High-Performance Gridding on Modern x86-based Multi-core Systems for 3D Non-Cartesian MRI

Dhiraj D. Kalamkar[1], Joshua D. Trzasko[2], Srinivas Sridharan[1], Mikhail Smelyanskiy[3], Daehyun Kim[3], Yunhong Shu[4], Matt A. Bernstein[4], Bharat Kaul[1], Pradeep Dubey[3], and Armando Manduca[2]

[1]Parallel Computing Lab, Intel Labs, Bangalore, KA, India, [2]Mayo Clinic, Rochester, MN, United States, [3]Parallel Computing Lab, Intel Labs, Santa Clara, CA, United States, [4]Department of Radiology, Mayo Clinic, Rochester, MN, United States

Overview: With increasing usage of higher-resolution acquisitions, more receiver channels, and iterative reconstruction strategies, the ability to quickly and accurately transform an image to and from k-space, known as "reverse gridding" and "gridding", is crucial for non-Cartesian MRI applications. "Reverse gridding" is realized via the forward non-uniform fast Fourier transform (NUFFT) [1], and comprises three operations: 1) image-domain weighting; 2) oversampled FFT; and 3) convolution interpolation. "Gridding" is the adjoint of this operation series. Despite significant advances (e.g., [2]), NUFFT evaluation, and particularly convolution interpolation, remains computationally challenging. Moreover, prevalent race conditions arising during scatter operations make parallelization of the adjoint NUFFT non-trivial. Thread privatization [3] can circumvent this problem, but with limited scalability. Specialized hardware implementations, such as on GPUs, may be memory limited [4], require major code adaptation [5], and/or be constrained to specific sampling trajectories [6]. In this work, we propose variable-size geometric partitioning along with a barrier-free task queue and selective privatization to parallelize the adjoint NUFFT convolution operation. We then demonstrate that this implementation strategy is substantially faster than contemporary x86 implementations, and computationally competitive with state-of-the-art GPU implementations.

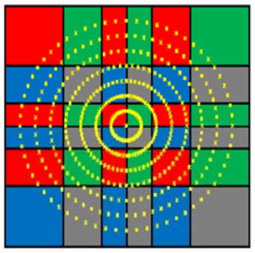
FIG 1: Variable size partitioning and task scheduling

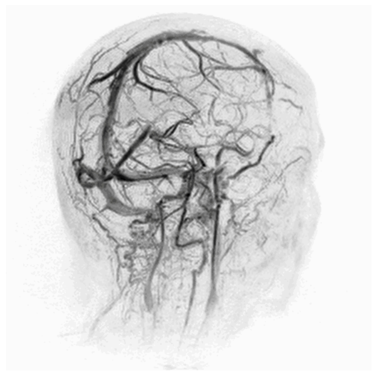
FIG 2: Oblique MIP of a SWIRLS CE-MRA. A contemporary parallelized gridding reconstruction [3] on an x86 (Intel® Xeon® X5670) system required ~12 s – on identical hardware, our proposed implementation of this same reconstruction takes only ~2.4 s.

Methods: Our preprocessing initiates by computing the histogram of the non-Cartesian sample positions in each dimension, which is used to partition the Cartesian grid corresponding to the targeted reconstruction into variable size blocks (Fig. 1). For each partition, we then reorder the subset of non-Cartesian samples to achieve better cache locality, and create a corresponding "task". Finally, we create a task dependency graph to schedule these tasks (TaskQ) such that there are no race conditions while scattering non-Cartesian samples onto the Cartesian grid. If the minimum width of a partition is limited to be greater than or equal to the width of the convolution window (W), samples from two tasks with partitions separated by at least one other partition can be processed in parallel. As shown in Fig. 1, our task coloring is similar to that in [7]; however, we exploit localized dependency information to circumvent the need for barriers, and use a priority queue to schedule longer running tasks earlier. Due to the limit on the minimum width of a partition, and the non-uniform distribution of samples, even a task associated with a small partition may have a very large number of samples contained in it. If a task has both a significantly higher number of samples relative to the average number per task and a long dependence chain before it can be scheduled, we execute it at the beginning using privatization. However, reduction onto a Cartesian grid is still performed per the original scheduling algorithm. For a typical 3D clinical volume, preprocessing executes in less than 1 s. Thus, for direct reconstructions, it is easily performed offline; and for iterative reconstruction strategies, it can even be performed online since the overhead is quickly amortized. Like "wisdom" in the FFTW library [8], our optimization data can also be recycled for later use with identical sampling trajectories.

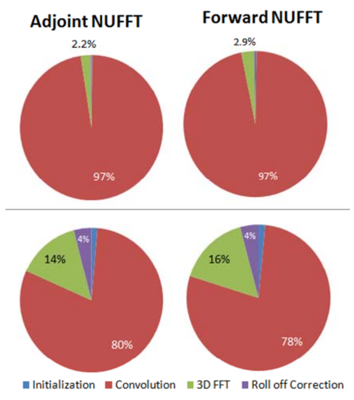

FIG 3: Workload distribution (a) before and (b) after optimization

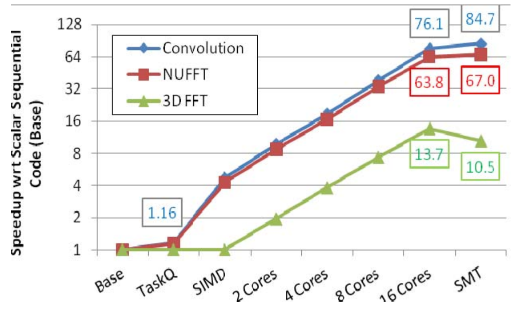

FIG 4: Speedup from successive optimizations

Results: Figure 2 shows a gridding reconstruction (OF=1.25, W=5 Kaiser-Bessel [2]) of an 8-channel undersampled SWIRLS MRA [3] of the neurovasculature (S=8047, K=512, N=240). Shu et al. [3] demonstrated a privatization-based parallel C++ implementation requiring only 1.4 s per adjoint NUFFT, or ~12 s to reconstruct the entire data set, on a dual socket 12-core Intel® Xeon® X5670 (WSM12C) system. By incorporating our preprocessing strategy, a like-parameterized adjoint NUFFT takes only 0.28 s on the same system, or about 2.4 s for the entire reconstruction – a 5x speedup. Migration to a dual socket 16-core Intel® Xeon® E5-2670 (SNB16C) system reduces this time to only 0.18 s per adjoint NUFFT. Fig. 3 shows a breakdown of execution time on SNB16C system for both the forward and adjoint NUFFT before (scalar sequential code) and after (optimized parallel code) our applied optimizations for a magnetization-prepared SWIRLS [9] example (S=17300, K=512, N=240). Although convolution still dominates the NUFFT, its relative computational share is substantially reduced. Fig. 4 shows the speedup resulting from successive optimizations (including SIMD and Simultaneous Multithreading (SMT)) for the NUFFT Gram operator (forward+adjoint), for the same MP-SWIRLS data set. Comparable speedups have been observed for simulated 3D radial, 3D random, and stack-of-spiral data sets [10]. Table 1 compares our x86-based strategy to Nam et al.'s [4] GTX480 NUFFT implementation for a 3D "kooshball" acquisition (S=9000, K=344, N=344). The GTX480 slightly bests our WSM12C implementation; however, our SNB16C implementation is 1.4x faster. This demonstrates that, for the NUFFT, computational performance normally associated with state-of-the-art GPU implementations is in fact realizable on readily available x86 hardware.

	Multi-core CPUs		GPU
	WSM12C	SNB16C	GTX480
ADJ NUFFT (sec)	0.93	0.58	0.94
FWD NUFFT (sec)	0.87	0.54	0.66
Total (sec)	1.79	1.11	1.6
Speedup	0.89x	1.44x	1.00x

Table 1: Comparison of performance between multi-core CPU based implementation and GPU based implementation

Summary: We have proposed a novel preprocessing and parallelization strategy for fast execution of NUFFTs on modern x86-based systems, and demonstrated that this approach is computationally competitive with advanced hardware implementations such as GPUs. Beyond direct non-Cartesian reconstructions, our proposed approaches have obvious applicability for iterative reconstruction strategies, such as those utilized in parallel imaging [11] and/or compressive sensing [12] applications. This allows the practical employment of such methods without the need to migrate to advanced hardware implementations.

References: [1] J. Fessler and B. Sutton, IEEE TSP 51:560-4, 2003; [2] P. Beatty et al., IEEE TMI 24:799-808, 2005; [3] Y. Shu et al., ISMRM 2011:2656; [4] S. Nam et al., ISMRM 2011:2548; [5] T. Sorensen et al., IEEE TMI 27:538–47, 2008; [6] T. Schiwietz et al., SPIE MI 2006:61423T; [7] Y. Zhang et al., EURO-PAR 2010:125–36; [8] http://www.fftw.org; [9] Y. Shu and M. Bernstein, ISMRM 2010:2900; [10] Submitted to IEEE IPDPS 2012; [11] K. Pruessmann et al., MRM 46:638-51, 2001; [12] M. Lustig et al., MRM 58:1182-95, 2007