MemXCT: Memory-Centric X-ray CT Reconstruction with Massive Parallelization

Mert Hidayetoğlu[1], Tekin Biçer[2], Simon Garcia de Gonzalo[1], Bin Ren[3], Doğa Gürsoy[2], Rajkumar Kettimuthu[2], Ian T. Foster[2], and Wen-mei W. Hwu[1]
[1]University of Illinois at Urbana-Champaign, IL 61810, USA
[2]Argonne National Laboratory, IL 60439, USA
[3]College of Willam & Mary, VA 23185, USA

ABSTRACT

X-ray computed tomography (XCT) is used regularly at synchrotron light sources to study the internal morphology of materials at high resolution. However, experimental constraints, such as radiation sensitivity, can result in noisy or undersampled measurements. Further, depending on the resolution, sample size and data acquisition rates, the resulting noisy dataset can be terabyte-scale. Advanced iterative reconstruction techniques can produce high-quality images from noisy measurements, but their computational requirements have made their use exception rather than the rule. We propose here a novel memory-centric approach that avoids redundant computations at the expense of additional memory complexity. We develop a system, MemXCT, that uses an optimized SpMV implementation with two-level pseudo-Hilbert ordering and multi-stage input buffering. We evaluate MemXCT on various supercomputer architectures incolving KNL and GPU. MemXCT can reconstruct a large (11K×11K) mouse brain tomogram in ~10 seconds using 4096 KNL nodes (256K cores), the largest iterative reconstruction achieved in near-real time.

ACM Reference Format:
Mert Hidayetoğlu[1], Tekin Biçer[2], Simon Garcia de Gonzalo[1], Bin Ren[3], Doğa Gürsoy[2], Rajkumar Kettimuthu[2], Ian T. Foster[2], and Wen-mei W. Hwu[1]. 2019. MemXCT: Memory-Centric X-ray CT Reconstruction with Massive Parallelization . In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3295500.3356220

1 INTRODUCTION

X-ray computed tomography (XCT) is a widely used nondestructive 3D imaging technique for observing and understanding the internal morphology of samples and materials. Synchrotron light sources, such as the Advanced Photon Source (APS), can provide high-brilliance x-rays that enable tomographic imaging of centimeter sized samples at sub-micrometer (μm) spatial resolution and these experiments can generate few GBs up to TBs of data volumes in short time with the typical pixelated detectors that can run at 16GiB/s [1]. However, the quality of data collected from CT experiments depends heavily on factors such as radiation exposure time (dose) and target spatial resolution. Much effort has been devoted to develop and implement advanced reconstruction algorithms to improve image quality when collected data is noisy or imperfect (e.g., due to limited dose.)

Modern x-ray reconstruction approaches involve either (i) direct solvers based on analytical inversion or (ii) iterative solvers when one can accurately model experimental conditions or constraint the solution based on prior knowledge of the sample. Analytical methods such as the filtered backprojection (FBP) algorithm are computationally efficient, but reconstruction quality is often poor when measurements are noisy or undersampled. Iterative methods, on the other hand, can use advanced optimization and regularization techniques to handle inherent noise in x-ray measurements [3–9]. However, these methods are computationally more demanding than the analytical methods, since forward and backprojection operations require many (irregular) accesses and computation at each iteration. Thus, efficient implementation and parallelization are crucial for high-quality large-scale image reconstructions.

Most state-of-the-art reconstruction software and libraries can perform parallel reconstruction to some extent. However, these implementations mostly rely on data replication [10, 11] and/or redundant computations [12, 13], each of which significantly limits the runtime performance of the system due to repeated computation of intermediate indices of ray tracing and irregular data accesses.

We propose here a novel memory-centric approach to avoid redundant computation by removing on-the-fly operations at the expense of greater memory complexity. Our system, MemXCT, uses an optimized sparse matrix-vector multiplication (SpMV) implementation with *multi-stage buffering* and *two-level pseudo-Hilbert ordering* to optimize data communication, partitioning, and accesses at different analysis levels and on both measurement (sinogram) and reconstructed (tomogram) data.

MemXCT memoizes on-the-fly operations and therefore requires more memory than alternatives. However, its overall per-node memory footprint decreases linearly with increasing computing resources, promoting scalability with massive parallelization and enabling iterative reconstruction of extremely large datasets with efficient resource use. Further, MemXCT stores intermediate data structures in compressed format to minimize memory footprint.

To illustrate the practical implications of MemXCT optimizations, we show in Fig. 1 a 2D tomogram from a 3D mouse brain image with total size 11 293×11 293×11 293, obtained by reconstructing a sinogram with 4501×11 293 dimensions. This single-slice reconstruction was generated with 30 conjugate gradient (CG) iterations in about 10 seconds on 4096 KNL nodes (256K cores) using MemXCT. This high speed capability makes it feasible, for the first time, to apply

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356220

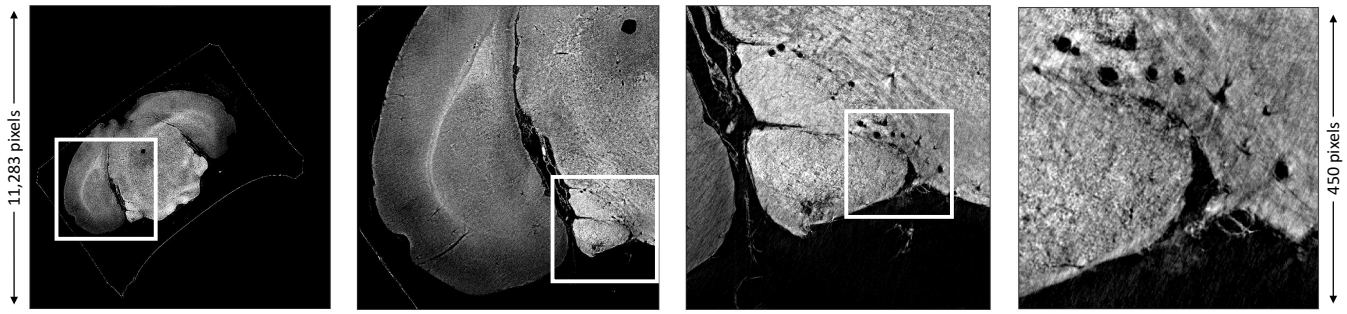

Figure 1: Multiple zooms on a single 11 293×11 293 2D slice of a mouse brain image generated by reconstruction, with the methods described here, from a single sinogram with size 4501×11 293. Data were collected by Dyer et al. [2] at the APS. Single-slice reconstruction with 30 CG iterations on 4096 KNL nodes takes ~10 seconds. Memory footprint of the application is 10.2 TiB. The full mouse brain consists of 11 293 slices.

advanced iterative reconstruction algorithms to extremely large CT datasets. The multi-scale nature of the reconstruction is presented by zooming progressively to the brain arteries, where great detail can be seen. High reconstruction quality at such scales is important as subsequent data analysis steps, such as segmentation of blood vessels and myelinated axon tracts, are highly dependent on the quality of the reconstruction.

The principal contributions of this paper are as follows:

- We propose a memory-centric approach that uses an *extended SpMV* to address race conditions due to scatter operations. Our implementation replaces scatter operations on sinogram and tomogram domains with gather operations.
- We implement a *two-level pseudo-Hilbert ordering* to organize communication and data access pattern, further improving communication performance, utilization of different levels of memory hierarchy, and cache utilization.
- We introduce a *multi-stage input buffering*. Domain partitioning improves process-level data communication and locality; multi-stage buffering enables data reuse in first-level caches and minimizes memory latency.
- We extensively evaluate *MemXCT* performance on both KNL and GPU systems up to 4096 nodes, and show architecture-specific considerations and optimizations.

2 BACKGROUND AND MOTIVATION

We first explain XCT experiments and their data acquisition process. Then, we describe the steps involved in iterative image reconstruction. Finally, we explain computational bottlenecks in iterative reconstruction approaches and introduce our solutions.

2.1 Data Acquisition and Measurement Process

In an XCT experiment, the sample is placed on a rotation stage and illuminated by an x-ray source, while collecting 2D images through a detector as the sample is being rotated: see Fig. 2. The mathematical model of the measurement process is based on Beer's law [14] that describes interaction between X-rays and matter: $I_\theta(s) = I_0(s)\exp[-p_\theta(s)]$, where $I_0(s)$ is the incident x-ray illumination on the sample and $I_\theta(s)$ are the collected measurements at a number of different θ angles as a result of a tomographic scan.

In Fig. 2, a set of collected projections from a sample is shown with p_{θ^i}. Considering parallel beam geometry, the sinogram $p_\theta(s)$ is a cross section of projections (shown in blue grid on p_{θ^*}) that consists of I_θ measurements from f. The goal of a tomographic reconstruction algorithm is to recover 2D image slice (tomogram) from its corresponding sinogram $p_\theta(s)$.

2.2 Iterative Formulation

Tomographic reconstruction solves the problem

$$\hat{x} = \underset{x \in C}{\operatorname{argmin}} \|y - Ax\|^2 + R(x), \tag{1}$$

where $\hat{x}$ is the reconstructed tomogram, A is the forward model, y is the sinogram, $R(x)$ is a regularizer functional, x is the search variable, and C is a constraint on x.

Almost all iterative solvers based on gradient descent perform three common steps in each iteration as depicted in Fig. 2. Specifically, for iteration i: first, the residual $r_i = y - Ax_i$ is found through *forward projection*; second, the gradient is found through *backprojection* as $\nabla\|y-Ax_i\|^2 = A^T r_i$; and finally, the candidate solution is updated in the negative-gradient direction as $x_{i+1} = x_i - \alpha\nabla\|y-Ax_i\|^2$, where α is the step size. Iterative approaches can also involve additional updates due to regularizer $R(x)$ and constraint C. However, this paper focuses on the common computational costs involving aforementioned steps.

2.3 Forward and Backprojection

Forward and backprojections are the two most computationally demanding kernels in iterative tomographic reconstruction approaches. Many reconstruction libraries, e.g. Trace and TomoPy[10, 13], implement Siddon's algorithm [15] to perform ray tracing on tomogram domain so that the exact length and weight information on each voxel can be computed for each intersecting x-ray. For forward model, these information are used for computing the residuals, whereas for backprojection they are used for gradient and update operations. Since storing voxel indices and length information for all rays and voxels require significant memory, most

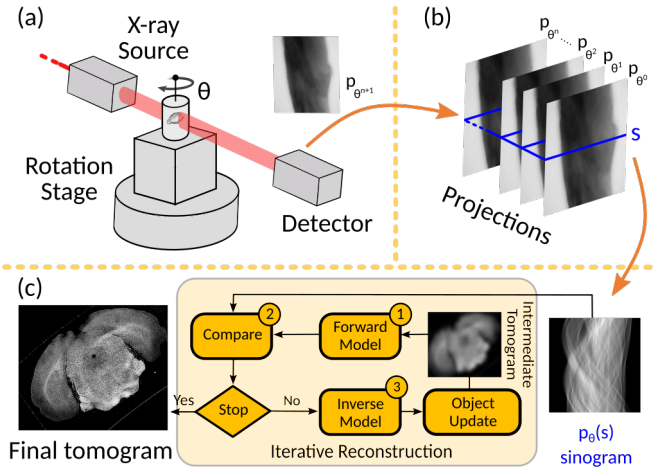

Figure 2: Fig. (a) illustrates an experimental setup where a target object is illumined with a synchrotron light source. Photons are attenuated by different amounts according to the object's attenuation coefficient profile and the photons' travel distance within the object. Fig. (b) presents how the detector measurements are organized and stored as projections (from different angles). Once the experiment is completed, the sinograms are extracted from the projections, and the images corresponding to the object are iteratively reconstructed as shown in Fig. (c).

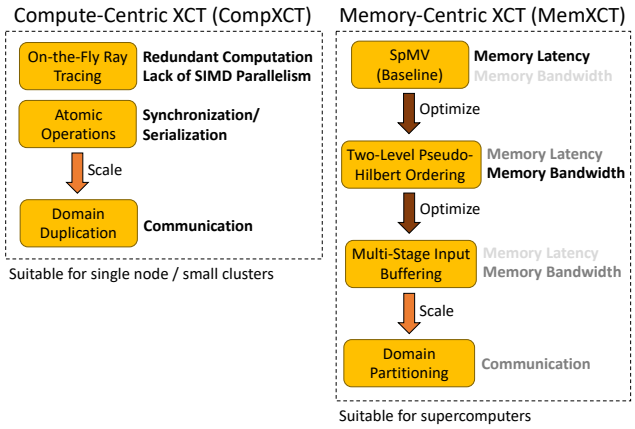

Figure 3: Memory-centric and compute-centric approaches for XCT and their areas for potential optimizations. Shades of black color indicate higher contribution to performance bottleneck.

state-of-the-art approaches perform ray tracing computations on-the-fly. For instance, a sinogram with 1501 x 2048 dimensions require 56GiB memory to store intermediate data structures. Listing 1 illustrates the high level implementation of iterative reconstruction and repetitive computations of indices and lengths arrays. We term reconstruction algorithms that compute on-the-fly ray tracing information as *compute-centric XCT* (CompXCT).

Listing 1: High Level Implementation of CompXCT

```
for(int i = 0; i < num_iters; ++i){ //iterations
 for(int j = 0; j < sinogram_rows; ++j){
  float theta = rotations (j); //Corresponding θ for row j
  for(int k = 0; k < sinogram_cols; ++k){
   int **indices = intersecting_voxels (j, k, theta, tomogram);
   float **lengths = compute_lengths (j, k, theta, tomogram);
   float m = sinogram[j, k]; //Measured data for simulated ray
   float residual = forward_model (m, indices, lengths, tomogram);
   // Apply inverse model and then update tomogram
   backprojection (residual, indices, lengths, tomogram);
  }
 }
}
```

2.4 Computational Bottlenecks and Overview of Our Solution: MemXCT

Fig. 3 illustrates the techniques used for traditional CompXCT and proposed MemXCT approaches in yellow boxes. The existing bottlenecks and their influence are given next to the areas for potential optimizations with different shades of black, in which darker shades indicate higher contribution to performance bottleneck.

CompXCT eliminates the need for storing intermediate data structures; however, since these data structures are recalculated for each iteration, they introduce additional computational complexity to the execution. Further, on-the-fly calculation of indices limits the optimizations for vectorization as repeated sorting and padding operations nullify any performance benefit. As a remedy, MemXCT embraces a memory-centric approach that memoizes ray-tracing operations and performs efficient SpMV using compressed (vectorizable) data structures on both sinogram and tomogram domains. Further, the irregular data access patterns are normalized using pseudo-Hilbert ordering and first level cache utilization are improved using multi-stage buffering. These optimizations result in high-performance computations and converts performance bottlenecks from computation to memory.

The parallelization of CompXCT approaches is typically based on x-rays, where each measured ray can independently be projected on tomogram domain. This type of parallelization performs well for forward projection kernel (residual computation) since the main operation is reading the intersected voxel values from tomogram domain, i.e. rays perform *gather operations* during forward projection. However, subsequent backprojection and update steps require synchronization among parallelized rays since there can be many updates from different rays on the same voxel, i.e. rays perform *scatter operations* during backprojection and updates. Recent work deals with race conditions by either applying atomic operations [16] or duplicating the pixel domain across threads/processes and then performing a reduction [10, 11]. Unfortunately, the performance of atomic operations hinges on hardware implementations, which differ substantially across architectures [17], and typically results in significant performance degradation on massively parallel architectures. Domain duplication is also impractical for GPU-like architectures, since each parallel unit requires a replica of the domain and the total memory footprint almost always exceed available resources. Further, distributed memory parallelization using duplication can result in redundant communication cost [10].

In contrast, MemXCT partitions both sinogram and tomogram domains among parallel units and transforms scatter operations to gathers. Note that this transformation can result in irregular data accesses on both domains, however our pseudo-Hilbert index ordering coupled with multilevel buffering amortize performance penalty

due to irregular data accesses. MemXCT also exploits Hilbert ordering (hence named two-level pseudo-Hilbert ordering) for process-level domain partitioning, which results in efficient communication and better connectivity between processes [18, 19]. We describe these optimization in detail in the following section.

3 MemXCT OPTIMIZATIONS

In this section, we provide detailed information about MemXCT. First, we explain the *baseline implementation* of MemXCT, in which we focus on explicit SpMV operations. Then, we present our system optimizations that use *two-level pseudo-Hilbert ordering* and *multi-stage input buffer*. We also explain our distributed memory parallelization considerations with MPI.

3.1 Baseline Implementation

MemXCT performs forward and backprojection operations as explicit SpMV operations to remove redundant and inefficient computations. Listing 2 shows the baseline Compressed Sparse Row (CSR) SpMV kernel used for both forward and back projection. In forward projection, `x` and `y` correspond to tomogram and sinogram data, respectively, and vice-versa in backprojection. In either case, `ind` and `val` data correspond to precomputed pixel-ray intersection indices and lengths that are reused to avoid redundant computation.

Listing 2: Baseline MemXCT Kernel

```
1  #pragma omp parallel for schedule(dynamic,partsize)
2  for(int i = 0; i < numrow; ++i){
3    float acc = 0;
4    //vectorize
5    for(int j = displ[i]; j < displ[i+1]; ++j)
6      acc += x[ind[j]]*val[j];
7    y[i] = acc;
8  }
```

3.1.1 Regular and Irregular Accesses. Each fused multiply-add (FMA) operation in Listing 2 performs three important memory accesses: `ind` and `val` are *regular*, and `x` is *irregular*. The *regular* accesses are sequential and hence exhibit low memory latency. The *irregular* accesses to `x` exhibit long latency due to high L2 miss rates. MemXCT addresses this problem by improving cache reuse through novel data layout techniques that increase locality, as illustrated in Section 3.2.

3.1.2 Row Partitioning & Parallelization. Listing 2 involves gather operations only and hence is suitable for massive parallelization. MemXCT parallelizes the outer loop among *row partitions*. On KNL, partitions are distributed across dynamically scheduled OpenMP threads. On GPU, each partition corresponds to a CUDA thread block. Each OpenMP thread processes many row partitions, whereas each CUDA thread processes a single row in a partition.

3.1.3 Vectorization on KNL. MemXCT enables efficient instruction-level parallelization through vectorization of the inner loop in Listing 2. Each KNL vector-processing unit (VPU) can load multiple regular and irregular data, and multiply them in one step with an AVX-512 instruction. Then, they perform parallel reductions on partial data, and add results to the accumulator. We provide necessary data alignments for efficient vectorization.

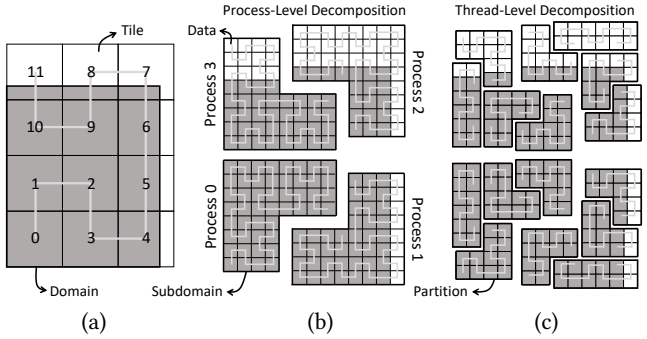

Figure 4: Two-level pseudo-Hilbert ordering and domain decomposition: (a) the first level, and then the lower level on (b) process and (c) thread levels.

3.1.4 Coalesced Memory Access on GPU. Our GPU implementation modifies Listing 2 to use the ELL (column-major) storage format instead of CSR. Transposed ELL data structures provide coalesced memory access through consecutive threads accessing consecutive memory locations. We minimize redundant computations and memory accesses due to ELL format by performing zero padding on thread-block (i.e., partition) level, rather than on matrix level.

3.2 Two-Level Pseudo-Hilbert Ordering

MemXCT implements a two-level pseudo-Hilbert ordering on both tomogram and sinogram data of arbitrary size. Fig. 4 shows an example for a 13×11 domain. First, the domain is tiled with a minimum number of equi-sized square tiles with dimension a power of two. In the figure, tile size is 4×4 and 12 tiles are used to cover the 13×11 domain. These tiles are indexed with a Hilbert ordering for rectangular domains [20], as shown in Fig. 4(a). A second-level Hilbert ordering is then applied to the data within each tile. Necessary rotations are performed to provide data connectivity among tiles, so as to achieve both data locality and connectivity for domain decomposition at the process and thread levels, as discussed next. The process-level domain decomposition is essential for MPI parallelization, as discussed in Section 3.4.

3.2.1 Irregular Data Access Patterns. As shown in Fig. 5, the processing of a single sinogram or tomogram results in a linear or a sinusoidal memory access footprint, respectively. In this example, they perform 25 and 30 accesses on respective domains. With row-major (naive) ordering of 2D data and 64 B cache line, each row in Fig. 5 would correspond to a single cache line. In this case, both tomogram and sinogram data would have 16 cache misses, yielding miss rates of 64% and 53%, respectively. Row-major (or column-major) ordering of 2D domains is very inefficient for XCT because of its poor cache locality. That is, a single cache line does not provide enough data reuse except on a few instances.

3.2.2 Cache Locality. In Fig. 5, Hilbert ordering maps each cache line to a 4×4 block in a 2D domain, increasing cache data reuse. As a result, cache misses are reduced to six and seven, yielding rates of 24% and 23% for forward and back projection, respectively. Hilbert ordering is portable to different cache-line sizes thanks to its recursive nature. The cache locality provided by Hilbert ordering

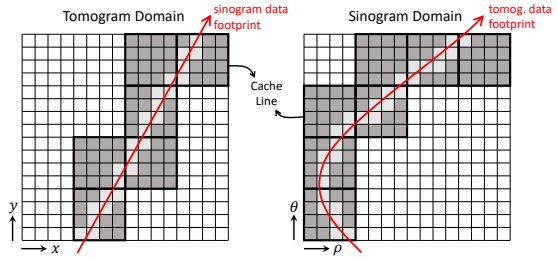

Figure 5: Data access patterns on 2D domains.

eliminates the memory latency bound of MemXCT to a great extent, as results show in Section 4.2.2.

3.2.3 Partition Locality. Hilbert ordering not only provides cache locality but also partition locality. That is, the outer loop of SpMV in Listing 2 is partitioned with respect to the index of y, as described in Section 3.1.2. In MemXCT, each partition remains connected at both the process and thread levels, thanks to the connectivity provided by two-level pseudo Hilbert ordering. In contrast, a Morton ordering would yield disconnected partitions, since it does not guarantee adjacent memory locations to have adjacent locations in the 2D domain. Partition locality is essential for our further optimizations.

3.3 Multi-Stage Input Buffering

MemXCT implements a multi-stage buffering mechanism for: (1) further reducing L2 cache miss rates of irregular data by explicit staging of corresponding accesses on an L1 buffer, and (2) reducing memory bandwidth consumed for index data by using two-byte addressing. Listing 3 shows SpMV kernel with input buffering, where stagedispl and stagenz arrays correspond to starting points of (multi-stage) buffers and number of (nonzero) elements in buffers, respectively.

Listing 3: Optimized MemXCT Kernel

```
1  #pragma omp parallel for schedule(dynamic)
2  for(int part = 0; part < numparts; ++part){
3    float output[partsize] = {0};
4    float input[buffsize];
5    for(int stage=partdispl[part];stage<partdispl[part+1];++stage){
6      int start = stagedispl[stage];
7      //vectorize
8      for(int i = 0; i < stagenz[stage]; ++i)
9        input[i] = x[map[start+i]];
10     for(int j = 0; j < partsize; ++j){
11       int start = stage*partsize;
12       //vectorize
13       for(int i = displ[start+j]; i < displ[start+j+1]; ++i)
14         output[j] += input[ind[i]]*val[i];
15     }
16   }
17   int start = part*partsize;
18   //vectorize
19   for(int i = 0; i < partsize; ++i)
20     if(start+i < numrows)
21       y[start+i] += output[i];
22 }
```

3.3.1 Data Reuse from Input Buffer. Fig. 6(a) shows memory access footprints for processing a 64×64 tomogram and sinogram partition in a 256×256 tomogram and sinogram domain, respectively. The tomogram partition reads from the sinogram domain and the sinogram partition from the tomogram domain. Data access footprints are shown in tomogram and sinogram domains. Darker shades represents higher data reuse. For processing each partition, MemXCT explicitly moves required data from memory to an L1

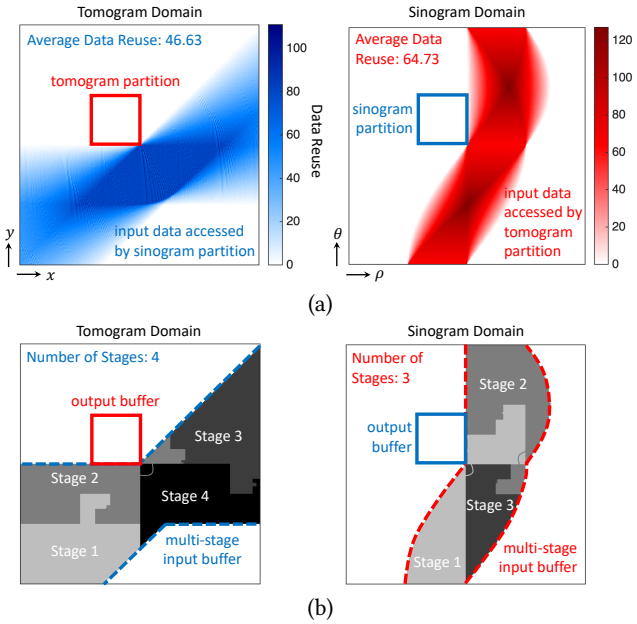

Figure 6: (a) Tomogram and sinogram partitions and respective data access footprints, and (b) multi-stage buffer shapes.

buffer through map at line 9 of Listing 3. Then it reuses buffered data for irregular accesses and hence performs less memory access.

3.3.2 Multi-Staging. In a single-stage (naive) buffering, the required buffer size grows with the problem geometry. If the buffer is too large, it cancels the buffering benefit since it leaks into L2 or even memory [21]. As a remedy, MemXCT implements a multi-stage buffering strategy with a constant buffer size. In this case, irregular data is accessed through multiple buffer stagings. Stages are determined with respect to Hilbert ordering, ensuring data locality. As an example, Fig. 6(b) shows mapping of buffer stages on 2D tomogram and sinogram domains. With a buffer size of 32 KB, MemXCT performs irregular accesses through four and three stages for projection and backprojection, respectively. On KNLs, buffer size should be smaller than 32 KB L1 cache to avoid L2 leak. On GPUs, the buffer is allocated through CUDA shared memory, guaranteeing reuse from L1 cache.

3.3.3 Staging Overhead. Input buffering involves a staging overhead. On KNL, an OpenMP thread stays idle while waiting staging to be completed. Similarly on GPU, CUDA threads across warps stall for synchronization before and after each staging. Input buffering also consumes some additional memory bandwidth for reading map data. The next two subsections describe two techniques for hiding buffering overhead and also to save some bandwidth.

3.3.4 Overlapping Staging and FMAs. MemXCT multi-stage buffering lends itself well to be used by the underlined hardware to hide staging overhead. On KNL, this is done through SMT (simultaneous multithreading): when there are multiple threads running on a single core, the hardware scheduler finds overlapping opportunities across buffer stagings and FMAs among threads. As a result,

MemXCT enables effective SMT utilization. Similarly, GPU hardware also takes advantage of overlapping through block scheduling on SMs (streaming multiprocessors).

3.3.5 Saving Memory Bandwidth. Hilbert ordering and input buffering eliminate most of the memory latency due to irregular data accesses, and therefore MemXCT is bounded by memory-bandwidth consumed by regular data. To alleviate this bottleneck, we use 16-bit addressing to access input buffer (see `ind` at line 14 on Listing 3), rather than 32-bit addressing (as in `ind` at line 6 on Listing 2). 16-bit addressing can address buffer sizes up to 256 KB. This saves 25% of total bandwidth consumption of regular data, and provides additional speedup (see Section 4.2.3 for results.)

3.4 MPI Parallelization

Traditionally, one domain (either tomogram or sinogram) is partitioned while the other is duplicated across all processes. MemXCT partitions both tomogram and sinogram domains by distributing tiles evenly across MPI processes, as seen in Fig 4(b). Each process is responsible for a single *tomogram subdomain* and a single *sinogram subdomain*. Each subdomain consists of a single or several tiles, e.g., subdomain 0 consists of tiles 0–2. While processes are not perfectly load balanced, it can be improved by finer tile granularity at the cost of more preprocessing.

3.4.1 Sparse Communications. MemXCT communicates only necessary data through `MPI_Alltoallv`. To explain, Fig. 7(b) shows communication footprint of two 256×256 tomogram subdomains shown in Fig. 7(a). For example, tomogram subdomain 7 interacts only with sinogram subdomains 1, 2, 8–11, 13, and 14. Fig. 7(c) shows the corresponding communication matrix, where each entry represents communication between two processes. Communication size between each pair depends on the interaction footprint, e.g., process 7 sends more data to process 1 than 14, as seen in Fig. 7(d). Partition locality provided by two-level pseudo-Hilbert ordering (described in Section 3.2) minimizes the footprint, and hence increase data reuse and reduce communications.

3.4.2 Overlapped Interactions. MemXCT communicates sinogram data rather than tomogram data because: (1) sinogram data is smaller in many applications, and (2) it yields more data reuse as compared to tomogram data. In forward projection, tomogram subdomains send partial sinogram data to corresponding sinogram subdomains where overlapped data is reduced, e.g., process 8, 9, and 11 reduces partial sinogram data received from process 7 and 10. In backprojection, processes duplicates overlapped sinogram data and send them to interacted processes which perform backprojections on their respective tomogram subdomains. The communication matrix for backprojection is the transpose of the one in Fig. 7(c). The total amount of communications and load balancing for all processes is shown in Fig. 7(e).

3.4.3 Parallelization Overhead. MemXCT parallelization of forward projection mathematically corresponds to a factorization of the projection matrix as $A = RCA_p$. As a result, forward projection can be modeled as three fundamental steps: A_p, C, and R which correspond to partial forward projection, communication, and reduction operations. Backprojection can also be seen as simply

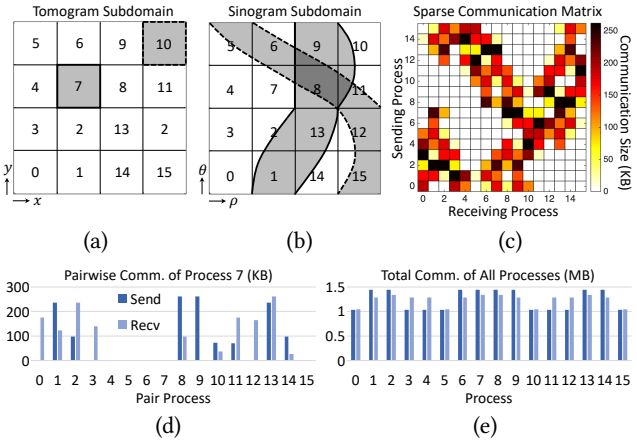

Figure 7: Communication footprint of two (a) tomogram subdomains on (b) sinogram subdomains. (c) Resulting communication matrix for all processes. (d) The amount of outgoing/incoming data from/to process 7 to/from others. (e) Total incoming and outgoing data for all processes.

$A^T = A_p^T C^T R^T$. The multiplication costs of A_p and original A are the same because they involve the same number of $O(MN^2)$ nonzeroes. Thus, communication C and reduction R can be seen as the MemXCT parallelization overhead. Both C and R have $O(MN\sqrt{P})$ nonzeroes, where P is the number of processes. It is because when P quadruples, total communication footprint on sinogram domain doubles. For communications, there is an extra $O(\sqrt{P})$ term comes due to the handshake overhead between processes. On the other hand, compute-centric approach involves race conditions and hence it duplicates the tomogram domain across all processes [10]. In that case, duplicated domains has to be reduced through `MPI_Allreduce` at the end of each backprojection, yielding $O(N^2 \log P)$ parallelization overhead. Resulting computational complexities are shown in Table 1.

Table 1: Computational Complexities

	Sequential	Trace	MemXCT
Memory	$MN + N^2$	$MN/P + N^2$	$MN^2/P + MN/\sqrt{P}$
Comput.	MN^2	MN^2/P	$MN^2/P + MN/\sqrt{P}$
Comm.	N/A	$N^2 \log P$	$MN/\sqrt{P} + \sqrt{P}$

M: # of projections, N: # of channels P: # of processes

3.5 Other Details

MemXCT requires an extra preprocessing step for avoiding redundant and inefficient computations as opposed to a compute-centric approach. The preprocessing involve: (1) Hilbert ordering and domain decomposition, (2) ray tracing for constructing forward projection matrix, (3) sparse transposition for constructing backprojection matrix, and (4) row partitioning and building corresponding buffer data structures. Preprocessing is MPI+OpenMP parallel and is performed on CPU.

3.5.1 Preserving Data Locality. MemXCT preserves data locality through all matrix manipulations, which is essential for our performance optimizations. For example, constructing backprojection matrix requires a sparse matrix transposition of the forward projection matrix. MemXCT performs this through a scan-based matrix

transposition [22] which preserves data ordering, rather than an atomic-based transposition which randomizes data ordering.

3.5.2 Iterative Solution. There is a plethora of solution schemes for iterative XCT. To mention a few, recent work implements SIRT [10], SGD [16], and ICD [23] iterations. Any of them can be implemented for our proposed memory-centric approach in a plug-and-play manner with minor modifications. Nevertheless, MemXCT implements CG iterations [24], which has a faster convergence rate than any of them (at a higher per-iteration cost). It is simply because: (1) a full gradient is found rather than a partial gradient, (2) optimal step size is found analytically via an additional forward projection, and (3) visiting redundant directions is prevented via three-term recursion. We use a heuristic early termination of iterations to prevent *overfitting*, which is practically considered as a regularization method. Convergence results are shown in Section 4.1.2.

4 NUMERICAL RESULTS

All experiments in this paper are performed on ALCF Theta [25], NCSA Blue Waters [26], ALCF Cooley [27], an IBM Minsky node [28], and an Nvidia DGX-1 node [29]. Table 2 characterizes these machines. Mem. B/W is theoretical on-chip memory bandwidth; we assume that ECC (error correcting code) degrades theoretical bandwidth of K20X and K80 by 15% [30]. Link describes interface between host and device.

Table 2: Key Features of Machines Used for Experiments

Machine	Nodes	Accel.	On-Chip Mem.	Mem. B/W	Mem.	Link
Theta	4392	KNL	16 GB MCDRAM	400 GB/s	192 GB	90 GB/s
Blue W.	4228	K20X	6 GB DDR5	121.5 GB/s	32 GB	PCIe
Cooley	126	2×K80	12 GB DDR5	204 GB/s	384 GB	PCIe
Minsky	1	4×P100	16 GB HBM2	720 GB/s	128 GB	NVLink
DGX-1	1	8×V100	16 GB HBM2	900 GB/s	512 GB	NVLink

We used six datasets, as shown in Table 3, in order to evaluate application performance. The first four are artificially created for performance evaluation and the latter two are from real synchrotron experiments at APS. Measurement sinograms are given for a single slice. The artificial datasets follow parallel raster scan geometry just as the real datasets. The irregular and regular data footprints, defined in Section 3.1.1, are given for all datasets in Table 3. The first/second entry for memory footprints are accessed in forward/backprojection, respectively. RDS1 involves a shale sample [31] and RDS2 involves a brain sample. RDS1 is available open source [32], and RDS2 is proprietary.

4.1 Evaluating Overall Performance

4.1.1 Comparison with Compute-Centric Approach. We compare our memory-centric MemXCT with compute-centric Trace [10], an open-source high-performance implementation that employs SIRT iterations. To enable a one-to-one comparison, we implement SIRT, and run 45 iterations with both codes on a single KNL for the ADS2 and RDS1 datasets (see Table 3). Table 4 reports solution times and corresponding speedups. In the best case, where MemXCT memory footprint fits within MCDRAM, it performs each iteration 49.2× faster than Trace. In the worst case, where MemXCT is bounded by slow DRAM bandwidth due to its large memory footprint, it still performs each iteration 6.86× faster. Trace memory footprint fits within MCDRAM in both cases, but it has to perform redundant and inefficient computation.

Table 3: Dataset Details and Memory Footprints

Name	Sinogram (M×N)	Sample	Irregular Data	Regular Data
ADS1	360×256	Artificial	256/360 KB	215/215 MB
ADS2	750×512	Artificial	1.0/1.5 MB	1.8/1.8 GB
ADS3	1500×1024	Artificial	4.0/6.0 MB	14/14 GB
ADS4	2400×2048	Artificial	16/19 MB	90/90 GB
RDS1	1501×2048	Shale Rock	16/12 MB	56/56 GB
RDS2	4501×11 283	Mouse Brain	500/198 MB	5.1/5.1 TB

Table 4: Comparison with Compute-Centric Approach

		Preproc.	Reconst.	Per-Iter.	Speedup
ADS2	**Trace**	N/A	26.05 s	579 ms	1×
	MemXCT	4.00 s	0.53 s	11.8 ms	49.2×
RDS1	**Trace**	N/A	425.3 s	9.45 s	1×
	MemXCT	146.3 s	62.00 s	1.37 s	6.86×

Table 4 reports preprocessing overhead of MemXCT. Although the preprocessing step appears to be a significant fraction of overall time, when it comes to many-slice reconstruction, the preprocessing cost is paid only once for the first slice. It is then reused for all the remaining slices, as shown in Table 5.

4.1.2 Iterative Convergence. MemXCT uses CG for iterative solutions, as discussed in Section 3.5.2, as opposed to SIRT used by Trace. Fig. 8(a) presents convergence properties by comparing L-curves of CG and SIRT up to 500 iterations: horizontal and vertical axes represents residual and reconstruction norms, respectively (see Section 2.2 for iterative formulation). As L-curve suggests, CG solution experiences overfitting soon after 30 iterations where the image does not further improve, but instead starts to be polluted by noise. Therefore we terminate CG solution after 30 iterations. On the other hand, SIRT does not converge even with 500 iterations. Fig. 8(b) shows a single-slice reconstruction from RDS1, and (c) and (d) compare image details after 30 CG iterations with MemXCT and 45 SIRT iterations with Trace, respectively.

4.1.3 Machine-Specific Considerations. We reconstruct RDS1 on small number of Theta, Cooley, and Blue Waters nodes. Machines were described briefly earlier. Unfortunately, reconstruction does not fit within DGX-1 or on less than eight nodes of Cooley or 32 nodes of Blue Waters, due to limited memory of their respective GPUs and the large memory complexity of MemXCT. Nevertheless, reconstruction fits well into a single Theta node thanks to its large DRAM capacity.

Table 5: Reconstruction on Various Nodes-Machines

Nodes-Machine	Preproc.	Speed.	Recon.	Speed.	All Slices
1-Theta (1 KNL)	139 s	1×	63.3 s	1×	1.44 d*
8-Theta (8 KNL)	16.5 s	8.42×	3.33 s	19.0×	1.89 h
8-Cooley (16 K80)	25.5 s	5.45×	2.89 s	21.9×	1.64 h
32-Blue W. (32 K20X)	14.6 s	9.52×	1.82 s	34.8×	62.1 m
32-Theta (32 KNL)	4.54 s	30.6×	1.37 s	46.2×	46.8 m
32-Cooley (64 K80)	6.31 s	22.0×	1.22 s	51.9×	41.6 m

* Calculated based on single-slice execution time.

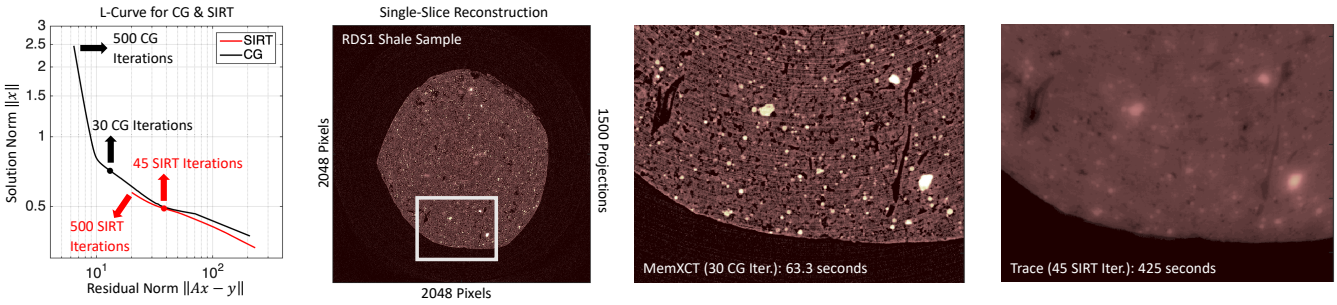

Figure 8: For RDS1: (a) L-curves for CG and SIRT iterations. (b) Single-slice reconstruction with (c) CG and (d) SIRT iterations.

Table 5 reports RDS1 preprocessing and reconstruction times on various numbers of nodes-machines. Results show that 32 nodes of all machines enjoy comparable time to solution: reconstruction of all slices reduces from 1.44 days down to about or less than an hour. Preprocessing time also scales well and, in fact, it is insignificant compared to reconstruction time of all slices. Table 5 also shows the super-linear speedup property of MemXCT on KNLs: 8-Theta is 19× faster than 1-Theta. This is a result of shrinking per-node memory footprint and extra memory bandwidth gain when it fits within 16 GB MCDRAM capacity. The super-linear speedup also demonstrates effective domain decomposition and reduced communications described in Section 3.

4.2 Performance Optimizations

This subsection presents single-device performance gains due to the optimizations described in Section 3. Fig. 9 shows forward and backprojection performance metrics for ADS1 through ADS4 on KNL and GPU. ADS3 and ADS4 are too large to fit in a single GPU (see Table 3). Hilbert ordering and input buffering are applied to the baseline in order because the first optimization enables the second. Since performance is dependent on both dataset and device, we tune all results (including the baseline) independently for maximum GFLOPS as described in Section 4.2.4.

Since there are two FLOPs per non-zero element in the projection matrix (one multiplication and one addition per FMA), GFLOPS metric is calculated as $2N_{\text{nz}}/t$, where t is the time measured for a single forward/backprojection. Similarly, average memory bandwidth utilization is calculated for regular data only as $N_{\text{nz}} \times B_{\text{reg}}/t$, where B_{reg} is regular data (in bytes) read from memory per FMA, respectively. L2 miss rates are measured by Intel VTune profiler. The performance metrics are aggregated over many iterations.

4.2.1 The Baseline GFLOPS on KNL drops as dataset size increases, regardless of whether or not it fits into high-bandwidth MCDRAM. The reason for this behaviour is that the baseline is bounded by memory latency rather that memory bandwidth, due to the high L2 miss rates of irregular access. As a result, larger datasets suffer more performance degradation due to their higher L2 miss rates, as seen in Fig. 9(b). In contrast, GPU performance improves slightly with larger datasets, as more parallelism hides the latency.

4.2.2 Pseudo-Hilbert Ordering provides data locality and reduces the L2 miss rates for all datasets as seen in Fig. 9(b). ADS1 does not benefit from Hilbert ordering as much as other datasets due

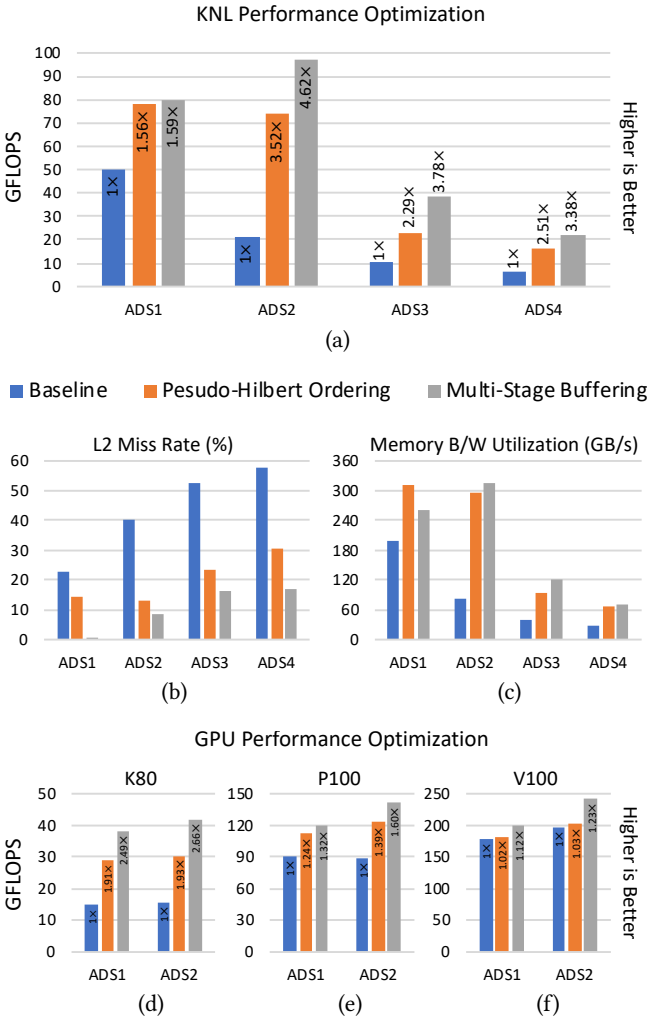

Figure 9: KNL performance: (a) GFLOPS, (b) L2 miss rate, and (c) memory bandwidth utilization. GPU GFLOPS performance: (d) K80, (e) P100, and (f) V100.

to its small size. Nevertheless, as opposed to the baseline, the performance of Hilbert ordering is bounded by memory bandwidth consumed by regular data rather than memory latency. On KNL, regular data smaller than 16 GBs (ADS1 and ADS2) fits completely into MCDRAM and performs better whereas large regular data

(ADS3 and ADS4) does not and is bounded by low DRAM memory bandwidth. On GPUs, the improvement is less noticeable with larger L2 cache because irregular data is better cached. As a result, pseudo-Hilbert ordering speeds up K80, P100, and V100 baselines about 1.93×, 1.39×, and 1.03×, respectively.

This analysis shows that ADS1 and ADS2 use at least 78% and 74% of the theoretical MCDRAM bandwidth, respectively. The ADS3 regular data (28 GB) is larger than MCDRAM (16 GB) where it is cached partially. We cannot say the same for ADS4 because it has too large regular data to be significantly cached at MCRAM. As a result, we deduce that ADS4 uses at least 73% of the theoretical DRAM bandwidth. These utilizations agree well with STREAM benchmarks in previous work [33]. Similarly, K80, P100 and V100 use 78%, 69%, and 92% of their theoretical HBM bandwidths, respectively. These numbers agree well with benchmark measurements in previous work [34].

4.2.3 Multi-Stage Buffering as we described in Section 3.3, comes with an staging overhead. On KNL, the gain amortizes the overhead when dataset is large, as seen in ADS2 and up. ADS1 is not large enough to show any further performance improvement with input buffering. The bandwidth utilization for input buffering in Fig. 9(c) and Fig. 9(d)–(f) is adjusted with respect to additional memory-to-buffer mappings as well as reduced bytes for buffer-address indices.

It is worth to comment that reduced L2 miss rate saves significant memory bandwidth on K80 since its utilization due to regular data increases to at least 67% of theoretical peak as seen in Fig. 9(d). The respective utilizations on P100 and V100 drop slightly, if not remain the same, because their L2 miss rates are already low and bandwidth utilizations are already high thanks to Hilbert ordering and their large L2 capacity. As a result, we can deduce that GFLOPS improvements on P100 and V100 are solely provided by the bandwidth saving due to reduced number of bytes needed for addressing shared-memory.

4.2.4 Tuning the baseline and Hilbert ordering are relatively simple on both KNL and GPU architectures. We perform an exhaustive search and find out that blocks size of 128 scheduled dynamically among 128 threads (2 SMT/core) provides good single KNL performance for ADS1 through ADS4. Similarly, block size of 32 or 64 provides good single GPU performance.

For input buffering optimization, parameters should be re-tuned along with buffer size for effective SMT utilization on KNL. Fig. 10(a)–(c) shows GFLOPS heat maps for ADS2 with various block and buffer sizes, and different numbers of SMT/core. Results show that input buffering can use more SMTs per core because it finds opportunity to overlap buffer stagings and accesses among SMTs. The following factors need to be considered when using SMTs in this way: A larger block size increases the per-block memory footprint and thus data reuse from the buffer, but also increases the number of buffer stagings per block, which comes with an overhead. In that case, it is better to increase buffer size for limiting the number of stagings. However, too few stagings decreases the opportunity to overlap buffer stagings and accesses among SMTs. Also, too large a buffer size can result in leaks to L2.

On KNL, peak GFLOPS performance for ADS1 and ADS2 is achieved with 4 SMT/core and buffer size of 8 KB. Tuning for ADS3 and ADS4 is cumbersome since they are severely limited by DRAM

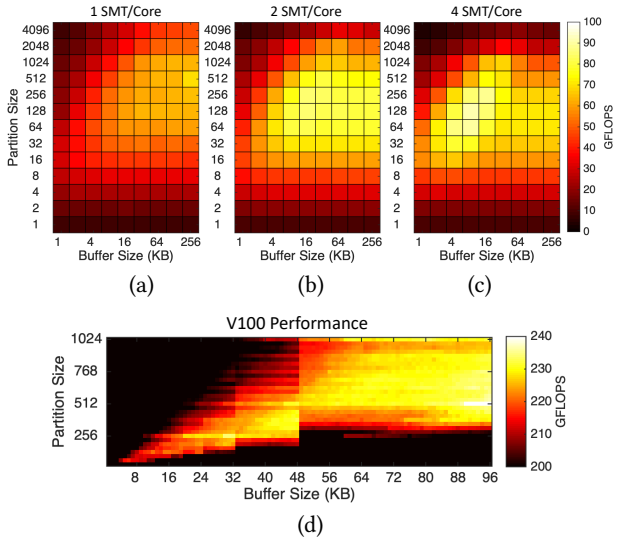

Figure 10: Tuning input buffer and block size for (a)-(c) various SMT/core on KNL and (d) for V100.

bandwidth and SMT effect is not that apparent. Therefore we use 2 SMT/core, and buffer size of 16 KB for them. Block size of 128 provides good performance for all datasets.

On GPUs block size of 512 or 1024 and buffer size of 48 KB or 96 KB provides good single-GPU performance for all GPUs. It is worth to note that addressable shared-memory size is limited to 48 KB on P100 and K80 and therefore tuning space is half the size. We follow a similar tuning strategy for GPUs as explained for KNL, as seen in Fig. 10(d).

4.2.5 Comparison with Existing Libraries. We compare MemXCT forward/backprojection kernel performance with SpMV functions of existing libraries. We use Intel MKL library on KNL, and Nvidia cuSPARSE library on GPU. Table 6 reports speedups of our MemXCT baseline implementation and optimizations compared to CSR (on KNL) and column-major ELL (on GPU) SpMV of respective libraries. Evidently, MemXCT baseline implementation is faster than the existing libraries, with the exception for K80. This is due to K80 small L2 cache compared to P100 and V100 GPUs. On P100 and V100, our baseline is 1.39× and 1.79× faster since cuSPARSE pads CSR matrix with −1 [35] which requires extra branches whereas we pad with 0 and perform redundant multiplication with 0 to avoid thread divergence on GPUs. Also, cuSPARSE pads CSR on a matrix level whereas we pad on a thread-block (partition) level, which saves some of the redundant operations. As a result, we outperform existing libraries thanks to our application-specific SpMV optimizations. Nevertheless, both MKL and cuSPARSE are optimized for general SpMV whereas our kernels are optimized for XCT application.

Table 6: Comparison with MKL and cuSPARSE for ADS2

	KNL	K80	P100	V100
MKL/cuSPARSE	1×	1×	1×	1×
MemXCT Baseline	1.42×	0.52×	1.39×	1.79×
Pseudo-Hilbert Ordering	4.99×	1.13×	1.93×	1.84×
Multi-Stage Buffering	6.55×	1.56×	2.23×	2.11×

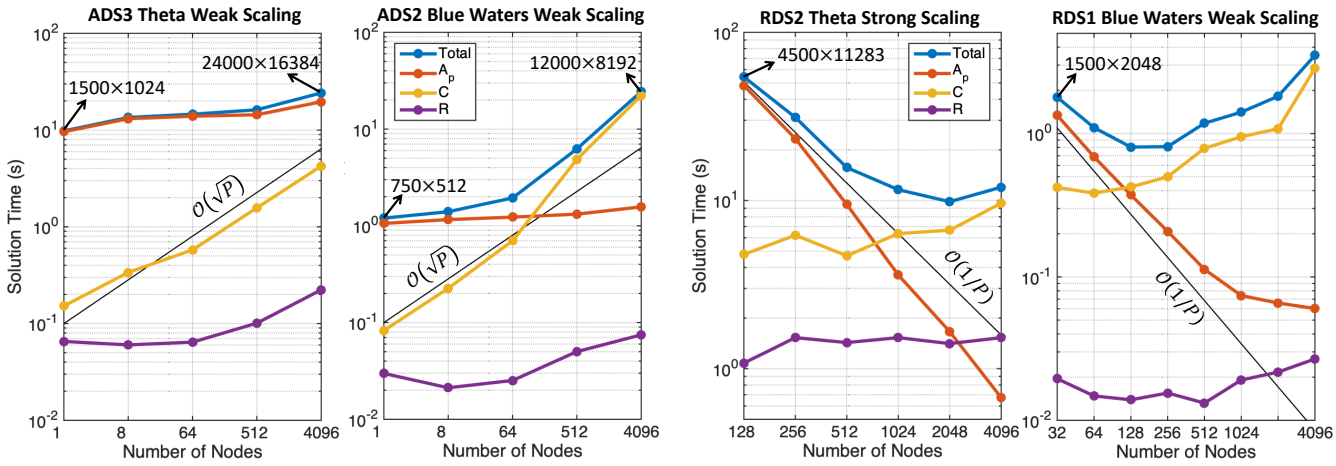

Figure 11: Weak scaling: (a) ADS3/Theta, (b) ADS2/Blue Waters. Strong scaling: (c) RDS2/Theta, (d) ADS3/Blue Waters.

4.3 Scalability

For presenting scaling of MemXCT on a large number of nodes, we perform weak and strong scaling experiments on Theta and Blue Waters. For each experiment, we perform a full solution with 30 CG iterations, and report total reconstruction times as well as A_p, C, and R kernel times as they are defined in Section 3.4. Kernel times involve both forward and backprojection. In order to accurately measure each individual kernel, we placed the necessary artificial barriers, though they slightly increase total execution time. Among the kernels, C involves inter-node communication therefore we include its host-device communication times. For demonstrating scaling, $O(\sqrt{P})$ and $O(1/P)$ curves are included in Fig. 11 as well as the sinogram data dimensions.

4.3.1 Weak Scaling. Each weak scaling experiment is performed by starting from the reconstruction of a root dataset, and then doubling the number of channels and projections in a sinogram on each step. As computational cost increases eight times per step, the number of nodes is increased eight times accordingly. Fig. 11(a)–(b) show weak scaling of root datasets ADS3 and ADS2 on Theta and Blue Waters, respectively. Both experiments exhibits good weak scaling except communication operations C because of its $O(\sqrt{P})$ complexity, as explained in Table 1. On Blue Waters, weak scaling is bounded by communication on 512 nodes and up; on Theta, communication is not the bounding component, but rather A_p. The difference is due to architectural differences across machines.

4.3.2 Strong Scaling. For strong scaling, we reconstruct RDS2 (brain) and RDS1 (shale) samples on Theta and Blue Waters respectively. We increase numbers of nodes and the dataset sizes are fixed. The minimum number of nodes needed for RDS2 and RDS1 to fit well within their respective systems is 128 and 32 nodes respectively. Reconstructions are scaled up to 4096 nodes of each system. Fig. 11(c)–(d) show strong scaling results. Theta exhibits good scaling up to 2048 nodes where as Blue Waters scales up to 128 nodes. The main reason (apart from the difference in network bandwidth and topology) is that the RDS2 solution is ~91× more costly than the RDS1, and therefore the former scales better than the latter. Also, A_p kernel exhibits super-linear speedup, demonstrating efficient domain partitioning and high-bandwidth memory utilization, as discussed in Section 4.1.3. RDS2 reconstruction results are in Fig. 1. In contrast, A_p performance saturates on 1024 nodes of Blue Waters (and up) since smaller per-node computation penalizes GPU performance.

Table 7: Comparison of Theta and Blue Waters

	RDS1	RDS2	12 000×8192
BW	805 ms (128 K20X)	74 s (4096 K20X)	24.4 s (4096 K20X)
Theta	474 ms (128 KNL)	10 s (2048 KNL)	3.25 s (4096 KNL)

4.3.3 Comparison of Theta and Blue Waters Systems. For cross-comparing Theta and Blue Waters systems, we pick the fastest reconstructions of both RDS datasets and their corresponding week scaling versions when ran on 4096 nodes. Results are shown in Table 7. RDS1 (shale) sample reconstruction runs fastest on 128 nodes on both Theta and Blue Waters. In this case, Theta is about 1.7× faster than Blue Waters. RDS2 (brain) sample reconstruction runs fastest on 2048 nodes of Theta and requires at least 4096 nodes to fits well into Blue Waters. In this case, Theta is 7.4× faster than Blue Waters. Lastly, Theta is about 7.5× faster on reconstructing the 12 000×8192 dataset.

5 RELATED WORK AND DISCUSSION

Iterative reconstruction algorithms provide superior image quality considering analytical approaches, however their usage has so far been limited with their computational demands [36–38].

The multicore parallelization of iterative reconstruction algorithms has long been studied [39–42]. However, most prior approaches provide limited scalability and only consider optimizations in the object domain. Further, they suffer from redundant computations when dealing with large datasets.

With the emergence of high-throughput many-core architectures and state-of-the-art reconstruction algorithms, the applicability of iterative techniques has become more feasible [12, 23, 43, 44]. Especially in medical imaging, iterative reconstruction approaches

have been extensively used to provide high-quality 3D reconstructions [45–47] in order to limit patient radiation exposure [3, 48]. Most of these approaches rely on GPUs to meet computational requirements of these algorithms [49–51]. Although GPUs can provide sufficient computational throughput, their limited memory can accommodate only small- to medium-scale datasets. For large-scale dataset host-device communication is known to be dominating overhead in GPUs [52]. Further, most of these approaches rely on on-the-fly computation of ray tracing, i.e. CompCXT type of data access pattern, which shows suboptimal performance compared to MemCXT.

Li et al. developed cuMBIR [16], a framework for model-based iterative reconstruction on GPUs. They implement two solvers ICD and SGD, and propose several optimizations including unified thread mapping. Their method is also based on on-the-fly computation of A matrix, and performs redundant computations. Further, their solution focuses on small to medium scale datasets that can fit into single node GPUs. In comparison, MemCXT is optimized for very large tomographic datasets that require not only single node performance but also efficient large-scale execution.

Wang et al. [43] use KNL many-core capabilities to reconstruct 1024^2 tomograms (or 1024^3 3D volumes). They introduce the *non-uniform parallel super-voxel* to exploit sinusoidal bands in sinograms and optimize data access patterns. Our approach shares similarities with their work, but we consider parallelization and performance optimizations in both the tomogram and sinogram domains. Moreover, we further improve data access patterns and communication using two-level pseudo Hilbert ordering and multi-stage input buffering on both domains. Our evaluation also extends to extremely large objects ($11\,293^2$ rather than 1024^2), showing the usability of our approach for very large datasets.

First space-filling curve example was presented by Peano in 1890 [53]. Later, space-filling curves have been successfully applied to improve performance of many applications [54–60]. In [61], Mellor-Crummey et al. showed that data and computation reordering based on space-filling curves can improve the performance of irregular applications significantly. Moon et al. analyzed clustering properties of Hilbert space-filling curve [62] on different query shapes [63]. Reissmann et al. showed the effects of Hilbert and Morton curves[64] on energy and locality [60]. In our work, we applied two-level pseudo Hilbert ordering to optimize the performance of tomographic reconstruction data access patterns and inter-process communication.

6 CONCLUSION

We have described a memory-centric approach to XCT reconstruction. Our implementation, MemXCT, performs projection and back-projection operations as explicit SpMVs with no on-the-fly (redundant) computations or race conditions; these features allow it to achieve up to 50× speedup over a high-performance compute-centric approach. Although MemXCT has high memory complexity, its per-node memory footprint decreases linearly with increasing number of compute resources, which favors large-scale resources. For solving the performance bottlenecks in traditional iterative reconstruction approaches, we propose and implement two-level pseudo-Hilbert ordering and multi-stage input buffering techniques in MemXCT. We compare the performance of our computational kernels on KNL and several generations of GPUs, and demonstrate scaling up to 4096 nodes using ALCF Theta (KNL) and NCSA Blue Waters (GPU) systems. We show that MemXCT can reconstruct a large-scale (11Kx11K) mouse brain tomogram in ~10 seconds using 4096 KNL nodes: the largest iterative reconstruction ever achieved in near-real time.

ACKNOWLEDGMENTS

This material was partially supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences, under Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Argonne Leadership Computing Facility, which is a U.S. Department of Energy, Office of Science User Facility. We thank Vincent De Andrade from Advanced Photon Source for sharing the mouse brain dataset. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This research is based in part upon work supported by: The Center for Applications Driving Architectures (ADA), a JUMP Center co-sponsored by SRC and DARPA. This material is based in part upon work supported by the XPACC Center for Exascale Simulation of Plasma-Coupled Combustion and Department of Energy, under Award Number DE-NA0002374.

REFERENCES

[1] "APS Science 2018," Tech. Rep. ANL-18/40, ISSN 1931-5007, Advanced Photon Source, Argonne National Laboratory, January 2019.
[2] E. L. Dyer, W. G. Roncal, J. A. Prasad, H. L. Fernandes, D. Gürsoy, V. De Andrade, K. Fezzaa, X. Xiao, J. T. Vogelstein, C. Jacobsen, *et al.*, "Quantifying mesoscale neuroanatomy using x-ray microtomography," *eNeuro*, pp. ENEURO–0195, 2017.
[3] E. Y. Sidky, C.-M. Kao, and X. Pan, "Accurate image reconstruction from few-views and limited-angle data in divergent-beam ct," *Journal of X-ray Science and Technology*, vol. 14, no. 2, pp. 119–139, 2006.
[4] X. Han, J. Bian, D. R. Eaker, T. L. Kline, E. Y. Sidky, E. L. Ritman, and X. Pan, "Algorithm-enabled low-dose micro-CT imaging," *IEEE Transactions on Medical Imaging*, vol. 30, pp. 606–620, March 2011.
[5] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pp. 945–948, IEEE, 2013.
[6] K. A. Mohan, S. Venkatakrishnan, L. F. Drummy, J. Simmons, D. Y. Parkinson, and C. A. Bouman, "Model-based iterative reconstruction for synchrotron x-ray tomography," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6909–6913, IEEE, 2014.
[7] D. J. Duke, A. B. Swantek, N. M. Sovis, F. Z. Tilocco, C. F. Powell, A. L. Kastengren, D. Gürsoy, and T. Biçer, "Time-resolved x-ray tomography of gasoline direct injection sprays," *SAE International Journal of Engines*, vol. 9, no. 1, pp. 143–153, 2016.
[8] D. Gürsoy, T. Biçer, A. Lanzirotti, M. G. Newville, and F. D. Carlo, "Hyperspectral image reconstruction for x-ray fluorescence tomography," *Optics Express*, vol. 23, pp. 9014–9023, Apr 2015.
[9] D. Gürsoy, T. Biçer, J. D. Almer, R. Kettimuthu, S. R. Stock, and F. De Carlo, "Maximum a posteriori estimation of crystallographic phases in x-ray diffraction tomography," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2043, p. 20140392, 2015.
[10] T. Bicer, D. Gürsoy, V. D. Andrade, R. Kettimuthu, W. Scullin, F. D. Carlo, and I. T. Foster, "Trace: A high-throughput tomographic reconstruction engine for large-scale datasets," *Advanced Structural and Chemical Imaging*, vol. 3, p. 6, Jan 2017.
[11] T. Bicer, D. Gursoy, R. Kettimuthu, F. De Carlo, G. Agrawal, and I. T. Foster, "Rapid tomographic image reconstruction via large-scale parallelization," in *Euro-Par*

2015: Parallel Processing, pp. 289–302, Springer, 2015.
[12] W. van Aarle, W. J. Palenstijn, J. De Beenhouwer, T. Altantzis, S. Bals, K. J. Batenburg, and J. Sijbers, "The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography," *Ultramicroscopy*, vol. 157, pp. 35–47, 2015.
[13] D. Gürsoy, F. De Carlo, X. Xiao, and C. Jacobsen, "TomoPy: A framework for the analysis of synchrotron tomographic data," *Journal of Synchrotron Radiation*, vol. 21, pp. 1188–1193, Sept. 2014.
[14] A. Beer, "Bestimmung der absorption des rothen lichts in farbigen flussigkeiten," *Ann. Physik*, vol. 162, pp. 78–88, 1852.
[15] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional ct array," *Medical Physics*, vol. 12, no. 2, pp. 252–255, 1985.
[16] X. Li, Y. Liang, W. Zhang, T. Liu, H. Li, G. Luo, and M. Jiang, "cuMBIR: An efficient framework for low-dose x-ray CT image reconstruction on GPUs," in *International Conference on Supercomputing*, pp. 184–194, ACM, 2018.
[17] S. G. De Gonzalo, S. Hammond, S. Huang, O. Mutlu, J. Gómez-Luna, and W.-m. Hwu, "Automatic generation of warp-level primitives and atomic instructions for fast and portable parallel reduction on GPUs,"
[18] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco, "Dynamic octree load balancing using space-filling curves," 2003.
[19] M. Parashar and J. C. Browne, "On partitioning dynamic adaptive grid hierarchies," in *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*, vol. 1, pp. 604–613, IEEE, 1996.
[20] J. Zhang, S.-i. Kamata, and Y. Ueshige, "A pseudo-hilbert scan algorithm for arbitrarily-sized rectangle region," in *International Workshop on Intelligent Computing in Pattern Analysis and Synthesis*, pp. 290–299, Springer, 2006.
[21] Y. Wang, F. De Carlo, D. C. Mancini, I. McNulty, B. Tieman, J. Bresnahan, I. Foster, J. Insley, P. Lane, G. von Laszewski, *et al.*, "A high-throughput x-ray microtomography system at the Advanced Photon Source," *Review of Scientific Instruments*, vol. 72, no. 4, pp. 2062–2068, 2001.
[22] H. Wang, W. Liu, K. Hou, and W.-c. Feng, "Parallel transposition of sparse data structures," in *International Conference on Supercomputing*, p. 33, ACM, 2016.
[23] X. Wang, A. Sabne, S. Kisner, A. Raghunathan, C. Bouman, and S. Midkiff, "High performance model based image reconstruction," in *ACM SIGPLAN Notices*, vol. 51, p. 2, ACM, 2016.
[24] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*, vol. 43. Siam, 1994.
[25] ALCF, "Theta | Argonne Leadership Computing Facility." https://www.alcf.anl.gov/theta. Accessed: 2019-01-16.
[26] NCSA, "Blue Waters user portal | home." https://bluewaters.ncsa.illinois.edu/. Accessed: 2019-01-16.
[27] ALCF, "Cooley | Argonne Leadership Computing Facility." https://www.alcf.anl.gov/user-guides/cooley. Accessed: 2019-01-16.
[28] IBM, "Ibm power system s822lc for high performance computing - overview." https://www.ibm.com/us-en/marketplace/high-performance-computing. Accessed: 2019-03-20.
[29] Nvidia, "Deep learning server for ai research | nvidia cgx-1." https://www.nvidia.com/en-us/data-center/dgx-1/. Accessed: 2019-03-20.
[30] OLCF, "Titan User Guide - Oak Ridge Leadership Computing Facility." https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#nvidia-k20x-gpus. Accessed: 2019-01-16.
[31] W. Kanitpanyacharoen, D. Y. Parkinson, F. De Carlo, F. Marone, M. Stampanoni, R. Mokso, A. MacDowell, and H.-R. Wenk, "A comparative study of x-ray tomographic microscopy on shales at different synchrotron facilities: Als, aps and sls," *Journal of synchrotron radiation*, vol. 20, no. 1, pp. 172–180, 2013.
[32] F. De Carlo, D. Gürsoy, D. J. Ching, K. J. Batenburg, W. Ludwig, L. Mancini, F. Marone, R. Mokso, D. M. Pelt, J. Sijbers, *et al.*, "Tomobank: a tomographic data repository for computational x-ray science," *Measurement Science and Technology*, vol. 29, no. 3, p. 034004, 2018.
[33] I. B. Peng, R. Gioiosa, G. Kestor, P. Cicotti, E. Laure, and S. Markidis, "Exploring the performance benefit of hybrid memory system on hpc environments," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 683–692, IEEE, 2017.
[34] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU architecture via microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.
[35] Nvidia, "CUSPARSE Library." https://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUSPARSE_Library.pdf. Accessed: 2019-01-16.
[36] M. Beister, D. Kolditz, and W. A. Kalender, "Iterative reconstruction methods in x-ray CT," *Physica Medica*, vol. 28, no. 2, pp. 94–108, 2012.
[37] J. Hsieh, B. Nett, Z. Yu, K. Sauer, J.-B. Thibault, and C. A. Bouman, "Recent advances in CT image reconstruction," *Current Radiology Reports*, vol. 1, no. 1, pp. 39–51, 2013.
[38] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Eurographics âĂŹ87*, pp. 3–10, 1987.
[39] J. Agulleiro and J.-J. Fernandez, "Fast tomographic reconstruction on multicore computers," *Bioinformatics*, vol. 27, no. 4, pp. 582–583, 2011.
[40] J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein, "Pushing the limits for medical image reconstruction on recent standard multicore processors," *International Journal of High Performance Computing Applications*, 2012.
[41] C. Johnson and A. Sofer, "A data-parallel algorithm for iterative tomographic image reconstruction," in *7th Symposium on the Frontiers of Massively Parallel Computation*, pp. 126–137, Feb 1999.
[42] M. Jones, R. Yao, and C. Bhole, "Hybrid MPI-OpenMP programming for parallel OSEM PET reconstruction," *Nuclear Science, IEEE Transactions on*, vol. 53, pp. 2752–2758, Oct. 2006.
[43] X. Wang, A. Sabne, P. Sakdhnagool, S. J. Kisner, C. A. Bouman, and S. P. Midkiff, "Massively parallel 3d image reconstruction," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 3, ACM, 2017.
[44] J. Qi and R. M. Leahy, "Iterative reconstruction techniques in emission computed tomography," *Physics in Medicine and Biology*, vol. 51, no. 15, p. R541, 2006.
[45] C.-Y. Chou, Y.-Y. Chuo, Y. Hung, and W. Wang, "A fast forward projection using multithreads for multirays on GPUs in medical image reconstruction," *Medical Physics*, vol. 38, no. 7, pp. 4052–4065, 2011.
[46] G. Pratx, G. Chinn, P. Olcott, and C. Levin, "Fast, accurate and shift-varying line projections for iterative reconstruction using the GPU," *Medical Imaging, IEEE Transactions on*, vol. 28, pp. 435–445, March 2009.
[47] D. Lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky, and A. W. Toga, "CUDA optimization strategies for compute-and memory-bound neuroimaging algorithms," *Computer Methods and Programs in Biomedicine*, vol. 106, no. 3, pp. 175–187, 2012.
[48] B. Jang, D. Kaeli, S. Do, and H. Pien, "Multi GPU implementation of iterative tomographic reconstruction algorithms," in *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*, pp. 185–188, IEEE, 2009.
[49] A. Sabne, X. Wang, S. J. Kisner, C. A. Bouman, A. Raghunathan, and S. P. Midkiff, "Model-based iterative CT image reconstruction on GPUs," in *22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 207–220, ACM, 2017.
[50] S. S. Stone, J. P. Haldar, S. C. Tsao, W.-m. Hwu, B. P. Sutton, Z.-P. Liang, *et al.*, "Accelerating advanced MRI reconstructions on GPUs," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1307–1318, 2008.
[51] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *Nuclear Science, IEEE Transactions on*, vol. 52, no. 3, pp. 654–663, 2005.
[52] T. B. Jablin, P. Prabhu, J. A. Jablin, N. P. Johnson, S. R. Beard, and D. I. August, "Automatic CPU-GPU communication management and optimization," in *ACM SIGPLAN Notices*, vol. 46, pp. 142–151, ACM, 2011.
[53] G. Peano, "Sur une courbe, qui remplit toute une aire plane," *Mathematische Annalen*, vol. 36, no. 1, pp. 157–160, 1890.
[54] M. Bader, *Space-filling curves: an introduction with applications in scientific computing*, vol. 9. Springer Science & Business Media, 2012.
[55] F. Günther, M. Mehl, M. Pögl, and C. Zenger, "A cache-aware algorithm for pdes on hierarchical data structures based on space-filling curves," *SIAM Journal on Scientific Computing*, vol. 28, no. 5, pp. 1634–1650, 2006.
[56] H. Sagan, *Space-filling curves*. Springer Science & Business Media, 2012.
[57] M. Bader and C. Zenger, "Cache oblivious matrix multiplication using an element ordering based on a peano curve," *Linear Algebra and Its Applications*, vol. 417, no. 2-3, pp. 301–313, 2006.
[58] I. H. Witten and R. M. Neal, "Using peano curves for bilevel display of continuous-tone images," *IEEE Computer Graphics and applications*, no. 3, pp. 47–52, 1982.
[59] J. K. Lawder and P. J. H. King, "Querying multi-dimensional data indexed using the hilbert space-filling curve," *ACM Sigmod Record*, vol. 30, no. 1, pp. 19–24, 2001.
[60] N. Reissman, J. C. Meyer, and M. Jahre, "A study of energy and locality effects using space-filling curves," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pp. 815–822, IEEE, 2014.
[61] J. Mellor-Crummey, D. Whalley, and K. Kennedy, "Improving memory hierarchy performance for irregular applications using data and computation reorderings," *International Journal of Parallel Programming*, vol. 29, no. 3, pp. 217–247, 2001.
[62] D. Hilbert, *Über die stetige Abbildung einer Linie auf ein Flächenstück*, pp. 1–2. Berlin, Heidelberg: Springer Berlin Heidelberg, 1935.
[63] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," *IEEE Transactions on knowledge and data engineering*, vol. 13, no. 1, pp. 124–141, 2001.
[64] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," 1966.