerical linear algebra libraries for both CPUs and GPUs are parallelized via an output-driven scheme that requires very little inter-thread synchronization. For example, when computing a matrix-matrix product $C = AB$ each element $c_{i,j}$ is computed as an inner product of a row a^i of A with a column b_j of B. All such products can be computed independently in parallel, and are typically blocked to ensure favorable cache behavior.

The SPIRiT Operator in Image Space Figure 4 (b) illustrates the image-domain implementation of SPIRiT interpolation $\mathbf{G}x$. $\mathbf{G}x$ computes a matrix-vector multiplication per voxel – the length n_c (# PI channels) vector is composed of the voxels at a given location in all PI channels. For efficiency in the Wavelet and Fourier transforms, each channel must be stored contiguously – thus the cross-channel vector for each voxel is non-contiguous. Our implementation of the interpolation streams through each channel in unit-stride to obviate inefficient long-stride accesses or costly data permutation. The image-domain implementation is substantially more efficient than the k-space implementation, which performs convolution rather than a multiplication. However, the image-domain representation of the convolution kernels requires a substantially larger memory footprint, as the compact k-space kernels must be zero-padded to the image size and Fourier transformed. Since SPIRiT's cross-coil interpolation is an all-to-all operation, there are n_c^2 such kernels. This limits the applicability of the image-domain SPIRiT interpolation when many large-coil-array 2D problems are in flight simultaneously. This limitation is more severe for the Cuda implementation than the OpenMP implementation, as GPUs typically have substantially less memory capacity than the host CPU system.

Enforcing Sparsity by Wavelet Thresholding Figure 4 (c) illustrates Wavelet Soft-thresholding. Similarly to the Fourier transforms, the Wavelet transforms are performed independently and in parallel for each channel. Our Wavelet transform implementation is a multi-level decomposition via a separable Daubechies 4-tap filter. Each level of the decomposition performs low-pass and high-pass filtering of both the rows and columns of the image. The number of levels of decomposition performed depends on the data size: we continue the wavelet decomposition until the approximation coefficients are smaller than the densely sampled auto-calibration region. Our OpenMP implementation performs the transform of a single 2D image in a single OpenMP thread, and parallelizes over channels 2D slices.

We will present performance results for two alternate GPU implementations of the Wavelet transform. The first parallelizes a 2D transform over multiple cores of the GPU, while the second is parallelized only over the vector lanes within a single core. The former is a finer-grained parallelization with a small working set per core, and permits an optimization that greatly improves memory system performance. As multiple cores share the transform for a single 2D transform, the per-core working set fits into the small l1-cache of the GPU. Multiple Cuda thread blocks divide the work of the convolutions

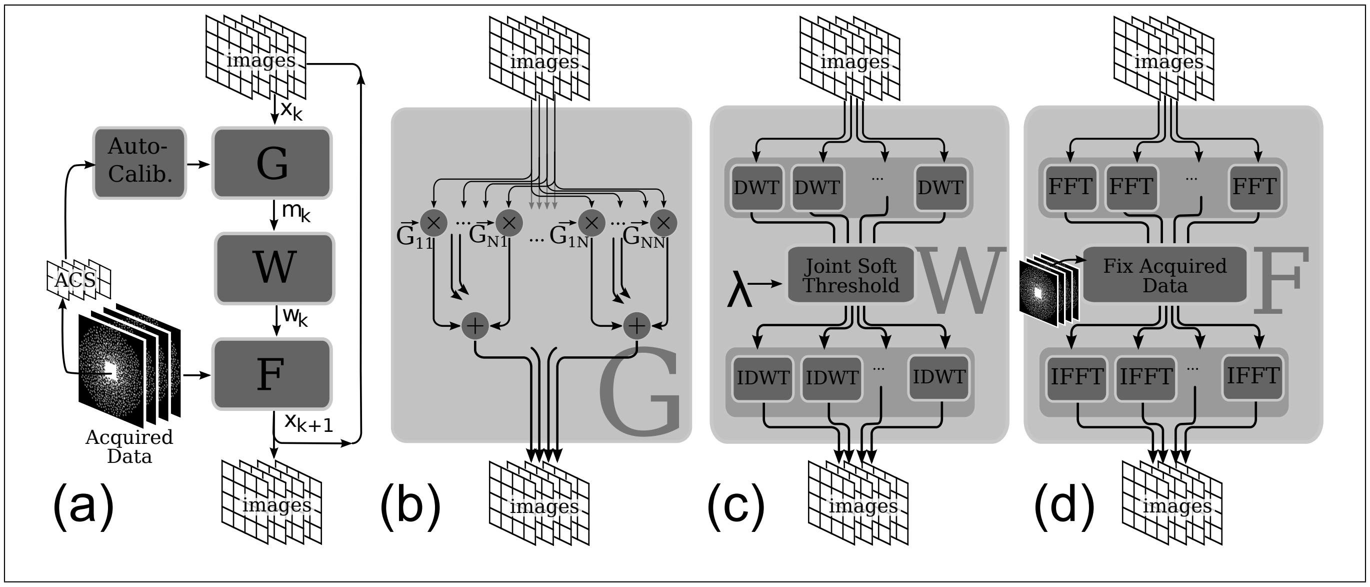

Figure 4: (a) Flowchart of the ℓ_1-SPIRiT POCS algorithm and the (b) SPIRiT, (c) Wavelet joint-threshold and (d) data-consistency projections

for each channel's image, and explicitly block the working data into the GPU's local store. Parallelism from multiple channels is exploited among multiple cores when a single 2D transform cannot saturate the entire GPU. The latter parallelization exploits all voxel-wise parallelism of a 2D transform within a single GPU core, and leverages the channel-wise and slice-wise parallelism across multiple cores. The working set of a single 2D image does not fit in the l1 cache of the GPU core and we cannot perform the explicit blocking performed in the previous case.

Enforcing k-space acquisition consistency Figure 4 (d) illustrates the operations performed in the k-space consistency projection. The runtime of this computation is dominated by the forward and inverse Fourier transforms. As the FFTs are performed independently for each channel, there is n_c-way embarrassing parallelism in addition to the voxel-wise parallelism within the FFT of each channel. FFT libraries typically provide APIs to leverage the parallelism at either or both of these levels. The 2D FFTs of phase-encode slices in ℓ_1-SPIRiT are efficiently parallelized over one or a few processor cores, and FFT libraries can very effectively utilize voxel-wise parallelism over vector lanes. We will present performance results for the GPU using both the `Plan2D` API, which executes a single 2D FFT at a time, and the `PlanMany` API which potentially executes many 2D FFTs simultaneously. The latter approach more easily saturates the GPU's compute resources, while the former approach is a more fine-grained parallelization with potentially more efficient cache-use.

Dataset	n_x	n_y	n_z	n_c
A	192	256	58	8, 16, 32
B	192	256	102	8, 16, 32
C	192	256	152	8, 16, 32
D	192	256	190	8, 16, 32
E	320	260	250	8, 16, 32
F	320	232	252	8, 16, 32

Table 1: Table of dataset sizes for which we present performance data. n_x is the length of a readout, n_y and n_z are the size of the image matrix in the phase-encoded dimensions, and n_c is the number of channels in the acquired data. Performance of SPIRiT is very sensitive to the number of channels, so we present runtimes for the raw 32-channel data as well as coil-compressed 8- and 16-channel data.

5 Methods

Section 6 presents performance results for a representative sample of datasets from our clinical application [19]. We present runtimes for a constant number of POCS iterations only, so the runtime depends only on the size of the input matrix. In particular, our reported runtimes do not depend on convergence rates or the amount of scan acceleration. We present performance results for six datasets, whose sizes are listed in Table 1.

We present several performance metrics of interest. First, we shall discuss the end-to-end runtime of our reconstruction to demonstrate the amount of wall-clock time the radiologist must wait from the end of the scan until the images are available. This includes the PI calibration, the POCS solver, and miscellaneous supporting operations. To avoid data-dependent performance differences due to differing convergence rates, we present runtime for a constant (50) number of POCS iterations.

To demonstrate the effectiveness of our $O(n^3)$ calibration algorithm, we compare its runtime to that of the "obvious" implementation which uses ACML's implementation of the Lapack routine `cposv` to solve each coil's calibration independently. The runtime of calibration does not depend on the final matrix size, but rather on the number of PI channels and the number of auto-calibration readouts. We present runtimes for calibrating $7 \times 7 \times 7$ kernels averaged over a variety of ACS sizes.

We also present per-iteration runtime and execution profile of the POCS solver for several different parallelizations, including both CPU and GPU implementations. The per-iteration runtime does not depend on the readout length or the rate of convergence. Since we decouple along the readout dimension, POCS runtime is simply linear in n_x. Presenting per-iteration runtime allows direct comparison of the different performance bottlenecks of our multiple implementations.

Additionally, we explore the dependence of performance on data-size by comparing two alternate implementations parallelized for the GPU. The first exploits voxel-wise parallelism and channel-wise parallelism at the Socket-level from Figure 2, and does not exploit Decoupled-2D parallelism. This implementation primarily synchronizes via ending Cuda grid launches, incurring substantial overhead. However, the reduced working-set size increases the likelihood of favorable cache behavior, and enables further caching optimizations as described in Section 4.4. Fourier transforms are performed via the 2D API, which expresses a single parallel FFT per grid launch. Fermi-class GPUs are able to execute multiple grid launches simultaneously, thus this implementation expresses channel-wise parallelism as well. The second implementation exploits voxel-wise parallelism only within a core of the GPU, and maps the channel-wise and Decoupled-2D parallelism at the Socket-level. This implementation is able to use the more efficient within-core synchronization mechanisms, but has a larger working set per core and thus cannot as effectively exploit the GPU's caches. It also launches more work simultaneously in each GPU grid launch than does the first implementation, and can more effectively amortize parallelization overheads. Fourier transforms are performed via the `PlanMany` API, which expresses the parallelism from all FFTs across all channels and all slices simultaneously.

All performance data shown were collected on our dual-socket × six-core Intel Xeon X5650 @2.67GHz system with four Nvidia GTX580s in PCI-Express slots. The system has 64GB of CPU DRAM, and 3GB of GPU DRAM per card (total 12 GB). We leverage Nvidia's Cuda [43] extensions to C/C++ to leverage massively parallel GPGPU processors, and OpenMP[2] to leverage multi-core parallelism on the system's CPUs. Additionally, multiple OpenMP threads are used to manage the interaction with the system's multiple discrete GPUs in parallel. We leverage freely available high-performance libraries for standard operations: ACML[3] for linear system solvers and matrix factorizations, FFTW[4] and CUFFT[5] for Fourier transforms.

6 Performance Results

Figures 5-10 present performance data for our parallelized ℓ_1-SPIRiT implementations.

Figure 5 shows stacked bar charts indicating the amount of wall-clock time spent during reconstruction of the six clinical datasets, whose sizes are listed in Section 5. The POCS solver is run with a single 2D slice in flight per GPU. This configuration minimizes memory footprint and is most portable across the widest variety of Cuda-capable GPUs. In the common case, it provides highest performance. Thus it is the default in our implementation. The stacked bars in Figure 5 represent:

3D Calibration: The SPIRiT calibration that computes the SPIRiT $\mathbf{G}$ operator from the ACS data as described in Section 3. Figure 9 presents more analysis of this portion.

POCS: The per-slice 2D data are reconstructed via the algorithm described in Figure 1. Figure 6 presents a more detailed analysis of the runtime of this portion.

other: Several other steps must also be performed during the reconstruction, including data permutation and IFFT of the readout dimension.

Figures 6 and 7 show the contribution of each individual algorithmic step step to the overall runtime of a single iteration of the 2D POCS solver. In Figure 6, the solver is parallelized so that a single 2D problem is in-flight per GPU. In Figure 7, a single 2D problem is in flight per CPU core. The stacked bars in Figure 6 and Figure 7 are:

FFT: The Fourier transforms performed during the k-space consistency projection.

SPIRiT Gx: Our image-domain implementation of the SPIRiT interpolation, which performs a matrix-vector multiplication per voxel.

Wavelet: The Wavelet transforms performed during wavelet soft-thresholding.

other: Other operations that contribute to runtime in-

[2]OpenMP: `http://www.openmp.org`

[3]ACML: `http://www.amd.com/acml`

[4]FFTW: `http://www.fftw.org`

[5]CUFFT: `http://developer.nvidia.com/cuda-toolkit-40`

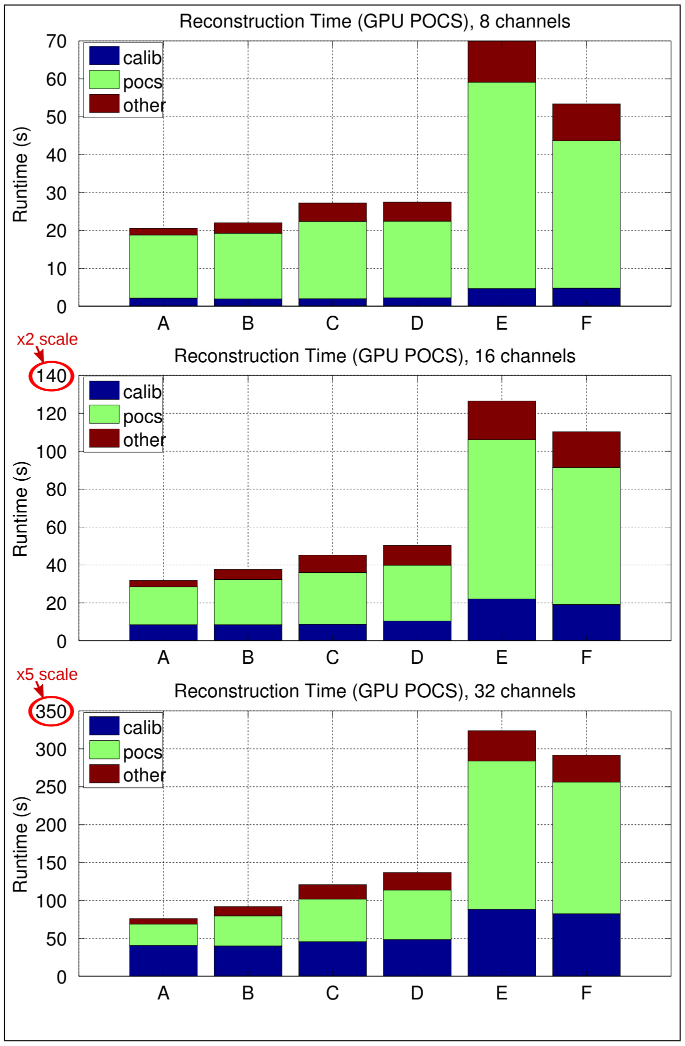

Figure 5: Reconstruction runtimes of our ℓ_1-SPIRiT solver for 8-, 16-, and 32-channel reconstructions using the efficient Cholesky-based calibration and the multi-GPU POCS solver.

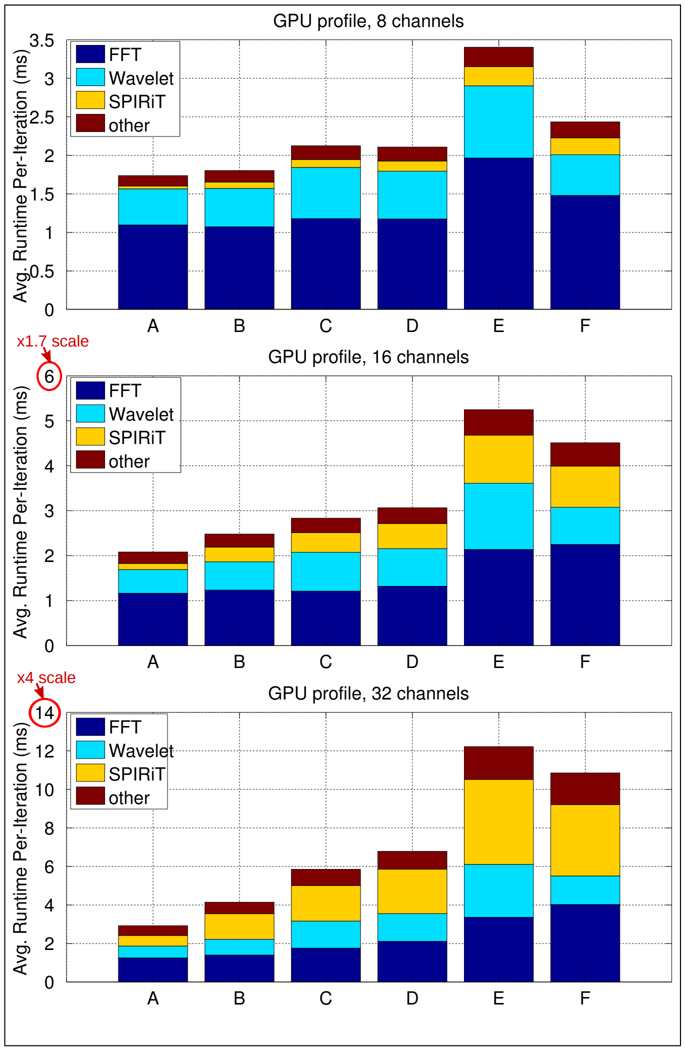

Figure 6: Per-iteration runtime and execution profile of the GPU 2-dimensional POCS solver. The solver is run on 2-dimensional slices, and the reconstruction is parallelized over multiple GPUs.

clude data movement, joint soft-thresholding, and the the k-space projection excluding Fourier transforms.

Figure 8 compares the runtime of the Parallel GPU and CPU POCS implementations to the runtime of a sequential C++ implementation, using high-performance libraries and compiled with full compiler optimization. The reported speedup is computed as the ratio of the sequential runtime to the parallel runtime.

Figure 9 demonstrates the runtime of the efficient Cholesky-based SPIRiT calibration algorithm described in Appendix A. The left graph compares the runtime of of our efficient $O(n^3)$ calibration to the naïve $O(n^4)$ algorithm. The right plot shows what fraction of the efficient algorithm's runtime is spent in the matrix-matrix multipliction, the Cholesky factorization, and the other BLAS2 matrix-vector operations.

7 Discussion

Figures 5 and 6 present performance details for the most portable GPU implementation of the POCS solver which runs a single 2D slice per GPU. As shown in Figure 5, our GPU-parallelized implementation reconstructs datasets A-D at 8 channels in less than 30 seconds, and requires about 1 minute for the larger E and F datasets. Similarly, our reconstruction runtime is 1-2 minutes for all but the 32-channel E and F data, which require about 5 minutes. Due to the $O(n_c^3)$ complexity of calibration, calibration requires a substantially higher fraction of runtime for the 32-channel reconstructions, compared to the 8 an 16-channel reconstructions. Similarly, Figure 6 shows that the $O(n_c^2)$ SPIRiT interpolation is a substantial fraction of the 32-channel POCS runtimes as well. Figures 5 and 6

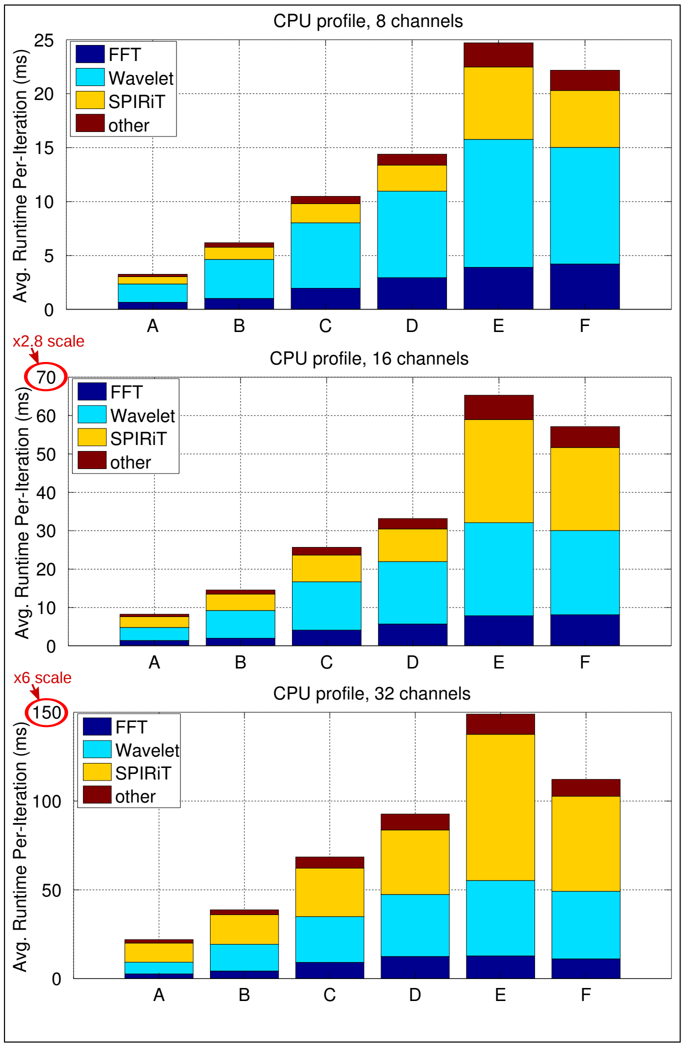

Figure 7: Per-iteration runtime and execution profile of the multi-core CPU 2-dimensional POCS solver.

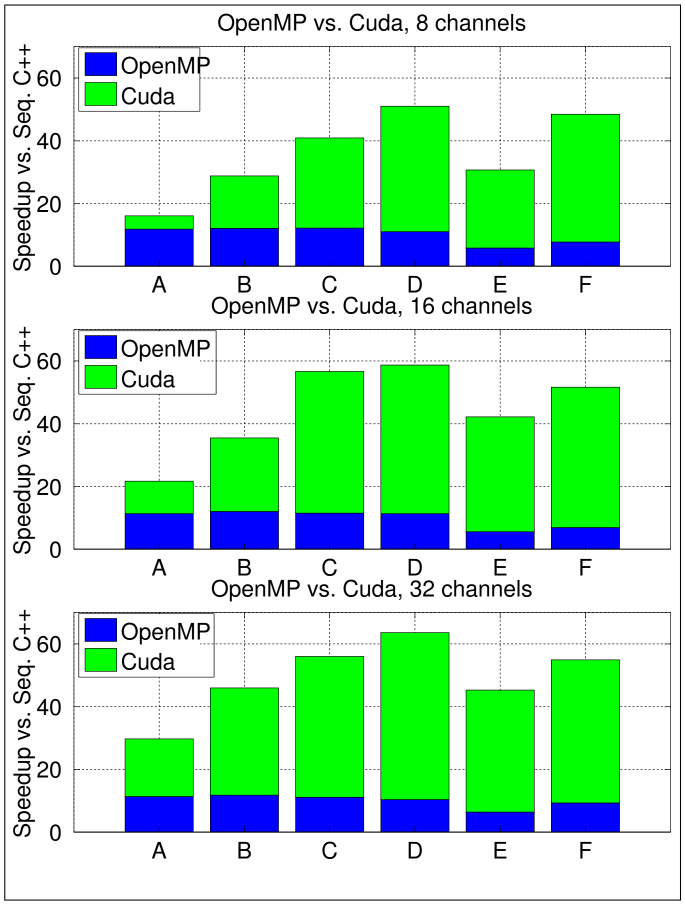

Figure 8: Speedup of parallel CPU and GPU implementations of the POCS solver over the optimized sequential C++ baseline.

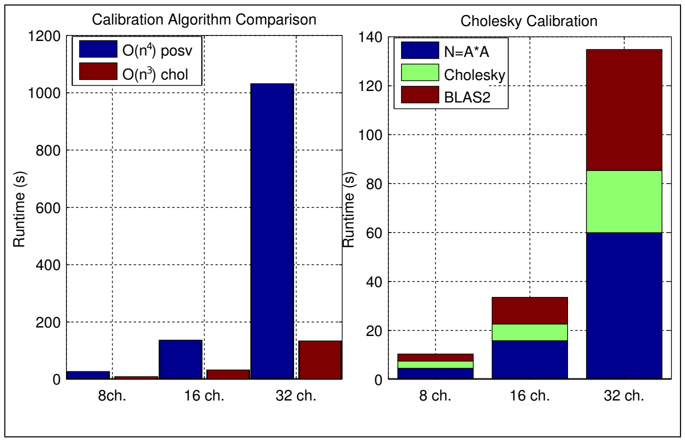

Figure 9: 3D SPIRiT Calibration runtimes, averaged over a wider variety of auto-calibration region sizes than that represented by Table 1. Calibration is always performed on the CPU via optimized library routines, using all available threads.

demonstrate another important trend of the performance of this GPU implementation. Although dataset D is 4× larger than dataset A, the 8-channel GPU POCS runtimes differ only by about 10%. The trend is clearest in the performance of the Fourier and Wavelet transforms, whose runtime is approximately the same for datasets A-D. This is indicative of the inefficiency of the CUFFT library's `Plan2D` API for these small matrix sizes. In a moment we'll discuss how an alternate parallelization strategy can substantially improve efficiency for these operations.

Figures 7 presents the averaged per-iteration execution profile of the OpenMP-parallelized CPU POCS solver, which uses an `#pragma omp for` to perform a single 2D slice's reconstruction at a time per thread. The relative runtimes of the Fourier and Wavelet transforms are more balanced in the CPU case. In particular, the CPU implementation does not suffer from low FFT performance for the small data sizes. The FFT is run sequentially within a single OpenMP thread, and it incurs no synchronization costs or parallelization overhead.

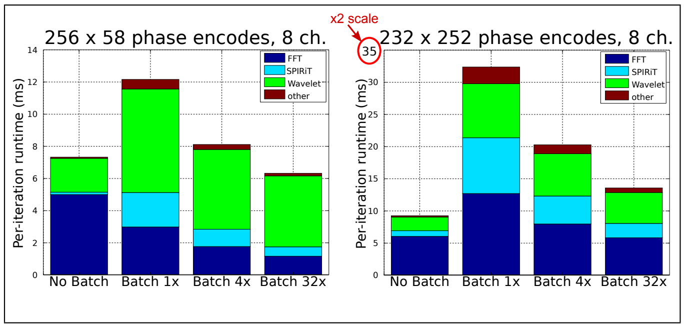

Figure 10: Data Size dependence of performance and comparison of alternate parallelizations of the POCS solver.

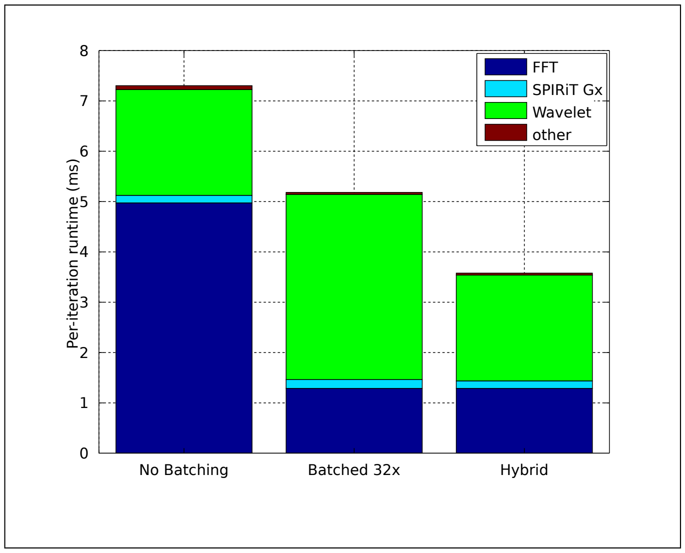

Figure 11: Performance achievable by a hybrid parallelization of the POCS solver on the 256×58 dataset.

Figure 8 presents the speedup of the multi-GPU solver and the multicore CPU solver over a sequential C++ implementation. Note that the 4-GPU implementation is only about 33% faster than the 12-CPU implementation for the smallest data size (dataset A at 8 channels), while for the larger reconstructions the GPU implementation is $5\times - 7\times$ faster. The OpenMP parallelization consistently gives $10\times - 12\times$ speedup over sequential C++, while the multi-GPU parallelization provides $30\times - 60\times$ speedup for most datasets.

Figure 9 demonstrates the enormous runtime improvement in SPIRiT calibration due to our Cholesky-based algorithm described in Section 3 and derived in Appendix A. The runtime of our calibration algorithm is dominated by a single large matrix-matrix multiplication, Cholesky decomposition, and various BLAS2 (Matrix-vector) operations. For 8 channel reconstructions, the $O(n^3)$ algorithm is faster by $2-3\times$, while it is $10\times$ faster for 32 channel data. In absolute terms, 8-channel calibrations require less than 10 seconds when computed via either algorithm. However, 32 channel calibrations run in 1-2 minutes via the Cholesky-based algorithm, while the $O(n^4)$ algorithm runs for over 15 minutes.

Figure 10 provides a comparison of alternate parallelizations of the POCS solver and the dependence of performance on data size. The "No Batching" implementation exploits the voxel-wise and channel-wise parallelism within a single 2D problem per GPU Socket. The remaining bars batch multiple 2D slices per GPU Socket. The top bar graph shows runtimes for a small 256×58 image matrix, and the bottom graph shows runtimes for a moderately sized 232×252 matrix. Both reconstructions were performed after coil-compression to 8 channels.

Fourier transforms in the "No Batching" implementation are particularly inefficient for the small data size. The 256×58 transforms for the 8 channels are unable to saturate the GPU. The "Batched 1x" bar uses the `PlanMany` API rather than the `Plan2D` API. This change improves FFT performance, demonstrating the relative ineffectiveness of the GPU's ability to execute multiple grid launches simultaneously. Performance continues to improve as we increase the number of slices simultaneously in-flight, and the FFTs of the small matrix are approximately $5\times$ faster when batched $32\times$. However, for the larger 232×252 dataset, $32\times$ batching achieves performance approximately equal to the non-batched implementation. That the $1\times$ batched performance is worse than the non-batched performance likely indicates that the larger FFT is able to exploit multiple GPU cores.

Our Wavelet transforms are always more efficient without batching, as the implementation is able to exploit the GPU's small scratchpad caches (Cuda `__shared__` memory) as described in Section 4.4. The Wavelet transform performs convolution of the low-pass and high-pass filters with both the rows and the columns of the image. Our images are stored in column-major ordering, and thus we expect good caching behavior for the column-wise convolutions. However, the row-wise convolutions access the images in non-unit-stride without our scratchpad-based optimizations. Comparing the runtimes of the "No Batching" and "Batched 1x" Wavelet implementations in Figure 10 shows that our cache optimization can improve performance by $3\times - 4\times$. This is a sensible result, as we use 4-tap filters and each pixel is accessed 4 times per convolution. The cache optimization reduces the cost to a single DRAM access and 3 cached accesses.

Performance can be improved by choosing different parallelization strategies for the various operations. In particular, the best performance would be achieved by using a batched implementation of the Fourier transforms, while using the un-batched implementation of the Wavelet transforms. Such an implementation would still require the larger DRAM footprint of the batched implementation, as multiple 2D slices must be resident in GPU DRAM simultaneously. However it could achieve high efficiency in the Wavelet transform via the caching optimization, and also in the Fourier transforms via higher processor utilization. Although our current implementation does not support this hybrid configuration, Figure 11 shows that it could perform up to 2× faster for the 256 × 58 dataset. Moore's Law scaling will result in higher core counts in future architectures. Per Gustafson's law [44], efficient utilization of future architectures will require increased problem size. In our context, we can increase problem size via larger batch sizes. Per-batch reconstruction time will remain constant, but total reconstruction time will be inversely proportional to batch size. Thus batching potentially provides linear performance scaling with increased core counts.

8 Image Quality

We present more comprehensive evaluation of image quality in prior works [20], and present in Figure 12 a case demonstrating the clinical advantage that high-performance reconstruction can provide. Our 3-Dimensional Compressed Sensing pulse sequence is a modified 3DFT spoiled gradient-echo (SPGR) sequence which undersamples in both of the phase-encoded dimensions (y) and (z). Acquisitions are highly accelerated, with 4×-8× undersampling of phase encodes. Our clinical imaging is performed using 3T and 1.5T GE systems with a with 32-channel pediatric torso coil. Typical accelerated scan times are 10-15 seconds, and typical ARC [45] reconstruction times are 30-60 seconds. In some cases, we perform partial k-space acquisition in the readout direction. The ℓ_1-SPIRiT solver is still able to decouple the 2D reconstructions as described in Section 4.2, using only the acquired portion of the readout. Subsequently, we perform Homodyne reconstruction [46] to estimate the missing portion of readout, preventing blur and phase from appearing in the final images [6]. Our reconstruction is performed on-line with coil compression, producing sub-minute runtimes for matrix sizes typically acquired in the clinic. Total latency from scan completion to image availability is 2-3 minutes, 20-70 seconds of which are the POCS solver. The remainder of the reconstruction time is spent performing Grad-Warp [47] and Homodyne processing steps, in addition to file transfers between the scanner and our reconstruction system.

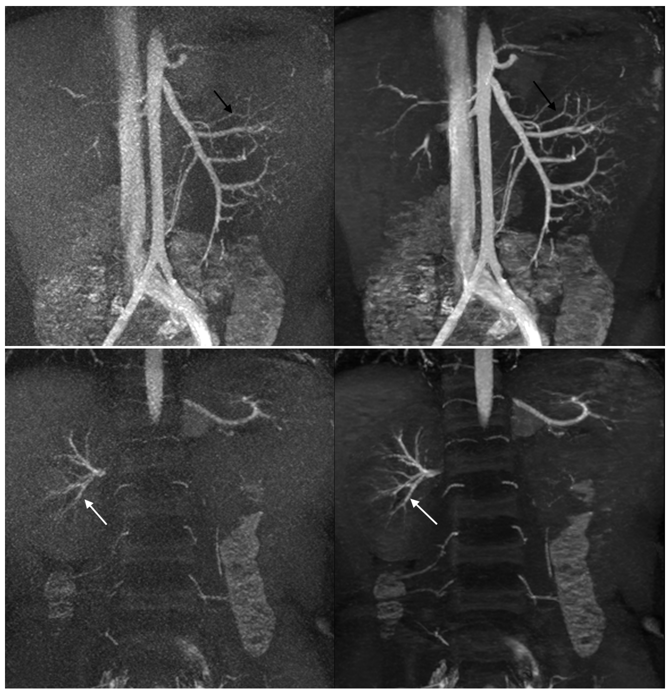

Figure 12: Comparison of GE Product ARC (Autocalibrating Reconstruction for Cartesian imaging) [45] reconstruction (left images) with our ℓ_1-SPIRiT reconstruction (right images). Both reconstructions use the same subsampled data, and require similar runtimes. These MRA images of a 5 year old patient were acquired with the 32-channel pediatric torso coil, have FOV 28 × 22.4 × 17.6 cm and spatial resolution 0.55 × 0.55 × 0.8 mm. The images were sampled and reconstructed in a $192 \times 256 \times 220$ image matrix, were acquired with 7.2× acceleration, via undersampling 3.6× in the y-direction and 2× in the z-direction. The pulse sequence used a 15 degree flip angle and a TR of 3.9 ms. The ℓ_1-SPIRiT reconstruction shows enhanced detail in the mesenteric vessels in the top images, and and the renal vessels in bottom images.

9 Conclusion

We have presented ℓ_1-SPIRiT, a compressive sensing extension to the SPIRiT parallel imaging reconstruction. Our implementation of ℓ_1-SPIRiT for GPGPUs and multi-core CPUs achieves clinically feasible sub-minute runtimes for highly accelerated, high-resolution scans. We discussed in general terms the software implementation and optimization decisions that contribute to our fast runtimes, and how they apply for the individual operations in ℓ_1-SPIRiT. We presented performance data for both CPU and GPU systems, and discussed how a hybrid parallelization may achieve faster runtimes. Finally, we present an image quality comparison with a competing non-iterative Parallel Imaging reconstruction approach.

In the spirit of reproducible research, the software described in this paper is available at: `http://www.eecs.berkeley.edu/~mlustig/Software.html`

10 Acknowledgments

This work was supported by NIH grants R01 EB009690, P41 RR09784, UC Discovery Grant #193037, American Heart Association Award #12BGIA9660006, and GE Healthcare.

References

[1] D K Sodickson and W J Manning. Simultaneous acquisition of spatial harmonics (smash): fast imaging with radiofrequency coil arrays. *Magn Reson Med*, 38(4):591–603, Oct 1997.

[2] K P Pruessmann, M Weiger, M B Scheidegger, and P Boesiger. Sense: sensitivity encoding for fast mri. *Magn Reson Med*, 42(5):952–62, Nov 1999.

[3] Mark A Griswold, Peter M Jakob, Robin M Heidemann, Mathias Nittka, Vladimir Jellus, Jianmin Wang, Berthold Kiefer, and Axel Haase. Generalized autocalibrating partially parallel acquisitions (GRAPPA). *Magn Reson Med*, 47(6):1202–10, 2002.

[4] E.J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52:489–509, 2006.

[5] D.L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52:1289–1306, 2006.

[6] Michael Lustig, David L. Donoho, and John Mark Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magn Reson Med*, 58(6):1182–1195, 2007.

[7] K P Pruessmann, M Weiger, P Börnert, and P Boesiger. Advances in sensitivity encoding with arbitrary k-space trajectories. *Magn Reson Med*, 46(4):638–51, Oct 2001.

[8] Kai Tobias Block, Martin Uecker, and Jens Frahm. Undersampled radial mri with multiple coils. iterative image reconstruction using a total variation constraint. *Magn Reson Med*, 57(6):1086–98, Jun 2007.

[9] Dong Liang, Bo Liu, Jiunjie Wang, and Leslie Ying. Accelerating sense using compressed sensing. *Magn Reson Med*, 62(6):1574–84, Dec 2009.

[10] Ricardo Otazo, Daniel Kim, Leon Axel, and Daniel K Sodickson. Combination of compressed sensing and parallel imaging for highly accelerated first-pass cardiac perfusion mri. *Magn Reson Med*, 64(3):767–76, Sep 2010.

[11] Joshua D Trzasko, Clifton R Haider, Eric A Borisch, Norbert G Campeau, James F Glockner, Stephen J Riederer, and Armando Manduca. Sparse-capr: Highly accelerated 4d ce-mra with parallel imaging and nonconvex compressive sensing. *Magn Reson Med*, 66(4):1019–32, Oct 2011.

[12] Bing Wu, Rick P Millane, Richard Watts, and Philip J Bones. Prior estimate-based compressed sensing in parallel mri. *Magn Reson Med*, 65(1):83–95, Jan 2011.

[13] Leslie Ying and Jinhua Sheng. Joint image reconstruction and sensitivity estimation in sense (jsense). *Magn Reson Med*, 57(6):1196–202, Jun 2007.

[14] Martin Uecker, Thorsten Hohage, Kai Tobias Block, and Jens Frahm. Image reconstruction by regularized nonlinear inversion–joint estimation of coil sensitivities and image content. *Magn Reson Med*, 60(3):674–82, Sep 2008.

[15] Florian Knoll, Christian Clason, Kristian Bredies, Martin Uecker, and Rudolf Stollberger. Parallel imaging with nonlinear reconstruction using variational penalties. *Magn Reson Med*, Jun 2011.

[16] Feng Huang, Yunmei Chen, Wotao Yin, Wei Lin, Xiaojing Ye, Weihong Guo, and Arne Reykowski. A rapid and robust numerical algorithm for sensitivity encoding with sparsity constraints: self-feeding sparse sense. *Magn Reson Med*, 64(4):1078–88, Oct 2010.

[17] Michael Lustig and John M. Pauly. SPIRiT: Iterative self-consistent parallel imaging reconstruction from arbitrary k-space. *Magnetic Resonance in Medicine*, 64(2):457–471, 2010.

[18] M. Lustig, M. Alley, S. Vasanawala, D.L. Donoho, and J.M. Pauly. ℓ_1-SPIRiT: Autocalibrating parallel imaging compressed sensing. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, page 379, 2009.

[19] S.S. Vasanawala, M.J. Murphy, M.T. Alley, P. Lai, K. Keutzer, J.M. Pauly, and M. Lustig. Practical parallel imaging compressed sensing MRI: Summary of two years of experience in accelerating body MRI of pediatric patients. In *Proceedings of IEEE International Symposium on Biomedical Imaging*, pages 1039 –1043, Chicago, 2011.

[20] Shreyas S Vasanawala, Marcus T Alley, Brian A Hargreaves, Richard A Barth, John M Pauly, and Michael Lustig. Improved pediatric MR imaging with compressed sensing. *Radiology*, 256(2):607–16, Aug 2010.

[21] Asanovic et al. The landscape of parallel computing research: a view from berkeley. Technical report, EECS Department, University o California, berkeley, Dec 2006.

[22] Ching-Hua Chang and Jim Ji. Compressed sensing mri with multichannel data using multicore processors. *Magn Reson Med*, 64(4):1135–9, Oct 2010.

[23] Daehyun Kim, Joshua Trzasko, Mikhail Smelyanskiy, Clifton Haider, Pradeep Dubey, and Armando Manduca. High-performance 3d compressive sensing mri reconstruction using many-core architectures. *Journal of Biomedical Imaging*, 2011:2:1–2:11, January 2011.

[24] S.S. Stone, J.P. Haldar, S.C. Tsao, W. m.W. Hwu, B.P. Sutton, and Z.-P. Liang. Accelerating advanced mri reconstructions on gpus. *Journal of Parallel and Distributed Computing*, 68(10):1307 – 1318, 2008. ¡ce:title¿General-Purpose Processing using Graphics Processing Units¡/ce:title¿.

[25] Xiao-Long Wu, Jiading Gai, Fan Lam, Maojing Fu, Justin Haldar, Yue Zhuo, Zhi-Pei Liang, Wen mei Hwu, and Bradley Sutton. Impatient mri: Illinois massively parallel acceleration toolkit for image reconstruction with enhanced throughput in mri. In *Proceedings of the IEEE International Symposium on Biomedical Imaging (ISBI)*, 2011.

[26] Yue Zhuo, Xiao-Long Wu, Justin P. Haldar, Wen-Mei W. Hwu, Zhi-Pei Liang, and Bradley P. Sutton. Multi-gpu implementation for iterative mr image reconstruction with field correction. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, 2010.

[27] P.J. Beatty, D.G. Nishimura, and J.M. Pauly. Rapid gridding reconstruction with a minimal oversampling ratio. *IEEE Trans Med Imaging*, 24:799–808, Jun 2005.

[28] T S Sorensen, T Schaeffter, K O Noe, and M S Hansen. Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. *IEEE Transactions on Medical Imaging*, 27(4):538–547, 2008.

[29] Nady Obeid, Ian Atkinson, Keith Thulborn, and Wen-Mei Hwu. Gpu-accelerated gridding for rapid reconstruction of non-cartesian mri. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, 2011.

[30] Mehmet Akcakaya, Tamer Basha, Warren Manning, Seunghoon Nam, Reza Nezafat, Christian Stehning, and Vahid Tarokh. A gpu implementation of compressed sensing reconstruction of 3d radial (kooshball) acquisition for high-resolution cardiac mri. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, 2011.

[31] S. Roujol, B. D. de Senneville, E. Vahala, T. S. Sorensen, C. Moonen, and M Ries. Online realtime reconstruction of adaptive tsense with commodity cpu/gpu hardware. *Magnetic Resonance in Medicine, 62:1658-1664*, 2009.

[32] Thomas Sangild Sørensen, David Atkinson, Tobias Schaeffter, and Michael Sass Hansen. Real-time reconstruction of sensitivity encoded radial magnetic resonance imaging using a graphics processing unit. *IEEE Trans. Med. Imaging*, 28(12):1974–1985, 2009.

[33] Tiejun Zhao and Xiaoping Hu. Iterative grappa (igrappa) for improved parallel imaging reconstruction. *Magnetic Resonance in Medicine*, 59(4):903–907, 2008.

[34] M. Murphy, K. Keutzer, S. Vasanawala, and M. Lustig. Clinically feasible reconstruction time for ℓ_1-SPIRiT parallel imaging and compressed sensing MRI. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, page 4854, 2010.

[35] Peng Lai, Michael Lustig, Anja CS. Brau, Shreyas Vasanawa la, Philip J. Beatty, and Marcus Alley. Efficient ℓ_1-SPIRiT reconstruction (ESPIRiT) for highly accelerated 3D volumetric MRI with parallel imaging and compressed sensing. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, page 345, 2010.

[36] RW Buccigrossi and EP Simoncelli. Image compression via joint statistical characterization in the wavelet domain. *IEEE Trans. Image Processing*, 8:1688–1701, 1999.

[37] D.L. Donoho and I.M. Johnstone. Ideal spatial adaptation via wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[38] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Comm. Pure Applied Mathematics*, 57:1413 – 1457, 2004.

[39] M.A.T. Figueiredo and R.D. Nowak. An em algorithm for wavelet-based image restoration. *Image Processing, IEEE Transactions on*, 12(8):906 – 916, August 2003.

[40] Martin Buehrer, Klaas P Pruessmann, Peter Boesiger, and Sebastian Kozerke. Array compression for MRI with large coil arrays. *Magn Reson Med*, 57(6):1131–9, 2007.

[41] Tao Zhang, Michael Lustig, Shreyas Vasanawala, and John Pauly. Array compression for 3d cartesian sampling. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, page 2857, 2011.

[42] Timothy Mattson, Beverly Sanders, and Berna Massingill. *Patterns for parallel programming*. Addison-Wesley Professional, first edition, 2004.

[43] Nvidia. Compute Unified Device Architecture (Cuda). `http://www.nvidia.com/object/cuda_get.html`. [Online; accessed 25 July, 2011].

[44] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31:532–533, May 1988.

[45] P.J. Beatty, A Brau, S Chang, S Joshi, Michelich C, Bayram E, T Nelson, R Herfkens, and J Brittain. A method for autocalibrating 2d-accelerated volumetric parallel imaging with clinically practical reconstruction times. In *Proceedincs of the Joint Annual Meeting ISMRM-ESMRMB*, page 1749, 2007.

[46] D. C. Noll, D. G. Nishimura, and A. Macovski. Homodyne detection in magnetic resonance imaging. *Medical Imaging, IEEE Transactions on*, 10(2):154–163, 1991.

[47] Gary H. Glover and Norbert J. Pelc. Method for correcting image distortion due to gradient nonuniformity, 05 1986.

A $O(n^3)$ SPIRiT Calibration

As discussed in Section 3, our ℓ_1-SPIRiT calibration solves a least-norm, least-squares (LNLS) fit to the fully-sampled auto-calibration signal (ACS) for each channel's interpolating coefficients. As each channel's set of coefficients interpolates from each of the the n_c channels, the matrix used to solve this system has $O(n_c)$ columns. Solving the n_c least-squares systems independently requires $O(n_c^4)$ time, and is prohibitively expensive. This section derives our algorithm for solving them in $O(n_c^3)$ time using a single matrix-matrix multiplication, single Cholesky factorization, and several inexpensive matrix-vector operations.

We construct a calibration data matrix A from the calibration data in the same manner as GRAPPA [3] and SPIRiT [17] calibration: each row of A is a window of the ACS the same size as the interpolation coefficients. This matrix is Toeplitz, and multiplication Ax by a vector $x \in \mathbb{C}^{n_c \cdot n_k}$ computes the SPIRiT interpolation: $y = \sum_{i=1}^{n_c} x_i * \text{ACS}_i$.

The LNLS matrices for each channel differ only by a single column from A. In particular, there is a column of A that is identical to the ACS of each coil. Consider the coil corresponding to column i of A, and let b be that column. We define $N = A - be_i{}' - A$ with the i^{th} column zeroed out. We wish to solve $Nx = b$ in the least-norm least-squares sense, by solving $(N^*N+\varepsilon I)x := \tilde{N}x = N^*b$. The runtime of our efficient algorithm is dominated computing the product N^*N and computing the Cholesky factorization $LL^* = \tilde{N}$.

Our derivation begins by noting that:

$$\begin{aligned}\tilde{N} &= N^*N + \varepsilon I \\ &= (A - be')^*(A - be') + \varepsilon I \\ &= A^*A + \varepsilon I - \tilde{b}e' - e\tilde{b}^*\end{aligned}$$

Where we have defined $\tilde{b} = A^*b$ with entry i multiplied by $\frac{1}{2}$ to avoid adding eb^*be'. If we have a Cholesky factorization $LL^* = A^*A + \varepsilon I$:

$$\begin{aligned}\tilde{N} &= LL^* - \tilde{b}e' - e\tilde{b}^* \\ &= LL^{-1}(LL^* - \tilde{b}e' - e\tilde{b}^*)L^{-*}L^* \\ &= L(I - \hat{b}\hat{e}^* - \hat{e}\hat{b}^*)L^*\end{aligned}$$

Where we've defined $\hat{b} = L^{-1}\tilde{b} = L^{-1}A^*b$, and $\hat{e} = L^{-1}e$. These vectors can be computed with BLAS2 triangular solves and matrix-vector multiplications. In fact, we can aggregate the b's and e's from all parallel imaging channels into matrices and compute all $\hat{b}$'s and $\hat{e}$'s with highly efficient BLAS3 solves. Now, to solve the system of equations $\tilde{N}x = \tilde{b}$:

$$\begin{aligned}x &= \tilde{N}^{-1}\tilde{b} \\ &= (L(I - \hat{b}\hat{e}^* - \hat{e}\hat{b}^*)L^*)^{-1}\tilde{b} \\ &= L^{-*}(I - \hat{b}\hat{e}^* - \hat{e}\hat{b}^*)^{-1}L^{-1}\tilde{b}\end{aligned}$$

It remains to compute the inverse of $(I - \hat{b}\hat{e}^* - \hat{e}\hat{b}^*)$. We can define two matrices $\hat{B}, \hat{E} \in \mathbb{C}^{n\times 2}$, where $\hat{B} = -\left(\hat{b}, \hat{e}\right)$, and $\hat{E} = \left(\hat{e}, \hat{b}\right)$. Using the Sherman-Morrison-Woodbury identity:

$$\begin{aligned}(I - \hat{b}\hat{e}^* - \hat{e}\hat{b}^*)^{-1} &= (I + \hat{B}\hat{E}^*)^{-1} \\ &= I - \hat{B}(I + \hat{E}^*\hat{B})^{-1}\hat{E}^*\end{aligned}$$

Note that $I + \hat{E}^*\hat{B}$ is a 2×2 matrix that is very inexpensive to invert. Thus we have our final algorithm:

$$\begin{aligned}
L &\leftarrow \mathrm{chol}(A^*A + \varepsilon I) \\
\hat{b} &\leftarrow L^{-1}A^*b \\
\hat{e} &\leftarrow L^{-1}e \\
\hat{B} &\leftarrow -\left(\hat{b}, \hat{e}\right) \\
\hat{E} &\leftarrow \left(\hat{e}, \hat{b}\right) \\
x &\leftarrow L^{-*}(I - \hat{B}(I + \hat{E}^*\hat{B})^{-1}\hat{E}^*)\hat{b}
\end{aligned}$$