32nd Annual International Conference of the IEEE EMBS
Buenos Aires, Argentina, August 31 - September 4, 2010

High-Performance 3D Compressive Sensing MRI Reconstruction

Daehyun Kim, Joshua D. Trzasko, Mikhail Smelyanskiy,
Clifton R. Haider, Armando Manduca, and Pradeep Dubey

Abstract— **Compressive Sensing (CS) is a nascent sampling and reconstruction paradigm that describes how sparse or compressible signals can be accurately approximated using many fewer samples than traditionally believed. In magnetic resonance imaging (MRI), where scan duration is directly proportional to the number of acquired samples, CS has the potential to dramatically decrease scan time. However, the computationally expensive nature of CS reconstructions has so far precluded their use in routine clinical practice – instead, more-easily generated but lower-quality images continue to be used.**

We investigate the development and optimization of a proven inexact quasi-Newton CS reconstruction algorithm on several modern parallel architectures, including CPUs, GPUs, and Intel's Many Integrated Core (MIC) architecture. Our (optimized) baseline implementation on a quad-core Core i7 is able to reconstruct a 256x160x80 volume of the neurovasculature from an 8-channel, 10x undersampled data set within 56 seconds, which is already a significant improvement over existing implementations. The latest six-core Core i7 reduces the reconstruction time further to 32 seconds. Moreover, we show that the CS algorithm benefits from modern throughput-oriented architectures. Specifically, our CUDA-base implementation on NVIDIA GTX480 reconstructs the same dataset in 16 seconds, while Intel's Knights Ferry (KNF) of the MIC architecture even reduces the time to 12 seconds. Such level of performance allows the neurovascular dataset to be reconstructed within a clinically viable time.

Keywords: Compressive Sensing, MRI, Angiography, Reconstruction, GPU, Many Core

I. INTRODUCTION AND MOTIVATION

Magnetic resonance imaging (MRI) is a powerful medical imaging modality commonly used to investigate soft tissues in the human body. However, as MRI scan duration is proportional to the number acquired data samples, obtaining high-resolution images can require a significant amount of time. Prolonged scan duration poses a number of challenges in a clinical setting. For example, during long examinations patients often exhibit involuntary (e.g. respiration) and/or voluntary motion (e.g. active response to discomfort), both of which can impart spatial blurring. High temporal resolution is also needed to accurately depict physiological processes. Under standard imaging protocols, spatial resolution must unfortunately be sacrificed to permit quicker scan termination or more frequent temporal updates.

Rather than executing a low spatial resolution exam, many MRI protocols acquire a subset of the samples associated with a high-resolution exam and attempt to recover the image using constrained reconstruction methods such as Compressive Sensing (CS). CS theory asserts that the number of samples needed to form an accurate approximation of an image is largely determined by the image's underlying complexity [1], [2]. Thus, if there exists a means of transforming the image into a more efficient (i.e. sparse or compressible) representation, significantly less time may actually be required to collect the data set needed to form the high-resolution image [3].

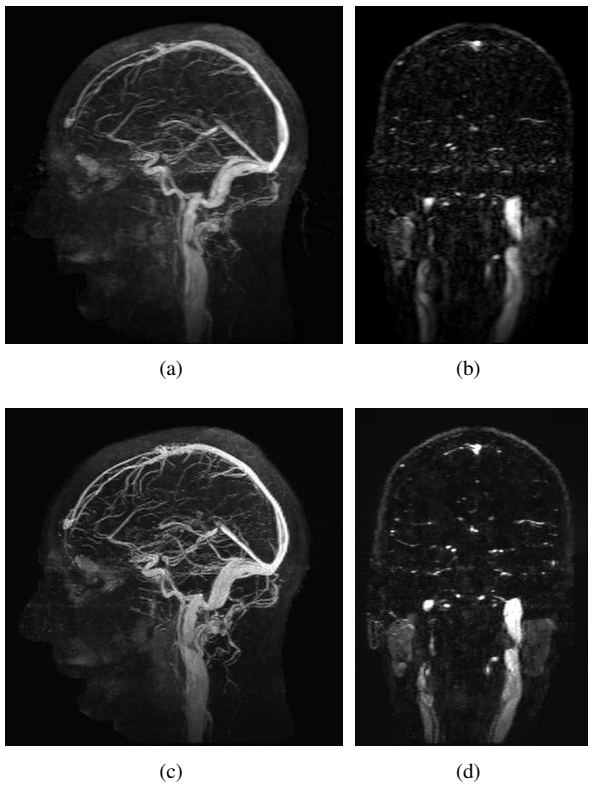


Fig. 1. Sagittal maximum intensity projection (MIP) images (column 1) and coronal cross section images (column 2) for test data set 5. (a-b) represent the current clinical reconstruction protocol result, and (c-d) represent the CS reconstruction obtained in 32s using an Intel six-core CPU. Note the superior vascular conspicuity and parotid gland homogeneity in the CS reconstruction images.

Contrast-enhanced MR angiography (CE-MRA) is a very natural clinical target for CS methods. As the diagnosis of many conditions like peripheral vascular disease are based on both vessel morphology and hemodynamics, images with both high spatial and temporal resolution are needed. CS enables the acquisition of such data in a single exam. Although several authors (e.g. [[4], [5], [6]]) have successfully demonstrated the application of CS methods to CE-MRA, the computationally-intensive nature of these applications has so far precluded their clinical viability. For example, published

DK, MS and PD are with Throughput Computing Lab, Intel Corporation
JT, CH and AM are with the Center for Advanced Imaging Research, Mayo Clinic

CS reconstruction times for a single 3D volume (CE-MRA or not) range from minutes to hours [3], [7], [8], [9], [6], even when advanced hardware environments were employed [10], [11] or the 3D problem was approximated by a series of 2D problems [3]. As the results of a CE-MRA exam are often needed as soon as the acquisition completes (either for immediate clinical intervention or to guide additional scans), it is not practical to wait for the result of any currently-implemented CS reconstructions. Instead, linear or other non-iterative reconstructions that can be executed online (i.e. subsecond run-times for 3D volume reconstruction) must be used, even if they provide suboptimal results.

Recently, Trzasko et al. [11], [6] demonstrated high quality non-convex CS reconstructions of 3D CE-MRA images acquired using the state-of-the-art CAPR acquisition strategy [12] in a matter of only minutes per 3D volume using an advanced code implementation on a cluster system. In this paper we investigate the development, optimization and performance analysis of a proven inexact quasi-Newton CS reconstruction algorithm on several modern parallel architectures, including the latest quad and six-core CPUs, NVIDIA GPUs and Intel's MIC architecture. Our optimized CS implementation on a six-core CPU is able to reconstruct a 256x160x80 volume of the neurovasculature from an 8-channel, 10x under-sampled data set within 32 seconds, which is more than a 3.4x improvement over other conventional implementations. Furthermore, we show that our CS implementation scales very well to the larger number of cores. Our GTX480 implementation reconstructs the same dataset in 16 seconds using the NVIDIA CUFFT library. And, Intel's KNF (an implementation of Intel's MIC architecture with 32 cores at 1.2GHz) can reconstruct the same dataset within a clinically viable 12 seconds.

II. METHODS

A. Acquisition and Recovery of CAPR CE-MRA Images

As described in [12], CAPR adopts a SENSE-type [13] parallel imaging strategy. Letting f be a discrete approximation of the underlying image of interest, the targeted data acquisition process can thus be modeled as

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_C \end{bmatrix} = \begin{bmatrix} \Phi\mathcal{F}\Gamma_1 \\ \Phi\mathcal{F}\Gamma_2 \\ \vdots \\ \Phi\mathcal{F}\Gamma_C \end{bmatrix} f + n, \qquad (1)$$

where g_c is the c^{th} coil sensor signal, Γ_c is the c^{th} coil sensitivity function, $\mathcal{F}$ is the 3D DFT operator, Φ is a (binary) sampling operator that selects a prescribed subset of k-space values, and n is complex AWGN. Raw CAPR data is traditionally background subtracted and view-shared prior to reconstruction. Accordingly, let $h_c(t)$ denote the result after such preprocessing of g_c.

It was demonstrated in [6] and [11] that background-suppressed CE-MRA images acquired by systems of the form in (1) can be accurately recovered by solving $\tilde{v} = \arg\min_v J(v)$, where the cost functional $J(v) = \alpha \sum_{n\in\eta} P(D_n v) + \sum_{c=1}^{C} \|\Phi\mathcal{F}\Gamma_c v - h_c\|_2^2$, D_n is the finite spatial difference operator for some offset n, and the penalty or prior functional $P(v) = \sum_{x\in\Omega} \rho\left(v(x)\right)$ for some concave metric functional $\rho(\cdot)$. Following [6], the non-convex Laplace functional $\rho(\cdot) = 1 - \exp(\sigma^{-1}|\cdot|)$, for some $\sigma \in [0, \infty)$, is herein adopted.

B. Numerical Optimization

In [6], an efficient inexact quasi-Newton algorithm was proposed for solving the described constrained minimization to reconstruct CAPR CE-MRA images. Recall that complex quasi-Newton iterations [14] are typically of the form $v_{i+1} = v_i - B^{-1}(v_i)L(v_i)$, where $L(\cdot)$ is the gradient of $J(v)$ (taken with respect to $\overline{v}$ [15]) and $B(\cdot)$ is an approximation of the complex Hessian of $J(v)$. The term "inexact" arises when Δ_i is only approximately determined, such as by truncated conjugate gradient (CG) iteration. Given $J(v)$, note that $L(v_i) = \alpha \sum_{n\in\eta} D_n^* \Lambda(D_n v_i) D_n v_i + \sum_{c=1}^{C} \Gamma_c^* \mathcal{F}^* \Phi^* (\Phi\mathcal{F}\Gamma_c v_i - k_c(t))$, where the (relaxed) diagonal operator $\Lambda(D_n v_i)_{(x,x)} = \frac{1}{2|[D_n v_i](x)|_{\epsilon_i}} \cdot \frac{\partial\rho\left(|[D_n v_i](x)|_{\epsilon_i}\right)}{\partial|[D_n v_i](x)|_{\epsilon_i}}$, for some arbitrarily small $\epsilon > 0$. In their work , Trzasko et al. [6] adopted the following analytical linear Hessian approximation: $B(v_i) = \frac{\alpha}{2} \sum_{n\in\eta} D_n^* \Lambda(D_n v_i) D_n + \sum_{c=1}^{C} \Gamma_c^* \mathcal{F}^* \Phi^* \Phi\mathcal{F}\Gamma_c$, which is a generalization of Hessian model used by Vogel and Oman [16] for total variation (TV) denoising. For improved convergence, decreasing continuation is also performed on ϵ [17].

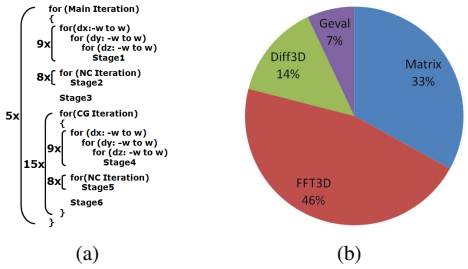
In [11], an efficient C++ implementation of the above algorithm that employed the templated class framework described by Borisch et al. [18] and both the MPI and OpenMP libraries was described and executed on an 8-node dedicated reconstruction cluster, where each node had two 3.4GHz Xeon processors and 16GB memory. For a single 256x160x80 head volume reconstruction from 8-channel data and only 6 difference neighbors, reconstruction times of slightly less than 2 minutes were reported. Although these times represent a significant advancement over other existing work, they are still too long for routine clinical use.

III. EXPERIMENTS

We used five datasets (two artificial and three clinical) to analyze the CS performance on three platforms: Intel CPUs, NVIDIA GPUs, and Intel's MIC architecture.

A. Computing Architectures

Intel Core i7 The Intel Core i7 is an x86-based multi-core architecture which provides four/six cores (731M/1.17B transistors) on the same die. It features a super-scalar out-of-order core supporting 2-way hyper-threading and 4-wide SIMD. Each core is backed by a 32KB L1 and a 256KB L2 caches, and all cores share an 8MB/12MB L3 cache. Quad and six-core CPUs provide 100 Gflops and 135 Gflops of peak single-precision computation respectively, as well 32 GB/s of peak memory bandwidth. To optimize CS on Core i7, we took advantage of its SSE4 instructions using the Intel ICC auto-vectorizing compiler as well as



(a) (b)

Fig. 2. (a) CS Implementation Overview (b) Execution Time Breakdown

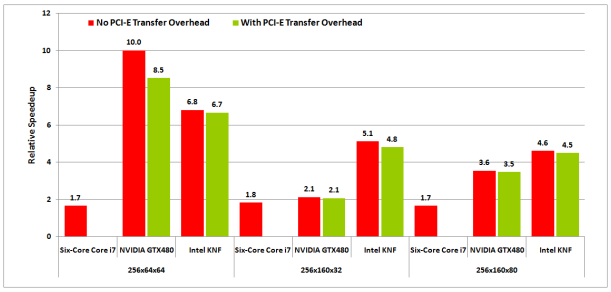


Fig. 3. Performance Comparison between Core i7, GTX480 and KNF

hand-vectorization intrinsics. We parallelized the code with OpenMP and adopted a highly optimized FFT implementation from Intel's Math Kernel Library (*MKL*) 10.2.

NVIDIA GTX480 The NVIDIA GTX480 (Fermi [19], 3B transistors) provides 16 multiprocessors, each with 32 scalar processing units that share 128KB of registers and a 64KB on-chip memory. 32 scalar units are broken into two groups, where each group runs in lockstep. All multiprocessors share a 768KB L2 cache. Its peak single-precision computing performance is about 1.35 Tflops and its on-board GDDR memory provides up to 121 GB/s bandwidth. We used the CUDA [20] programming environment to implement CS on the GTX480. CUDA allows programmers to write a scalar program that is automatically organized into thread blocks to be run on multiprocessors. CUDA provides an open source FFT library (CUFFT 2.1 [21]), although more optimized FFT implementations such as Nukada's [22] have been published.

Intel Knights Ferry The Intel's MIC architecture is a Aubrey Isle [23] based silicon platform and Knights Ferry (KNF) [24] is its first implementation with 32 cores running at 1.2GHz. It is an x86-based many-core processor based on small in-order cores that combines the full programmability of today's general-purpose CPU architectures with the compute-throughput and memory bandwidth capabilities of modern GPU architectures. Each core is a general-purpose processor, which has a scalar unit based on the Pentium processor design, as well as a vector unit that supports 16 32-bit float or integer operations per clock. It is equipped with two levels of cache: a low latency 32KB L1 cache and a larger globally coherent total 8MB L2 cache that is partitioned among the cores. It offers a peak throughput of 1.2 Tflops (single-precision). Because the MIC architecture is based on x86, it provides a natural extension to the conventional x86 programming models. Thus we could use similar data and thread level implementation as on Core i7.

B. Assessment of Computational Burden

Figure 2(a) shows the overview of our CS implementation. The targeted CS reconstruction algorithm is composed of multiple iterations of 3D matrix arithmetics. We divide the application into six stages based on the loop structure (denoted as *Stage1*, *Stage2* and so on). Each stage performs a series of matrix computations such as element-wise additions and 3D FFTs. The pie-chart in Figure 2(b) shows the execution time breakdown of the key kernels. *FFT3D* (performed in *Stage2* and *Stage5*) is the most time-consuming and accounts for 46% of the total execution time. To achieve optimal performance, FFT requires architecture-specific optimization. Thus we use the best FFT libraries available for each architecture. Simple element-wise matrix arithmetics (*Matrix*) are the second most time consuming kernels. Because they stream large amount of data from/to the main memory, our optimizations focus on hiding latency and utilizing bandwidth efficiently. *Diff3D* (in *Stage1* and *Stage4*) calculates the differences from the original matrix to its shifted copy. Since the same data are used multiple times, we block the matrix to exploit data reuse in fast on-die memories. Finally, *GEval* (in *Stage1* and *Stage3*) performs transcendental operations such as division and exponentiation. On the GTX480 we take advantage of fast math functions; however, the performance gain due to the faster math is marginal because *GEval* comprises only 7% of the total execution time.

C. Experimental Data and Reconstruction Specifications

Five datasets are used for our experiments, whose volume size and memory footprint are: (256x64x64, 224MB), (256x160x32, 280MB), (256x160x80, 700MB), (256x160x84, 735MB), and (256x160x88, 770MB), for datasets 1 to 5, respectively. Datasets 1 and 2 were artificially generated, Dataset 3 represents a non-contrast-enhanced brain, and Datasets 4 and 5 represent the contrast-enhanced vasculature. All MRI data were acquired on a 3T GE Signa scanner (v.20) using an 8-channel head array using the CAPR acquisition sequence. Prior to reconstruction, view-sharing was performed on datasets 3-5, and background reference subtraction on datasets 4 and 5 as described in [12]. For all experiments, 5 outer and 15 inner (CG) iterations were executed under $W = 1$ (corresponding to 26 finite difference neighbors). ϵ-continuation (0.1 reduction) was performed at each outer iteration. Figure 1 shows the reconstruction result for dataset 5 obtained from our optimized CS implementation, which is visually identical to that obtained using the conventional CS implementation.

IV. RESULTS

We compare the performance of our CS implementation on Core i7, GTX480 and KNF, and provide performance analysis from a computer architecture perspective.

A. Performance Comparison: CPU, GPU and Intel's MIC

Figure 3 compares CS performance on three architectures: six-core Core i7, NVIDIA GTX480, and Intel KNF. We

normalize the speedups with respect to the quad-core Core i7 implementation (56 sec runtime) and show them only for datasets 1, 2, and 3. Datasets 4 and 5 perform very similar to dataset 3. For GTX480 and KNF, we show two speedup bars: one without data transfer overhead from the CPU host, and the other with the overhead. The data transfer overhead results in very small performance degradation of less than 1%, because CS spends significant time performing computation, and can hide the most of data transfer time.

The six-core CPU performs about 1.7x faster than the quad-core CPU, thanks to the increased number of cores. The GTX480 is 3.5x faster than the quad-core CPU for dataset 3. However, its performance exhibits big variance across datasets. The GTX480 shows more than 10x speedup for dataset 1, but shows only 2.1x speedup for dataset 2. It is due to FFT performance optimization. We believe that CUFFT 2.1 is specially optimized for small power of two datasets like dataset 1. Thus, FFT performance on dataset 1 is significantly better than on dataset 2 and 3. For dataset 3 (actual clinical data), KNF achieves 4.5x speedup over quad-core CPU, which is about 1.3x faster than GTX480.

Note that Core i7 and KNF are more efficient than GTX480 in terms of resource utilization. Though GTX480 has 10x peak flops and 3.5x peak bandwidth than Core i7, it only provides 2x performance. Also, while KNF delivers similar peak flops and 20% less peak bandwidth than GTX480, KNF shows 1.3x better performance than GTX480.

B. Impact of Performance Optimization

We demonstrate the importance of performance optimization by showing an example. Figure 4 shows the performance improvement of our CS implementation on a quad-core CPU, as we apply our optimization in sequence. The bar represents the execution time in seconds for each optimization step, and the line shows the corresponding relative speedup over the baseline. *Base* is a original single-core implementation of the algorithm, compiled with the highest level of optimization including auto-vectorization, function in-lining and inter-procedural optimization. As our first optimization, we replace the FFTW [25] used in the original implementation with the faster Intel MKL. This results in 1.15x speedup as represented by the second bar (*MKL*). Second, we hand-vectorize the codes that can not be auto-vectorized by the compiler. Hand-vectorization provides additional 1.19x speedup (*Vector*). Third, we apply cache blocking to exploit data reuse in the *Diff3D* kernel, which shows another 1.10x speedup (*Tile*). Through these three single-thread optimizations, we achieve overall 1.51x speedup over the baseline implementation. To take advantage of multiple cores/threads, we parallelize the application. For *FFT3D*, we use the parallel implementation of the MKL library, and for the other kernels we hand-parallelize using the OpenMP library. Parallelization achieves another 2.14x speedup on four cores over the single-core baseline. Overall, by combining the single-thread optimization and the multi-thread parallelization, we achieve 3.21x performance improvement from the baseline implementation,

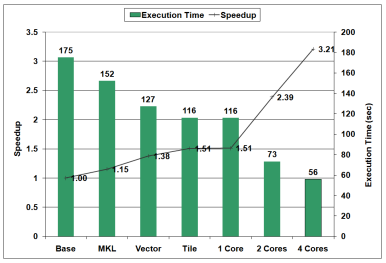


Fig. 4. Performance Optimization on quad-core Core i7

which reduces the total execution time from 175 seconds (*Base*) to 56 seconds (*4 Cores*).

To optimize for MIC, we applied almost identical techniques, because MIC is a x86-based architecture and hence supports the same programming model. Optimization for GTX480 required additional efforts, because it has a different architecture with a different programming model.

V. Conclusion

In this work, we have shown that advanced computing architectures can facilitate significant improvements in the performance of CS MRI reconstructions, and particularly that optimized use of modern many-core architectures cans diminish the computational barrier associated with this class of techniques. This suggests that as many-core architectures continue to evolve, CS methods can be employed in routine clinical MRI practice. Although CE-MRA was targeted in this work, the implication of the results apply to many other MRI applications as well as other areas in medical imaging.

ACKNOWLEDGEMENTS

The authors thank David Holmes III and Stephen Riederer for their help in establishing the collaboration between Mayo Clinic and Intel Corporation.

References

[1] E. Candès et al. *IEEE TIT*, 52(2):489–509, 2006.
[2] D. Donoho. *IEEE TIT*, 52(4):1289–1306, 2006.
[3] M. Lustig et al. *MRM*, 58(6):1182–1195, 2007.
[4] M. Lustig et al. In *Proc. ISMRM*, page 695, 2006.
[5] T. Çukar et al. *MRM*, 61(5):1121–1131, 2009.
[6] J. Trzasko et al. In *Proc. ISBI*, pages 274–277, 2009.
[7] A. Bilgin et al. In *Proc. ISMRM*, page 337, 2008.
[8] M. Doneva et al. In *Proc. ISMRM*, page 336, 2008.
[9] Y.-C. Kim et al. *MRM*, 61(6):1434–1440, 2009.
[10] C.-H. Chang and J. Ji. In *Proc. EMBS*, pages 2684–2687, 2009.
[11] J. Trzasko et al. In *Proc. ISMRM*, page 347, 2010.
[12] C. Haider et al. *MRM*, 60(3):749–760, 2008.
[13] K. Pruessmann et al. *MRM*, 46(4):638–651, 2001.
[14] A. van den Bos. *IEE Proc.: Vis., Image, and Sig. Proc.*, 141(6):380–383, 1994.
[15] D. Brandwood. *IEE Proc.: Comm., Radar, and Sig. Proc.*, 130(1):11–16, 1983.
[16] C. Vogel and M. Oman. *IEEE TIP*, 7(6):813–824, 1998.
[17] R. Chartrand. *IEEE SPL*, 14(10):707–710, 2007.
[18] E. Borisch et al. In *Proc. ISMRM*, page 1492, 2008.
[19] Nvidia's Next Generation CUDA Compute Architecture: FERMI. 2009.
[20] NVIDIA CUDA Programming Guide 2.3. 2009.
[21] CUDA CUFFT Library 2.1. 2008.
[22] Nukada et al. In *Proc. SC*, pages 1–10, 2009.
[23] L. Seiler et al. *ACM Trans. Graphics*, 27(3):1–15, 2008.
[24] K. Skaugen. *ISC 2010 Keynote*.
[25] M. Frigo et al. *Proc. IEEE*, 93(2):216–231, 2005.