Consensus Equilibrium Framework for Super-Resolution and Extreme-Scale CT Reconstruction

Xiao Wang
Harvard Medical School
Boston Children's Hospital

Venkatesh Sridhar
Purdue University

Zahra Ronaghi
NVIDIA Corporation

Rollin Thomas
Jack Deslippe
Dilworth Parkinson
Lawrence Berkeley Laboratory

Gregery T. Buzzard
Samuel P. Midkiff
Charles A. Bouman
Purdue University

Simon K. Warfield
Harvard Medical School
Boston Children's Hospital

ABSTRACT

Computed tomography (CT) image reconstruction is a crucial technique for many imaging applications. Among various reconstruction methods, Model-Based Iterative Reconstruction (MBIR) enables super-resolution with superior image quality. MBIR, however, has a high memory requirement that limits the achievable image resolution, and the parallelization for MBIR suffers from limited scalability. In this paper, we propose Asynchronous Consensus MBIR (AC-MBIR) that uses Consensus Equilibrium (CE) to provide a super-resolution algorithm with a small memory footprint, low communication overhead and a high scalability. Super-resolution experiments show that AC-MBIR has a 6.8 times smaller memory footprint and 16 times more scalability, compared with the state-of-the-art MBIR implementation, and maintains a 100% strong scaling efficiency at 146880 cores. In addition, AC-MBIR achieves an average bandwidth of 3.5 petabytes per second at 587520 cores.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; **Image processing**;

ACM Reference format:
Xiao Wang, Venkatesh Sridhar, Zahra Ronaghi, Rollin Thomas, Jack Deslippe, Dilworth Parkinson, Gregery T. Buzzard, Samuel P. Midkiff, Charles A. Bouman, and Simon K. Warfield. 2019. Consensus Equilibrium Framework for Super-Resolution and Extreme-Scale CT Reconstruction. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis, Denver, CO, USA, November 17–22, 2019 (SC '19),* 13 pages.
https://doi.org/10.1145/3295500.3356142

1 INTRODUCTION

A computed tomography (CT) system is an imaging modality to produce a 3D reconstruction of a scanned object. As one of the most popular imaging systems, CT is heavily used for medical imaging, with 72 million scans per year in the US alone, and is also widely used in security imaging [12], scientific imaging [18, 24, 25, 27, 28], and industrial imaging for inspecting manufacturing parts [19].

An important goal for CT imaging is super-resolution, namely to improve image resolution beyond the limit of detector resolution. CT super-resolution is not only used to detect early-stage cancers [13], but is also used to study microscopic structure, such as brain neurons and synthetic material micro-structures [8]. To accomplish super-resolution, interpolation methods produce a Filtered-back Projection (FBP) image through an inverse radon transform, and then interpolate the FBP image to super-resolution. Interpolation-based methods, however, fail to provide more image detail than the uninterpolated FBP images [17]. Example-based methods, such as deep learning and dictionary learning [11], are another popular choice for super-resolution. They extract image details from similar example images and apply these details to new images. In many cases, unfortunately, example-based methods require an unattainably large training database. More importantly in the presence of unusual image outliers such as metal, image details learned from example images are not always consistent with the object's taken X-ray measurements and example-based methods fail to truthfully reconstruct.

In contrast, Model-Based Iterative Reconstruction (MBIR) [23, 29] is a Bayesian based method that not only achieves good super-resolution quality consistent with X-ray measurements [9, 21], but also reduces the radiation doses by up to 80% for certain applications [14]. At a high level, MBIR has better image quality because it incorporates complex mathematical models to preserve X-ray acquisition information, while estimating image details and suppressing noise from image spatial property.

Despite of the benefits described above, the realistic but complex models for MBIR have a very large memory requirement to store a physics-related geometry system matrix, whose size grows linearly with image size. Any naive way to shrink or distribute the models among distributed nodes corrupts the CT physics-geometry system matrix. Therefore, the high memory requirement restricts the image resolution that MBIR can achieve, as such geometry system matrix information has to be broadcast to and stored on all nodes.

In addition to an excessive memory requirement, the parallel scalability of MBIR is another performance bottleneck. With its complex models, MBIR often has much longer reconstruction times and is computationally much more expensive than other methods

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

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356142

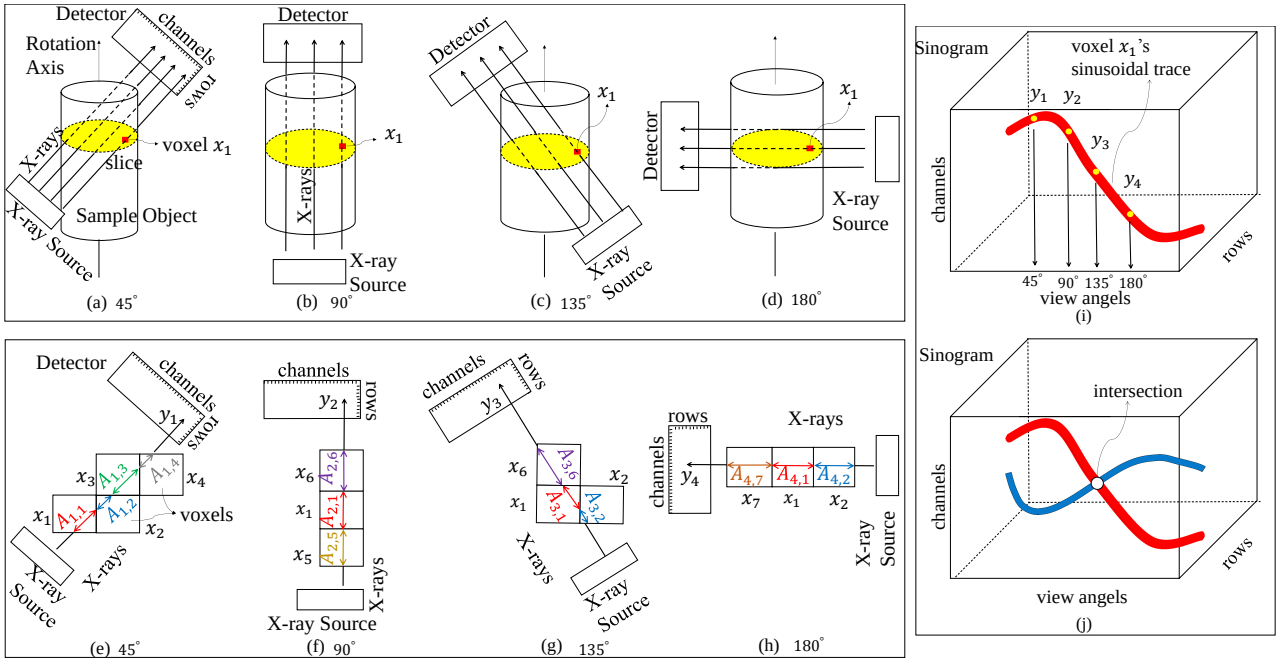


Figure 1: (a) A CT system at 45° view angle with a slice in yellow and a voxel x_1 in red. (b), (c) and (d) CT system rotates to 90°, 135° and 180° respectively. (e) An X-ray at 45° view angle passes through x_1 and its neighbor voxels and hits the detector with a measurement y_1. The intersection between the X-ray and each voxel is shown as a line segment with a different color for each. (f), (g) and (h) X-rays pass through x_1 at 90°, 135° and 180° respectively. (i) Shows the sinusoidal memory layout for x_1 in Sinogram. (j) Demonstrate an intersection between two voxel traces.

by several orders of magnitude. Therefore, four parallelization levels have been proposed to reduce computation times: (1) parallelizing SIMD operations for a voxel update [27], where a voxel is a three-dimensional pixel in the reconstruction volume; (2) parallelizing across voxels [6, 31]; (3) parallelizing across super-voxels (SVs), where an SV is a group of spatially-close voxels in the shape of a rectangular cuboid [26, 27]; and (4) parallelizing across slices [3, 7, 18], where a slice is defined as a cross-section of the scanned object with a fixed thickness. Unfortunately, these four levels of parallelization are insufficient to fully utilize a large supercomputer. With all four parallelization levels, the scalability for the state-of-the-art MBIR implementation is still limited to a small portion of a supercomputer [27].

This paper proposes the *Asynchronous Consensus MBIR* Algorithm (AC-MBIR) that significantly reduces MBIR's memory footprint and achieves better scalability. Without AC-MBIR, MBIR has a limited super-resolution and cannot scale to a large supercomputer. AC-MBIR allows each node to perform an individual reconstruction using a subset of measurements, and then uses a lossless *Consensus Equilibrium Method* (CE) to merge the individual reconstructions asynchronously into a consensus solution. But unlike the previous work on CE that has no memory footprint reduction or distributed node implementation [4, 22], AC-MBIR uses a new technique, *grouped-view partition*, to optimize memory footprint reduction. Furthermore, parallelism for AC-MBIR is not only achieved by dividing a reconstruction volume into slices and SVs, but also by dividing models and measurements into data subsets without corrupting their information, and performing parallel individual reconstructions with a low communication overhead.

In summary, this paper makes three contributions: (1) proposing a novel grouped-view partition technique to distribute the CT geometry model and measurements among nodes, and optimizing memory footprint reduction; (2) introducing the AC-MBIR algorithm that enables parallel asynchronous consensus updates for a high scalability on a supercomputer with a low communication overhead; and (3) applying the CE Method to AC-MBIR to ensure a convergence to the global optimum. Super-resolution experiments show that AC-MBIR has a 6.8 times smaller memory footprint and 16 times more scalability, compared with the state-of-the-art MBIR implementation [27], and maintains a 100% strong scaling efficiency at 146880 cores. In addition, AC-MBIR achieves an average bandwidth of 3.5 petabytes per second at 587520 cores.

2 BACKGROUND

2.1 CT System Setup

A CT system places the sample object in the center and the CT system has a gantry that rotates around the sample object with an x-ray source at one end of the gantry and an X-ray detector at the other end. At each view angle of a rotation, the X-ray source emits X-rays that pass through the sample object and project onto an X-ray detector, whose horizontal sensor units are channels and vertical sensor units are rows. Each received CT measurement on the X-ray detector corresponds to the integral of radio density of the sample

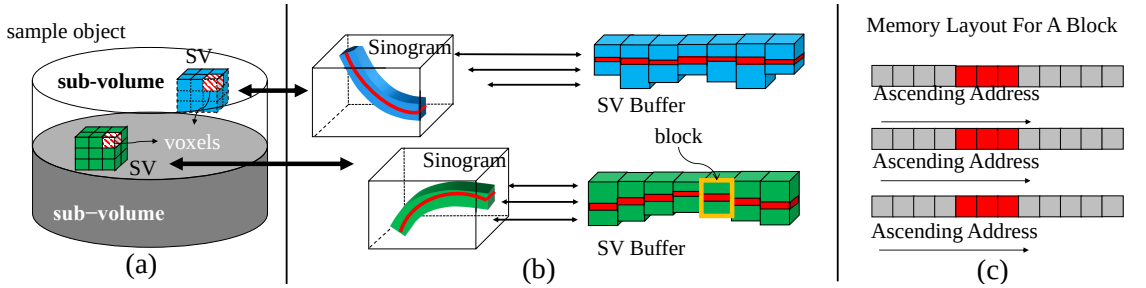


Figure 2: (a) Two nodes update two sub-volumes simultaneously and each core of a node updates a different SV in its assigned sub-volume. (b) Two private SV buffers for efficient data operations and each SV buffer consists of blocks of measurements. (c) Shows the memory layout for a block with completely regular access pattern.

object along the path of X-rays at the current view angle. Figure 1(a) shows the CT system setup at view angle of 45° with a single slice depicted in yellow and a voxel, x_1, in that slice depicted in red. In addition, detector channels and rows are represented as short dashes in the detector. Figure 1(e) shows an X-ray that passes through voxel x_1, and x_1's neighbor voxels, x_2, x_3 and x_4, at 45° view angle and hits the right side of the detector with a CT measurement, y_1. We define the length of intersection between the X-ray in Figure 1(e) and voxel x_2 as $A_{1,2}$, shown as a blue line segment. Similarly, the lengths of intersection for the X-ray in Figure 1(e) and voxels x_1, x_3 and x_4 are denoted as $A_{1,1}$, $A_{1,3}$ and $A_{1,4}$ respectively. Since y_1 is a line integral of all voxels' radio density on the path of the X-ray, y_1 can be written as $y_1 = A_{1,1}x_1 + A_{1,2}x_2 + A_{1,3}x_3 + A_{1,4}x_4 + e_1$, where e_1 is y_1's data error and represents the data corruption from scanner noise and detector artifacts.

Figure 1(b), (c) and (d) are the CT system setup rotated to view angles of 90°, 135° and 180° respectively, and Figure 1(f), (g) and (h) show measurements, y_2, y_3 and y_4 for the X-rays that pass through voxel x_1 at corresponding view angles. Note that the X-rays for y_2, y_3 and y_4 pass through a different set of voxels with different lengths of intersections at each view angle. For example, $A_{2,1}$ is longer than $A_{3,1}$ and $A_{1,1}$. In addition, X-rays hit different detector channels at different view angles. In this example, a right detector channel receives measurement y_1 at 45° and a center detector channel receives measurement y_4 at 180°. If we use e_2, e_3 and e_4 to represent the data errors for y_2, y_3 and y_4, then voxel x_1 can then be computed as the solution to the following linear equations:

$$y_1 = A_{1,1}x_1 + A_{1,2}x_2 + A_{1,3}x_3 + A_{1,4}x_4 + e_1 \,, \quad (1)$$

$$y_2 = A_{2,5}x_5 + A_{2,1}x_1 + A_{2,6}x_6 + e_2 \,, \quad (2)$$

$$y_3 = A_{3,2}x_2 + A_{3,1}x_1 + A_{3,6}x_6 + e_3 \,, \quad (3)$$

$$y_4 = A_{4,7}x_7 + A_{4,1}x_1 + A_{4,2}x_2 + e_4 \,, \quad (4)$$

If we generalize the above equations for all voxels, then we need to compute the solution to the following linear system:

$$y = Ax + e \,, \quad (5)$$

where reconstruction x is an $Nx \times Ny \times Nz$ matrix ($Nx \times Ny$ voxels in a slice and Nz slices); measurements y is an $N_r \times N_c \times N_v$ matrix, also called *Sinogram*, with N_r detector rows, N_c detector channels, and N_v view angles; A is a 6 dimensional geometry system matrix and the size of A increases proportionally with the sizes of x and y. For simplicity, all following equations in this paper vectorize x matrix as an N dimensional vector, where $N = Nx \times Ny \times Nz$, y matrix as an M dimensional vector, where $M = N_r \times N_c \times N_v$, and A as an $M \times N$ system matrix. Each element of A matrix, $A_{i,j}$, represents the length of intersection between the X-ray for measurement y_i and voxel x_j. $A_{i,j}$ is zero when x_j is not on the path of the X-ray for y_i. To show how large A matrix is and why it leads to a large memory footprint, consider an x and an y of size $1000 \times 1000 \times 1000$ for both. In addition, we assume that the A matrix is stored in memory in floating point sparse matrix format with 0.1% non-zero elements. In this example, the entire A matrix then requires a memory size of 3638 terabytes!

In addition, if we trace out the memory locations of each voxel's measurements in Sinogram, the memory locations follow a sinusoidal trace in Sinogram as X-rays that pass through the same voxel hit different detector channels at different views. To update each voxel, cache lines must read and write data for the voxel following the same sinusoidal trace in memory. Cache lines, however, has a high cache miss rate as no cache lines can efficiently operate on data following a sinusoidal path in memory. In addition, each voxel has a unique trace in Sinogram and different voxel's traces often overlap at some intersections in Sinogram. Spatially close voxels have highly overlapped traces and more intersections. Spatially separated voxels have fewer overlapped traces with few or no intersections. Figure 1(j) shows an intersection colored in white between two voxel traces. To read and write data for voxel updates, locks or atomic operations must be applied to data at the intersections.

2.2 MBIR Formulation

Unfortunately, the solution to Equation (5) cannot be directly computed by taking a linear inverse, given that the data error e is unknown and cannot be measured. In addition, the size of reconstruction for super-resolution, N, is often much bigger than the size of measurements, M, leading to undetermined solutions to Equation (5). For example, Equations (1-4) have 4 measurements but 7 voxels to be computed. Therefore, voxels are undetermined with many possible solutions. For reconstruction x to be a determined solution, the size of measurements, M, must be at least as large as the size of reconstruction, N. To have a larger M size, the

CT detector resolution must be higher as M is proportional to the detector resolution.

To address the above problems without spending millions of dollars on a scanner for a higher detector resolution, the super-resolution MBIR is the solution to the following cost function:

$$x = \underset{x}{\operatorname{argmin}} \left\{ \frac{1}{2} \|y - Ax\|_{\Lambda}^2 + R(x) \right\} \tag{6}$$

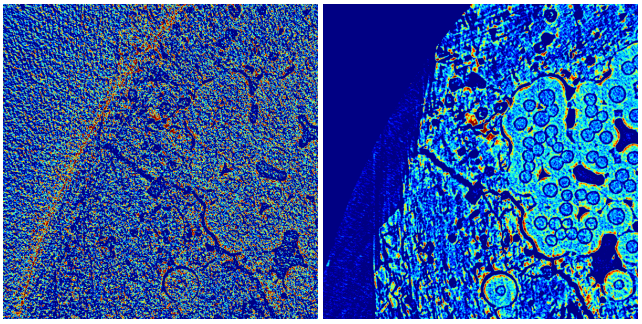
where Λ is a diagonal weighting matrix that models the CT system statistical noise. $R(x)$ is a prior model to estimate image details when measurements y are insufficient. In Equation (6), $\frac{1}{2}\|y - Ax\|_{\Lambda}^2$ is the forward model to fit reconstruction x from measurements y. The prior model, $R(x)$, is to penalize irregular image spatial property, suppress noise and estimate image details based on prior information. In this paper, we use a strictly convex Generalized Markov Random Field (GMRF) prior model to estimate each voxel's image details based on its neighboring voxels [23, 30] and Equation (6) has a unique global optimum given that both the forward and the prior models are strictly convex. Equation (6) can be viewed as a supervised machine learning problem that trains on a high variance dataset with limited size, whose forward model is the mean-square-error (MSE) of a weighted linear regression model and the prior model is a regularizer to prevent data over-fitting.

2.3 Related Work

To compute Equation (6) efficiently, there are two approaches: the volume partition and measurement partition methods. The volume partition methods include Grouped Coordinate Descent [6, 31], Parallel Iterative Coordinate Descent [10, 16, 20], and the Non-Uniform Parallel Super-Voxel (NU-PSV) algorithm [27]. To achieve parallelism, the volume partition methods split the reconstruction volume, x, into smaller voxel groups and assign each core with a voxel group to be updated in parallel. The scalability in such methods, however, is limited by the number of voxels in a reconstruction. In addition, the volume partition methods often have a high communication overhead with little speedup [6, 31]. Furthermore, the volume partition method cannot reduce the memory footprint for MBIR and the attainable image resolution is therefore limited.

The measurement partition methods, such as Parallel Mini-Batch Gradient Descent [15] and Ordered-Subset methods [1, 5], split measurements, y, and the system matrix into subsets and distribute them among cores. Each core reconstructs a private volume using its assigned subset, and then the private reconstruction volumes from all cores are merged into a consensus reconstruction. The measurement partition methods, however, are approximation methods and a convergence to the global optimum is not guaranteed. If measurements y are uniformly sampled among subsets, the methods converge close to the global optimum. If y are partitioned non-uniformly, the converged solution is far from the global optimum. In addition to the convergence issue, the amount of parallelism for the measurement partition methods is very limited as the number of parallel cores is at most the number of subsets, and subsets are usually few (≤ 40) as too many subsets slow down convergence [1].

Recently, the Consensus Equilibrium Method (CE) shows that it can partition measurements in any order and always converge to the global optimum with a proof in [4]. The paper in [4], however, only discusses CE mathematical theory and does not have any application to massive parallelism, memory footprint reduction or image reconstruction. The Multi-Agent Consensus Equilibrium (MACE) algorithm in [22] uses the CE Method for CT image reconstruction and partitions measurements into *uniform sparse-view subsets*, which are subsets of interleaved view angles. Taking Figure 1 as an example with 2 uniform sparse-view subsets, measurements for Figure 1(a) and (c) are in a subset, and measurements for Figure 1(b) and (d) are in the other subset. The MACE algorithm, however, is for single slice 2D reconstruction, and does not have a distributed node implementation or experiments to measure the reduction in the memory footprint.



(a) (b)

Figure 3: (a) An LR image with significant noise reconstructed from 1/4 of the uniformly sampled measurements. (b) A consensus HR image with much less noise and clearer details.

In comparison, AC-MBIR utilizes both volume partition and measurement partition for a fully 3D reconstruction. AC-MBIR not only scales to a large supercomputer with little communication overhead, but also guarantees a convergence to the global optimum. In addition, AC-MBIR proposes a new measurement partition method, grouped-view partition, that optimizes memory footprint reduction for super-resolution CT reconstruction.

2.4 The State of The Art Implementation

This paper compares AC-MBIR with the NU-PSV algorithm, which is currently the fastest MBIR implementation and scales to hundreds of thousands of cores, and was selected as a Gordon Bell Prize finalist [27]. We do not compare AC-MBIR with Mini-Batch Gradient Descent or Ordered-Subset (except convergence experiments in Figure 7) because their reconstructions are approximated and incomparable with AC-MBIR's exact solution. AC-MBIR is not compared with the MACE algorithm as MACE is for 2D single-slice reconstruction but AC-MBIR is for fully 3D reconstruction.

NU-PSV achieves its scalability by dividing a full volume x into sub-volumes, where a sub-volume is a consecutive group of slices. Nodes update different sub-volumes simultaneously, and different cores in a node update spatially separated super-voxels (SVs) in a sub-volume in parallel. Figure 2(a) shows an example volume divided into 2 sub-volumes and two SVs in a sub-volume, and each SV is a group of neighboring voxels in the shape of a $3 \times 2 \times 2$ rectangular cuboid. Since both spatially separated SVs and different

sub-volumes have little shared data to access and update in Sinogram, as explained in Section 2.1, NU-PSV has a low communication overhead by minimizing shared data among nodes and cores.

To achieve a high per-core performance, each core updates voxels in its assigned SV in sequence using the Coordinate Descent method [26, 27]. Note that measurements for an SV are closely grouped together in the shape of a sinusoidal band in the Sinogram, as shown in Figure 2(b), and each voxel in the SV follows a sinusoidal trace within the band. Example voxels and their corresponding traces in Sinogram are colored in red in Figure 2(a) and (b). Different voxels in an SV follow different, but highly overlapped, traces in the same band with many shared data in common. Therefore, measurements for voxels in an SV are close in memory and updating voxels in an SV in sequence is cache efficient with up to 187 times speedup on 20 cores [26].

Furthermore, NU-PSV reorganizes the sinusoidal memory layout for an SV into blocks of memory, called *super-voxel buffer* (SV buffer), that reads and writes data from memory in a completely regular pattern. To reorganize the memory layout for a good hardware prefetching and SIMD performance, NU-PSV introduces redundant measurements to the SV buffers and pads zero-value elements to the system A matrix [20, 26, 27]. Figure 2(b) shows two private SV buffers and outlines an example block in an SV buffer with a yellow boundary. Figure 2(c) shows the memory layout for the example block and features a completely regular access pattern, where red cells in the figure are memory locations for a voxel's measurements.

Although NU-PSV has the above benefits, its resolution is limited by the memory footprint and its scalability is limited by the number of slices. As each sub-volume must have at least 1 slice, the number of parallel nodes is at most the same as the number of slices. NU-PSV cannot increase its scalability further because it has depleted all sources of parallelism from dividing reconstruction x.

3 METHODS

3.1 Reducing Memory Footprint and Ensuring Convergence

To reduce the memory footprint, the system matrix A and measurements y should be partitioned and stored in distributed memory without corrupting their information. In this section, we will apply the CE Method to super-resolution MBIR, so that the system matrix and the measurements can be partitioned in **any** order, including any non-uniform sampling, and the final reconstruction from a partitioned system matrix and measurements is exactly the same as the solution to unpartitioned measurements. With the CE Method, the memory footprint for MBIR reduces significantly as a memory only needs to hold data for a partitioned subset.

To use the CE Method, we first partition measurements y into G subsets in any order, and the i^{th} measurement subset is denoted as y^i. Similarly, the system matrix A is partitioned in the same order, and the system matrix rows related to y^i is denoted as A^i. For each subset y^i, MBIR produces a low-resolution (LR) reconstruction, denoted as v^i. More precisely, an MBIR LR reconstruction from the i^{th} subset is the solution to a sub-problem, $f^i(v^i)$, defined below:

$$v^i \leftarrow \underset{v^i}{\operatorname{argmin}} f^i(v^i) = \underset{v^i}{\operatorname{argmin}} \left\{ \frac{1}{2} \left\| y^i - A^i v^i \right\|^2_{\Lambda^i} + \frac{R(v^i)}{G} \right\}, \quad (7)$$

where Equation (7) is the same as Equation (6), except that v^i is not a reconstruction from all measurements, but from the i^{th} subset. Then LR reconstructions from all subsets are merged into a consensus high-resolution (HR) reconstruction. Figure 3(a) shows an LR image reconstructed from 1/4 of the uniformly sampled measurements. Figure 3(b) shows a consensus HR solution after merging all LR reconstructions. Figure 3(a) has much image noise and lacks image details while Figure 3(b) suppresses noise and recovers image details.

A crucial question is how to merge LR reconstructions into a consensus HR solution, x. A popular method is Parallel Mini-Batch Gradient Descent with its pseudo-code in Algorithm 1 [15], where α is a fixed learning rate and ∇f^i is the gradient of f^i with respect to v^i. In step 4, v^i is updated along the direction of ∇f^i. After all LR reconstructions are updated, the consensus solution, x, is updated in step 6 by averaging all LR reconstructions. In step 8, the averaged LR reconstruction is used as the input to update v^i in the next iteration. If each subset only holds a single measurement, then Algorithm 1 is the Parallel Stochastic Gradient Descent [32]. If the learning rate, α, is replaced with a relaxed learning rate that diminishes across iterations, then Algorithm 1 is the Ordered-Subset algorithm [1]. To converge to an approximated solution close to the global optimum, Mini-Batch Gradient Descent requires that measurements and the system matrix are partitioned uniformly and the gradient, ∇f^i, needs to be approximately the same across different subsets. Otherwise, Mini-Batch Gradient Descent converges to a solution far from the actual global optimum.

Algorithm 1 Parallel Mini-Batch Gradient Descent Algorithm

INPUT: Uniformly partitioned measurements y and matrix A.
LOCAL: G: the number of subsets. α: A fixed learning rate. ∇f^i: A gradient of f^i. v_i: the i^{th} LR reconstruction.
OUTPUT: x: the consensus HR solution.

1: Initialize x, v^i
2: **while** Not Converged **do**
3: **for** each measurement subset i from 1 to G **do in parallel**
4: $v^i \leftarrow v^i - \alpha \nabla f^i(v^i)$
5: **end for**
6: $x \leftarrow \text{Average}(v^1, \cdots, v^G)$
7: **for** each measurement subset i from 1 to G **do in parallel**
8: $v^i \leftarrow x$
9: **end for**
10: **end while**

To ensure a convergence to the global optimum of Equation (6) and enable non-uniform subsets, we introduce the CE Method's proximal function to MBIR, denoted as $F_i(x)$, whose goal is to find a balance between an LR reconstruction, v^i, and the consensus HR solution, x:

$$v^i \leftarrow \underset{v^i}{\operatorname{argmin}} F^i(x) = \underset{v^i}{\operatorname{argmin}} \left\{ f^i(v^i) + \frac{\left\| v^i - x \right\|^2}{2\sigma^2} \right\}, \quad (8)$$

where the proximal map function input is x and output is v_i. The goal of $f^i(v^i)$ is to find v^i that fits the measurements for the subset,

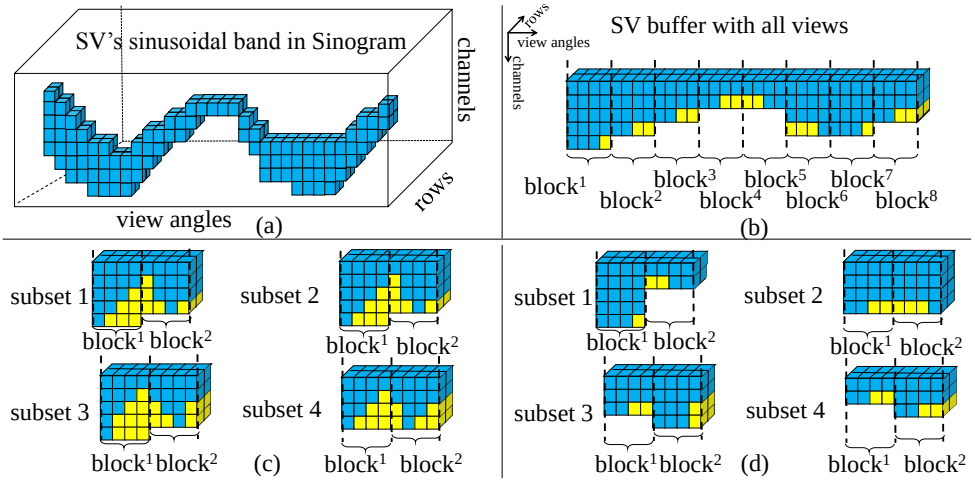


Figure 4: (a) A zoom-in on an SV's sinusoidal band in Sinogram. (b) An SV buffer copied from the band in Sinogram and every four columns is a block. Yellow cells in the SV buffer are redundant measurements. (c) 4 measurement subsets uniformly sampled across view angles. The yellow cell redundant measurements are significantly more than those in (b). (d) 4 grouped-view partitioned subsets. Notice that the redundant measurements are a lot fewer than (c).

and the goal of the penalizing term, $\frac{\|v^i - x\|^2}{2\sigma^2}$, is to constrain v^i to be close to the consensus HR solution, x. σ is a constant scalar that controls the strength of the penalizing term for fastest convergence. The optimal value for σ is found through convergence experiments and σ value increases when the number of subsets, G, increases. In each iteration, every subset evaluates its proximal map function, F^i, and produces an LR reconstruction. Then the consensus solution, x, is updated by fusing all LR reconstructions, where the fusing operation is more than a simple averaging, as we will discuss below. Iterations repeat until $x = v^i$ for $i = 1, 2, ..., N$ and the final consensus solution x is exactly the same as the solution to Equation (6).

Algorithm 2 Consensus Equilibrium MBIR

INPUT: Measurements y and matrix A partitioned in any order.
LOCAL: w^i: a temporary copy for v^i. u^i: input change to proximal map function, $F^i(x)$. Other variables are the same as in Algorithm 1.
OUTPUT: x: the consensus HR solution.

1: Initialize x, v^i, w^i, and u^i
2: **while** $x \neq v^i$, $i = 1, 2, ..., G$ **do**
3: **for** each subset i from 1 to G **do in parallel**
4: $v^i \leftarrow \text{argmin}_{v^i} F^i(x + u^i)$
5: $(w')^i \leftarrow w^i$
6: $w^i \leftarrow (2v^i - x - u^i)$
7: $w^i \leftarrow \rho w^i + (1 - \rho)(w')^i$, $0 \leq \rho \leq 1$
8: **end for**
9: $x \leftarrow \text{Average}(w^1, \cdots, w^G)$
10: **for** each subset i from 1 to G **do in parallel**
11: $u^i \leftarrow x - w^i$
12: **end for**
13: **end while**

Algorithm 2 summarizes how to apply the CE Method to MBIR and what the fusing operation does. For each subset, step 4 adds the proximal function input with u^i and produces an LR reconstruction. Step 6 adjusts the LR reconstruction and stores the new value in w^i. Step 7 damps the adjustment in step 6, where ρ is a constant scalar between 0 and 1 to control the degree of damping for fast convergence. The optimal value for ρ is chosen through convergence experiments and in general, more parallelism requires a smaller ρ for fastest convergence. Step 9 updates the consensus solution, x, by averaging w^i across all subsets. If $x \neq v^i$, the proximal function input change, u^i, is updated and iterations repeat until a convergence is reached, and a convergence proof is provided in Appendix A.

Note that Algorithm 2 allows the measurements and the system matrix to be partitioned in any order. In the special case of a uniform sparse-view partition, Algorithm 2 is the same as the MACE algorithm [22]. To show convergence rate comparison between Algorithm 1 and Algorithm 2, experiments in Figure 7 show that AC-MBIR has a much faster convergence than Mini-Batch Gradient Descent for both the uniform sparse-view and the non-uniform partitions. In addition, Mini-Batch Gradient Descent with few sparse-view subsets converges closer to the global optimum than the same algorithm with non-uniform partition. In contrast, AC-MBIR always converges to the global optimum for both the uniform and non-uniform partitions.

3.2 More Memory Footprint Reduction

The uniform sparse-view partition is widely used but does not optimize memory reduction. This section proposes a non-uniform grouped-view partition that further reduces memory footprint.

Figure 4(a) zooms in on an SV's 3D sinusoidal band in Sinogram and the band curves up and down in Sinogram, as explained in Section 2.1, with a varying width at different view angles [27]. In addition, neighboring view angles have similar widths and distant

view angles have dissimilar widths. Figure 4(b) copies data from the sinusoidal band to an SV buffer column by column (in channel direction) and every 4 view angles is a block. Each block is padded with redundant data to make the block a rectangular cuboid. For each redundant data, its corresponding system matrix row is also padded with zeros so that computations on the redundant data do not change the final reconstruction. In the example of Figure 4(b), redundant data are colored in yellow and the essential data for reconstruction are colored in blue. In addition, Figure 4(b) has 8 blocks and neighboring blocks are separated by short vertical dash lines. Columns in the same block have a constant width, although the width varies across different blocks. To have a good data vectorization, cache lines operate on data in the SV buffer block by block with a completely regular data access pattern.

Figure 4(c) uniformly partitions the SV buffer into 4 subsets across view-angles. Subset 1, for example, takes the first column of each block in the SV buffer and groups these columns into 2 new blocks in the shape of rectangular cuboids. Subset 2 takes the second column of each block in the SV buffer and group columns into 2 new blocks. Note that Figure 4(c) has 31% more redundant data than in the SV buffer. This phenomenon is due to the fact that uniformly partitioned subsets consist of data from distant view angles and have widely varying widths in each block. Therefore, significant amounts of redundant data are needed to pad each block into rectangular cuboids and the benefit of memory footprint reduction diminishes. To make matters even worse, the uniform sparse-view partition has more redundant data with more subsets.

To reduce redundant data, Figure 4(d) partitions the SV buffer into 4 grouped-view subsets. Each block in a subset consists of data from neighboring view angles, and different blocks in the same subset consist of data from distant view angles. In the example of Figure 4(d), subset 1 consists of SV buffer's first and fifth blocks. Subset 2 consists of SV buffer's second and sixth blocks. With the grouped-view partition design, Figure 4(d) has the same amount of redundant data as the unpartitioned SV buffer in Figure 4(b), and significantly fewer redundant data than the uniform sparse-view partition in Figure 4(c). Synchrotron imaging experiments in Table 1 shows that the redundant data for 16 uniform sparse-view subsets increase the memory footprint and the computations on data by 2.1 times, compared to the unpartitioned baseline. 16 grouped-view subsets, however, reduce the uniform sparse-view's memory footprint by 33% and the amount of computations by 38%. Table 2 further shows that the grouped-view partition has a 3.9 times speedup over the uniform sparse-view partition at 8640 nodes (587520 cores).

3.3 Improving Scalability

The CE Method has a unique source of parallelism across measurement subsets as different LR reconstructions can be updated in parallel. To improve scalability, the AC-MBIR algorithm is a new implementation that improves on the NU-PSV algorithm with an orthogonal level of parallelism across subsets, and can scale to the entire supercomputer even for a dataset with few slices.

The AC-MBIR algorithm divides the consensus HR solution, x, into sub-volumes with a unique color for each sub-volume. Example Figure 5(a) shows a consensus solution with 4 different

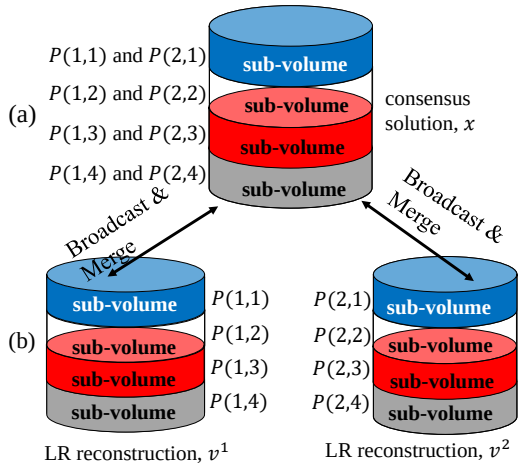
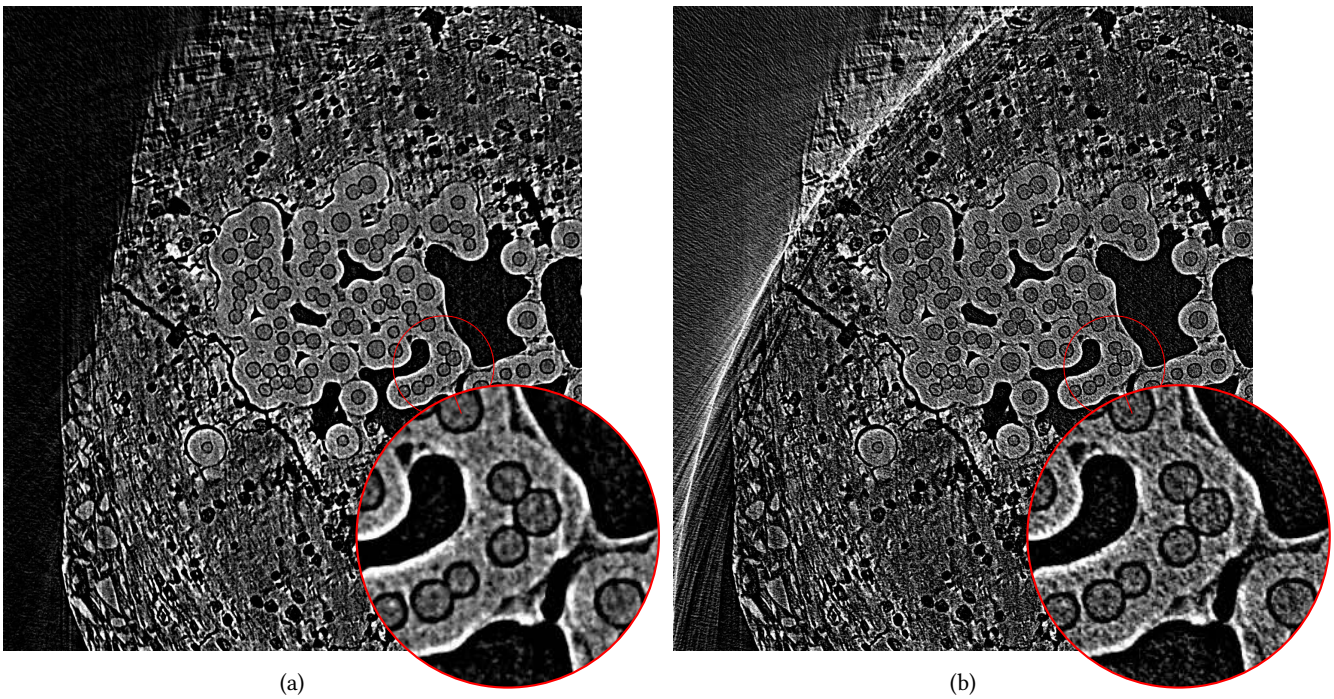


Figure 5: (a) Shows a consensus HR solution with 4 sub-volumes. (b) Shows 2 LR reconstructions with 4 sub-volumes. Nodes in the same cabinet update the same color sub-volumes in v^1 and v^2 as well as the same color sub-volume in the consensus solution.

color sub-volumes. Similarly, each LR reconstruction (2 subsets) in Figure 5(b) is also divided into 4 sub-volumes, with different sub-volumes across subsets colored the same if they merge to the same color sub-volume in the consensus HR solution. Then the AC-MBIR algorithm divides nodes in a supercomputer into clusters and cabinets, where the number of clusters equals to the number of measurement subsets and the number of cabinets is equal to the number of sub-volumes. Each node, denoted as $P(i, j)$, has dual identities with i as the conceptual cluster it belongs to and j as the conceptual cabinet it belongs to. Example Figure 5(b) shows 8 nodes divided into 2 clusters with $P(1, 1), P(1, 2), P(1, 3), P(1, 4)$ in a cluster. At the same time, nodes are also divided into 4 cabinets with $P(1, 1)$ and $P(2, 1)$ in a cabinet.

To map the node configuration with computations for image reconstruction, AC-MBIR lets each node $P(i, j)$ update the j^{th} sub-volume of the i^{th} LR reconstruction, and nodes in the same cluster share measurements and system matrix rows from the same subset, but update different color sub-volumes simultaneously. Nodes in the same cabinet have different measurements and system matrix rows, but update and merge the same color sub-volumes simultaneously across different subsets. After merging, nodes in the same cabinet broadcast the merged sub-volume in x to the same color sub-volumes in the LR reconstructions. To fully utilize parallel cores and SIMD computations, cores of a node are mapped to computations for a SV update in a sub-volume, and SIMD vectors are mapped to computations for a voxel update in a manner similar to the NU-PSV algorithm. See the pseudo code in Algorithm 3 for details. The fusing operation across subsets is achieved through MPI all reduce operations among nodes in the same cabinet. In step 14, neighboring sub-volumes exchange sub-volumes' boundary voxels and use the boundary voxels in evaluating the prior model in Equation (7).



(a) (b)

Figure 6: (a) A zoom-in image of a slice reconstructed by AC-MBIR with a 4.9 times super-resolution. (b) The same slice reconstructed by interpolation method with more noise and artifacts.

Algorithm 3 AC-MBIR

INPUT: The partitioned CT measurements and system matrix.
LOCAL: G: the number of subsets. v^i: the i^{th} LR reconstruction. $[v^i]^j$: the j^{th} sub-volume of v^i. $[v^i]^{j,k}$: the k^{th} SV of $[v^i]^j$. Similar notations apply to w^i, u^i.
OUTPUT: x: the consensus HR solution.

Initialize x, v^i, w^i, and u^i
while $x \neq v^i$, $i = 1, 2, ..., G$ **do**
  for each subset i **do in parallel across clusters**
    for each sub-volume $[v^i]^j \in v^i$, **do in parallel across nodes**
      for each SV $[v^i]^{j,k} \in [v^i]^j$, **do in parallel across cores**
        $[v^i]^{j,k} \leftarrow \text{argmin}_{[v^i]^{j,k}} F^i([x]^{j,k} + [u^i]^{j,k})$
        $[(w^{'})^i]^{j,k} \leftarrow [w^i]^{j,k}$
        $[w^i]^{j,k} \leftarrow (2[v^i]^{j,k} - [x]^{j,k} - [u^i]^{j,k})$
        $[w^i]^{j,k} \leftarrow \rho[w^i]^{j,k} + (1-\rho)\,[(w^{'})^i]^{j,k}$
        $[x]^{j,k} \leftarrow \text{Average}([w^1]^{j,k}, \cdots, [w^G]^{j,k})$, through MPI all reduce among nodes in a cabinet
        $[u^i]^{j,k} \leftarrow [x]^{j,k} - [w^i]^{j,k}$
      end for
    end for
    Neighboring sub-volumes exchange boundary voxels through MPI send/receive operations.
  end for
end while

Note that AC-MBIR is a unique asynchronous consensus optimization for reducing communication overhead. The common approach for consensus optimization selects a master node to update the consensus solution while the remaining worker nodes update LR reconstructions [30]. This approach, however, has a large global communication overhead as worker nodes' updates are bounded by how fast the master node updates and vice versa. AC-MBIR, however, has asynchronous updates and much smaller communication overhead because each node is a worker node and the master node at the same time. For example in Figure 5(b), $P(1,1)$ and $P(2,1)$ are in the same cabinet updating the blue sub-volumes of the LR reconstructions, but also update, merge and broadcast the blue sub-volume in the consensus solution. Therefore, AC-MBIR reduces communication overhead by restricting inter-node communications to nodes in the same cabinet and avoids global communications. Experimental results in Section 4.1 show that the asynchronous updates keep AC-MBIR's communication overhead at a low level, with 16% of its runtime spent on communication overhead at 272 nodes (18496 cores) and 13% of its runtime spent on communication overhead at 1080 nodes (73440 cores).

4 RESULTS

All performance numbers are evaluated on two datasets, a Ceramic Matrix Composite (CMC) dataset, imaged under synchrotron at Lawrence Berkeley National Laboratory, and an Iron Hydroxide (FeOOH) dataset, imaged under X-ray microscope at Air Force Research Lab. The CMC is a composite material for the next-generation

Grouped-view subsets	$G = 1$ (NU-PSV)	2	4	8	16
Nodes	108	216	432	864	1728
Memory footprint per node (Gigabytes)	58.0	31.4	18.3	11.2	8.5
Memory footprint reduction	1.0x	1.8x	3.2x	5.2x	6.8x
Run time (secs)	1681.0	779.8	342.6	258.4	215.7
Computation increase	1.0x	1.0x	1.0x	1.1x	1.3x
Strong scaling efficiency	100%	108%	123%	89%	63%
Bandwidth (petabytes/sec)	<0.1	<0.1	<0.1	<0.1	<0.1

(a) AC-MBIR with grouped-view subsets

Uniform-view subsets	$G = 1$ (NU-PSV)	2	4	8	16
Nodes	108	216	432	864	1728
Memory footprint per node (Gigabytes)	58.0	33.4	21.3	15.0	11.4
Memory footprint reduction	1.0x	1.7x	2.7x	3.9x	5.1x
Run time (secs)	1681.0	807.1	385.7	347.1	347.5
Computation increase	1.0x	1.0x	1.2x	1.5x	2.1x
Strong scaling efficiency	100%	104%	131%	91%	63%
Bandwidth (petabytes/sec)	<0.1	<0.1	<0.1	<0.1	<0.1

(b) AC-MBIR with uniform sparse-view subsets

Table 1: CMC dataset experiments for AC-MBIR with 108 sub-volumes but with different numbers of subsets, G. The left table is for grouped-view subsets and the right table is for uniform sparse-view subsets. More subsets lead to a rapidly decreasing memory footprint per node and an increased number of operations. Note that the grouped-view partition with all numbers of subsets has less memory footprint, shorter runtimes, and fewer computations than the uniform sparse-view partition.

aviation turbine-engine blades and space shuttles engine components [2]. The FeOOH is a material under development for nanorods.

The computing platform is the Cori Knights Landing supercomputer at the National Energy Research Scientific Computing Center (NERSC) with 68 cores per node. More information on the hardware can be found in the artifact description appendix. All data are stored and loaded into memory as single-precision floating point except for the system matrix. The system matrix is pre-computed as single-precision floating point, but stored in the NERSC data filesystem in a sparse matrix format and quantized as 4-bit unsigned characters for minimal memory footprint. When performing reconstructions, the quantized system matrix is loaded into memory and converted back to single-precision floating point [3, 20].

All runtimes are based on the entire program execution minus the IO time, and are measured at least three times with the average taken. The memory footprint is measured as the allocated memory for the program. Communication overhead is measured through NERSC IPM tool. Both the peak computing speed in petaflops and the average bandwidth in petabytes per second are measured through Intel SDE tool. To compute the peak performance and the average bandwidth, we count the number of FLOPS (72% single precision, 28% double) as well as the total bytes read and written on 4 nodes, and then extrapolate the performance to 8640 nodes.

For fastest convergence, we determine the optimal values for ρ and σ empirically. We use $\sigma = 10$ for 2 subsets, $\sigma = 20$ for 4 subsets, $\sigma = 45$ for 8 subsets, and $\rho = 0.8$ for all experiments. To decouple the convergence rate from runtimes, all programs in tables 1, 2, and 3 are run with 16 full iterations, where a full iteration is a cycle through N voxel updates. We pick 16 full iterations because the Normalized Root Mean Square Error (NRMSE) at 16 iterations is measured to be less than 2%, compared with a fully converged reconstruction.

4.1 Ceramic Matrix Composite (CMC) Dataset

The CMC dataset has a measurement size M of $2560 \times 2160 \times 1024$ (2560 channels, 2160 slices, and 1024 view angles), and the highest image resolution limited by the CT detector is 1609^2 voxels per slice. To demonstrate AC-MBIR's super-resolution performance, we reconstruct at an image resolution 4.9 times higher than the hardware limit, with 3600^2 voxels per slice and 2160 slices. A zoom-in picture of an example reconstruction is shown in Figure 6. The left image is a region reconstructed by AC-MBIR, and the right image is the same region reconstructed by the FBP interpolation method, whose reconstruction has more image noise and artifacts than AC-MBIR's. For example, the upper left corner of the FBP interpolation image has many braiding artifacts, while AC-MBIR image has no braiding artifacts. Therefore, material scientists can use AC-MBIR to better examine the micro-structure at super-resolution.

Table 1 summarizes AC-MBIR and NU-PSV's performance with different numbers of subsets, G, but with the same number of sub-volumes (108 sub-volumes). The left table is for grouped-view subsets and the right table is for uniform sparse-view subsets. Nodes are configured such that the number of clusters equals to the number of subsets G, and the number of cabinets equals to 108. Row 1 of the tables shows different numbers of subsets and AC-MBIR with $G = 1$ is identical to the NU-PSV algorithm.

Row 2 is the number of nodes (68 cores per node), computed as the number of cabinets multiplied by the number of clusters, and AC-MBIR with 16 subsets has 16 times more scalability than NU-PSV. Row 3 is the memory footprint per node and row 4 is AC-MBIR's memory footprint reduction over NU-PSV. Both the grouped-view and the uniform sparse-view partitions significantly reduce the memory footprint by 6.8 times and 5.1 times respectively, although the grouped-view partition has 25% more memory footprint reduction. In addition, both the grouped-view and the uniform sparse-view partitions reduce the memory footprint non-linearly with more subsets, as the number of redundant data in the subsets increases non-linearly with more subsets.

Row 5 is the program runtimes and NU-PSV's runtime is 1681 seconds. In contrast, the grouped-view partition with 16 subsets has a smaller runtime of 215.7 seconds by reducing the memory pressure. The uniform sparse-view partition also has a much smaller

Number of sub-volumes	68	135	270	540	1080	2160
Nodes	272	540	1080	2160	4320	8640
Memory footprint per node (Gigabytes)	20.7	17.4	15.7	14.9	14.5	14.3
Memory footprint reduction	2.8x	3.3x	3.7x	3.9x	4.0x	4.1x
Run time (secs)	558.0	271.7	147.2	84.4	51.4	40.0
Computation increase	1.0x	1.0x	1.0x	1.0x	1.0x	1.0x
Strong scaling efficiency	120%	124%	114%	100%	82%	53%
Bandwidth (petabytes/sec)	<0.1	<0.1	0.1	0.4	1.4	3.5

(a) AC-MBIR with grouped-view subsets

Number of sub-volumes	68	135	270	540	1080	2160
Nodes	272	540	1080	2160	4320	8640
Memory footprint per node (Gigabytes)	23.7	20.4	18.7	17.9	17.5	17.3
Memory footprint reduction	2.4x	2.8x	3.1x	3.2x	3.3x	3.4x
Run time (secs)	654.6	326.4	174.3	159.2	154.0	154.0
Computation increase	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x
Strong scaling efficiency	122%	124%	116%	63%	33%	16%
Bandwidth (petabytes/sec)	<0.1	<0.1	0.1	0.3	0.5	1.1

(b) AC-MBIR with uniform sparse-view subsets

Table 2: CMC dataset experiments for AC-MBIR with 4 subsets but with different numbers of sub-volumes. The left table is for grouped-view subsets and the right table is for uniform sparse-view subsets. With more sub-volumes, the memory footprint per node decreases slowly and the number of operations stays the same. The grouped-view subsets with 1080 sub-volumes have a 17% smaller memory footprint, a 3.9 times faster runtime, and a 3.3 times higher strong scaling efficiency than the uniform sparse-view subsets.

runtime than NU-PSV with 347.5 seconds at 16 subsets. The runtime for the uniform sparse-view partition, however, is larger than that for the grouped-view partition because the uniform sparse-view partition has a larger memory footprint and more redundant data.

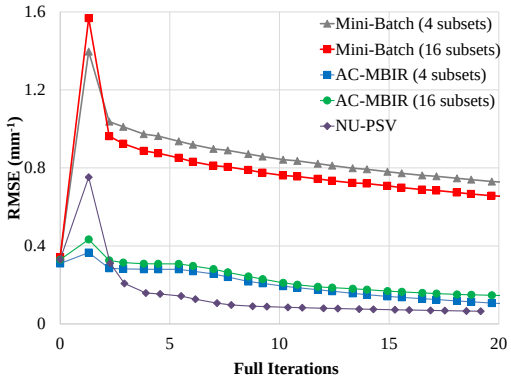
Row 6 is the increase in floating point operations, compared to those for NU-PSV. Floating point operations increase non-linearly with more subsets as more subsets incur more redundant data and operations on the data. With 16 subsets, the grouped-view partition has 38% less redundant data than the uniform sparse-view partition. Row 7 is the parallel strong scaling efficiency compared to NU-PSV ($G = 1$) and row 8 is the average bandwidth. The grouped-view and the uniform sparse-view subsets have similar strong scaling efficiencies with a super-linear speedup up to 4 subsets. From the strong scaling efficiency numbers, we can conclude that the grouped-view partition does not have a better strong scaling efficiency over the uniform sparse-view partition. The grouped-view partition, however, has smaller runtimes by reducing memory footprint and computations.

Table 2 summarizes AC-MBIR's performance with the same number of view-subsets (4 view-subsets) but with different numbers of sub-volumes. With the same number of subsets, the memory allocation for the system matrix does not change across different numbers of sub-volumes. However, more sub-volumes reduce the number of slices and measurements per sub-volume. The memory footprint in row 3 thereby decreases slowly with more sub-volumes. In row 4, the grouped-view partition reduces more memory footprint than the sparse-view partition, as the former has fewer redundant data than the latter. In addition, the number of operations does not increase with more sub-volumes as the amount of redundant data remains the same for the same number of subsets.

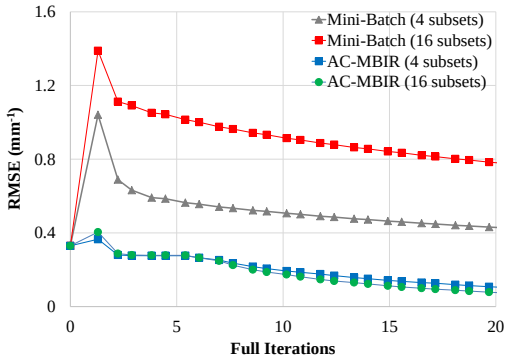
Row 5 shows the runtimes and the grouped-view partition with 2160 sub-volumes (90% of Cori supercomputer with 587520 cores) has a 3.9 times speedup over the uniform sparse-view partition, a 46.5 times speedup over the NU-PSV algorithm, and has a measured peak computing performance of 1.5 petaflops at 8640 nodes. If not counting the MPI Allreduce overhead in step 10 of Algorithm 3, the peak performance number is 5.6 petaflops. The strong scaling efficiencies in row 7 show that the grouped-view partition has a 100% strong scaling efficiency at 2160 nodes, compared to NU-PSV. AC-MBIR has a high strong scaling efficiency for two reasons: (1) a smaller memory footprint leads to more data kept in cache, and (2) a modest communication overhead. The IPM tool shows that grouped-view subsets has a low communication overhead with 16% of its runtime spent on communication overhead at 272 nodes, 13% communication overhead at 1080 nodes, and an increased communication overhead of 73% at 4320 nodes. Without AC-MBIR's asynchronous updates, the communication overhead is more than 50% at few numbers of nodes.

Row 8 is AC-MBIR's average bandwidth and the grouped-view partition achieves a bandwidth of 3.5 petabytes per second at 8640 nodes, which is more than 35 times higher than that for NU-PSV. AC-MBIR has a good bandwidth because AC-MBIR's runtimes are bounded by memory and most of the runtimes are spent on reorganizing sinusoidal data for high cache locality, regular data access pattern, and a low memory footprint. In contrast, AC-MBIR's floating point operations are much fewer than its memory operations, leading to an impressive bandwidth of 3.5 petabytes per second, but a modest peak performance of 1.5 petaflops.

Convergence evaluations are another interesting topic. Figure 7 compares the convergence rate for the first 20 full iterations for AC-MBIR, NU-PSV and Mini-Batch Gradient Descent on an example CMC dataset sub-volume with 20 slices, with Figure 7(a) for the grouped-view partition and Figure 7(b) for the uniform sparse-view partition. For the grouped-view partition, the Mini-Batch Gradient Descent never converges to the global optimum even with more iterations, and converges to a solution far from the global optimum, whereas AC-MBIR converges to the global optimum rapidly both with 4 and 16 subsets. Compared to NU-PSV, AC-MBIR has a similar convergence rate at 20 full iterations. In addition, we can observe that all algorithms has a RMSE spike in early iterations



(a) Convergence for grouped-view subsets.



(b) Convergence for uniform sparse-view subsets.

Figure 7: (a) The CMC dataset convergence plot for Mini-Batch Gradient Descent and AC-MBIR with 4 and 16 grouped-view subsets. (b) The convergence plot for the two algorithms with 4 and 16 uniform sparse-view subsets.

from a convergence overshoot caused by the excessive updates. For the uniform sparse-view partition, Mini-Batch Gradient Descent with 4 subsets converges to a solution close to the global optimum, although its convergence rate is still much slower than AC-MBIR.

4.2 Iron Hydroxide (FeOOH) Dataset

The FeOOH dataset is much smaller than the CMC dataset, with a measurement size M of $1024 \times 1024 \times 900$ (1024 channels, 1024 slices and 900 view angles). The highest image resolution limited by the CT detector is 960^2 voxels per slice. We reconstruct at an image resolution 1.14 times higher than the hardware limit, with 1024^2 voxels per slice and 1024 slices. Table 3 summaries AC-MBIR and NU-PSV's performance. The second row is NU-PSV's runtimes and NU-PSV only scales to 1024 nodes, as each node needs to operate on a sub-volume with at least 1 slice. The third row is AC-MBIR's runtimes with 4 grouped-view subsets and AC-MBIR has a 4 times higher scalability than NU-PSV. The fourth row is AC-MBIR's speedup over NU-PSV with a 3.7 times speedup at 4096 nodes (278528 cores). Since NU-PSV cannot scale to 4096 nodes, the speedup at 4096 nodes is AC-MBIR's runtime compared with NU-PSV's best runtime at 1024 nodes. The fifth row is AC-MBIR's memory footprint reduction over NU-PSV and AC-MBIR's memory footprint at 4096 nodes is 3.7 times smaller than that for NU-PSV.

Nodes	16	64	256	1024	4096
NU-PSV runtimes (secs)	1776.4	457.5	110.8	33.1	NA
AC-MBIR runtimes (secs)	556.2	142.2	47.1	18.0	10.1
Speedup	3.2x	3.2x	2.4x	1.8x	3.3x
Memory footprint reduction	0.9x	1.9x	3.0x	3.5x	3.7x

Table 3: Performance comparison for the FeOOH dataset between AC-MBIR (with 4 grouped-view subsets) and NU-PSV.

5 DISCUSSION

AC-MBIR has a significant memory footprint reduction, a smaller runtime and more scalability over NU-PSV. Its parallelism across data subsets and sub-volumes, however, has advantages and disadvantages. The data subsets parallelism has better memory footprint reduction over the sub-volume parallelism, but has worse strong scaling efficiency and a slower runtime. The sub-volume parallelism has a better strong scaling efficiency and a faster runtime, but performs worse for memory footprint reduction. To achieve the best performance, node configurations between the two parallelisms have to be determined empirically as there is no theory that can accurately pre-determine the node configuration. One way for node configuration is to follow the ablation studies in Table 1 to find the number of subsets with best performance, while fixating on the same number of sub-volumes. Then, nodes are configured by following the studies in Table 2 to find the number of sub-volumes with best performance, while fixating on the same number of subsets.

6 IMPLICATIONS

AC-MBIR can be applied to supervised machine learning problems with the following form:

$$x = \underset{x}{\operatorname{argmin}} \left\{ G(x) + R(x) \right\} , \tag{9}$$

where x is the output; $G(x)$ is a statistical model that estimates x from a limited size noisy training dataset, denoted as y, and $R(x)$ is a regularizer to prevent data over-fitting. If $G(x)$ is a weighted linear regression model, then Equation (9) has the same form as Equation (6), where M is the dataset size; A is a sparse system matrix; Λ is a weighting matrix; and $R(x)$ is an L_0 norm, L_1 norm or other regularizer.

If the output x is viewed as an image or a volume, then Equation (9) with a weighted linear regression training model is a cost function for image reconstruction problems, such as CT, PET, MRI, electron microscopy, synchrotron, neutron, proton, and ultrasound imaging. If x is a matrix of measurements or an object under test, then Equation (9) with a weighted linear regression training model is a cost function for compressive sensing problems and predicts the state of the physical world from noisy sensor measurements.Such applications include autonomous navigation, depth sensors, digital holography, geophysical sensing, radar, radio astronomy, crystallography, machine learning techniques such as the least absolute shrinkage and selection operator.

AC-MBIR partitions Equation (9) into sub-problems and merge the sub-problem solutions into a consensus one with a guaranteed convergence if Equation (9) is convex. This algorithm has two benefits: (1) reducing memory pressure by distributing data among sub-problems, and (2) mapping to a supercomputer for fast computations. To solve Equation (9), AC-MBIR non-uniformly partitions the dataset y into G data subsets and data in the same subset are highly similar and different subsets are least similar. Each subset has a new training model, $g_i(x)$, such that $G(x) = \sum_{i=1}^{G} g_i(x)$. Each subset solves the proximal map function formulated in Equation (8) and the individual solutions from all subsets are merged asynchronously into the consensus solution by following Algorithm 3.

A MATHEMATICAL PROOF FOR ALGORITHM 2

The consensus-equilibrium Method [4, 22] provides a parallel method to solve the following MBIR optimization problem:

$$x = \underset{x}{\operatorname{argmin}} \left\{ \frac{1}{2} \left\| y - Ax \right\|_{\Lambda}^{2} + R(x) \right\} = \underset{x}{\operatorname{argmin}} \sum_{i=1}^{G} f^{i}(x) \,, \qquad (10)$$

where f^i has the same definition as in (7), and represents a sub-problem for the i^{th} measurement subset. G is the total number of subsets. Theorem 1 of [4] shows that finding the global minimum to Equation (10), x, is exactly the same as finding the average of $\{w^1, ..., w^G\}$, denoted as $\overline{w}$, for the following system of equations:

$$\begin{bmatrix} F^1(w^1) \\ \vdots \\ F^N(w^G) \end{bmatrix} = \frac{1}{G} \begin{bmatrix} w^1 + w^2 + \cdots + w^G \\ \vdots \\ w^1 + w^2 + \cdots + w^G \end{bmatrix} = \begin{bmatrix} \overline{w} \\ \vdots \\ \overline{w} \end{bmatrix} , \qquad (11)$$

where F^i is the proximal map function for the i^{th} measurement subset, as defined before in Equation (8). Then, to prove that the output of Algorithm 2 is the same as the solution to Equation (10), we need to demonstrate that the output of Algorithm 2 equals to $\overline{w}$.

If we let $\mathbf{F}$ represents the stacked function for $[F^1, \cdots, F^G]^T$, $\mathbf{w}$ as the list, $[w^1, ..., w^G]^T$, and function $\mathbf{D}$ stacks G copies of $\overline{w}$, then we can rewrite Equation (11) with a more compact short-hand notation:

$$\mathbf{F}(\mathbf{w}) = \mathbf{D}(\mathbf{w}) \,, \qquad (12)$$

After multiplying both sides of Equation (12) with a constant 2 and subtract them with $\mathbf{w}$, we get

$$(2\mathbf{F} - \mathbf{I})\mathbf{w} = (2\mathbf{D} - \mathbf{I})\mathbf{w} \qquad (13)$$

where $\mathbf{I}$ is the identity function. Note that for any $\mathbf{w}$, applying function $\mathbf{D}$ twice to $\mathbf{w}$ yields the same result as applying $\mathbf{D}$ only once to $\mathbf{w}$. So, $\mathbf{D}^2\mathbf{w} = \mathbf{D}\mathbf{w}$, and consequently, $(2\mathbf{D} - \mathbf{I})^2\mathbf{w} = \mathbf{w}$. Based on this special property, we apply function $(2\mathbf{D} - \mathbf{I})$ to both sides of equation (13) and we obtain

$$(2\mathbf{D} - \mathbf{I})(2\mathbf{F} - \mathbf{I})\mathbf{w} = \mathbf{w} \qquad (14)$$

To find $\mathbf{w}$ that satisfies Equation (14), $\mathbf{w}$ can be computed iteratively by finding the fixed-point to Equation (14). In every iteration, $\mathbf{w}$ is updated as below:

$$\mathbf{w} \leftarrow \rho\mathbf{w} + (1 - \rho)\mathbf{w}' \,, \qquad (15)$$

where ρ is a convergence parameter chosen between 0 and 1 and $\mathbf{w}'$ stores a copy of $\mathbf{w}$ from the previous iteration. Note that Equation (15) is exactly the same as step 7 in Algorithm 2. After updates in Equation (15) are converged to a fixed-point vector, the final output, $\overline{w}$, is computed as the average for all elements in $\mathbf{w}$, namely $\{w^1, ..., w^G\}$.

ACKNOWLEDGMENTS

The authors would like to thank the National Energy Research Scientific Computing Center (NERSC) for providing supercomputing resources under contract No. DE-AC02-05CH11231. This research was supported by the National Science Foundation (NSF) under Award CCF-1763896. Additional support was provided by the DHS ALERT Center for Excellence under Grant Award 2013-ST-061-ED0001.

REFERENCES

[1] S. Ahn and J. A. Fessler. 2003. Globally Convergent Image Reconstruction for Emission Tomography Using Relaxed Ordered Subsets Algorithms. *IEEE Transactions on Medical Imaging* 22, 5 (May 2003), 613–626. https://doi.org/10.1109/TMI.2003.812251

[2] GE Aviation. 2016. Space Age Ceramics Are Aviation's New Cup Of Tea. https://www.ge.com/reports/space-age-cmcs-aviations-new-cup-of-tea/. (2016).

[3] T. Balke, S. Majee, G. T. Buzzard, S. Poveromo, P. Howard, M. A. Groeber, J. McClure, and C. A. Bouman. 2018. Separable Models for cone-beam MBIR Reconstruction. In *Computational Imaging XVI, Burlingame, California, USA, 28 Jan 2018 - 1 Feb 2018.*

[4] G. Buzzard, S. Chan, S. Sreehari, and C. Bouman. 2018. Plug-and-Play Unplugged: Optimization-Free Reconstruction Using Consensus Equilibrium. *SIAM Journal on Imaging Sciences* 11, 3 (2018), 2001–2020. https://doi.org/10.1137/17M1122451 arXiv:https://doi.org/10.1137/17M1122451

[5] H. Erdogan and J. Fessler. 1999. Ordered Subsets Algorithms for Transmission Tomography. *Physics in Medicine & Biology* 44(11) (1999).

[6] J. A. Fessler, E. P. Ficaro, N. H. Clinthorne, and K. Lange. 1997. Grouped-Coordinate Ascent Algorithms for Penalized-Likelihood Transmission Image Reconstruction. *IEEE Transactions on Medical Imaging* 16(2) (1997).

[7] J. A. Fessler and D. Kim. 2011. Axial Block Coordinate Descent (ABCD) Algorithm for X-ray CT Image Reconstruction. In *11th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine.*

[8] M. C. Fonseca, B. H. S. Araujo, C. S. B. Dias, N. L. Archilha, D. P. A. Neto, E. Cavalheiro, H. Westfahl, A. J. R. da Silva, and K. G. Franchini. 2018. High-Resolution Synchrotron-Based X-ray Microtomography as A Tool to Unveil The Three-Dimensional Neuronal Architecture of The Brain. *Scientific Reports* 8, 1 (Aug 2018), 12074.

[9] C. Fournier, F. Jolivet, L. Denis, N. Verrier, E. Thiebaut, C. Allier, and T. Fournel. 2017. Pixel Super-Resolution in Digital Holography by Regularized Reconstruction. *Applied Optics* 56, 1 (Jan 2017), 69–77. https://doi.org/10.1364/AO.56.000069

[10] S. Ha and K. Mueller. 2018. A GPU-Accelerated Multivoxel Update Scheme for Iterative Coordinate Descent (ICD) Optimization in Statistical Iterative CT Reconstruction (SIR). *IEEE Transactions on Computational Imaging* 4, 3 (Sep. 2018), 355–365. https://doi.org/10.1109/TCI.2018.2833622

[11] C. Jiang, Q. Zhang, R. Fan, and Z. Hu. 2018. Super-Resolution CT Image Reconstruction Based on Dictionary Learning and Sparse Representation. *Scientific Reports* 8(1) (2018).

[12] P. Jin, C. A. Bouman, and K. D. Sauer. 2013. A Method for Simultaneous Image Reconstruction and Beam Hardening Correction. In *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC).* 1–5.

[13] E. A. Kazerooni. 2001. High-resolution CT of the lungs. *American Journal of Roentgenology* 177, 3 (Sep 2001), 501–519.

[14] A. Korn, M. Fenchel, B. Bender, S. Danz, T. K. Hauser, D. Ketelsen, T. Flohr, C. D. Claussen, M. Heuschmid, U. Ernemann, and H. Brodoefel. 2012. Iterative Reconstruction in Head CT: Image Quality of Routine and Low-Dose Protocols in Comparison with Standard Filtered Back-Projection. *American Journal of Neuroradiology* 33, 2 (Feb 2012), 218–224.

[15] M. Li, T. Zhang, Y. Chen, and A. J. Smola. 2014. Efficient Mini-batch Training for Stochastic Optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14).* ACM, New York, NY, USA, 661–670. https://doi.org/10.1145/2623330.2623612

[16] X. Li, Y. Liang, W. Zhang, T. Liu, H. Li, G. Luo, and M. Jiang. 2018. cuMBIR: An Efficient Framework for Low-dose X-ray CT Image Reconstruction on GPUs.

In *Proceedings of the 2018 International Conference on Supercomputing (ICS '18)*. ACM, New York, NY, USA, 184–194. https://doi.org/10.1145/3205289.3205309

[17] P. Milanfar. 2010. *Super-Resolution Imaging*. CRC Press.

[18] K. A. Mohan, S. V. Venkatakrishnan, J. W. Gibbs, E. B. Gulsoy, X. Xiao, M. D. Graef, P. W. Voorhees, and C. A. Bouman. 2015. TIMBER: A Method for Time-Space Reconstruction from Interlaced Views. *IEEE Transactions on Computational Imaging* 1, 2 (June 2015), 96–111.

[19] Z. Nadir, M. S. Brown, M. L. Comer, and C. A. Bouman. 2017. A Model-Based Iterative Reconstruction Approach to Tunable Diode Laser Absorption Tomography. *IEEE Transactions on Computational Imaging* 3, 4 (Dec 2017), 876–890.

[20] A. Sabne, X. Wang, S. J. Kisner, C. A. Bouman, A. Raghunathan, and S. P. Midkiff. 2017. Model-Based Iterative CT Image Reconstruction on GPUs. In *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '17)*. ACM, New York, NY, USA, 207–220. https://doi.org/10.1145/3018743.3018765

[21] S. Sreehari, S. V. Venkatakrishnan, K. L. Bouman, J. P. Simmons, L. F. Drummy, and C. A. Bouman. 2016. Multi-Resolution Data Fusion for Super-Resolution Electron Microscopy. *CoRR* abs/1612.00874 (2016). arXiv:1612.00874 http://arxiv.org/abs/1612.00874

[22] V. Sridhar, G. T. Buzzard, and C. A. Bouman. 2018. Distributed Framework for Fast Iterative CT Reconstruction from View-subsets. In *Computational Imaging XVI, Burlingame, California, USA, 28 Jan 2018 - 1 Feb 2018*.

[23] J. B. Thibault, K. D. Sauer, C. A. Bouman, and J. Hsieh. 2007. A Three-Dimensional Statistical Approach to Improved Image Quality for Multi-Slice Helical CT. *Medical Physics* 34(11) (2007).

[24] S. V. Venkatakrishnan, L. F. Drummy, M. Jackson, M. D. Graef, J. Simmons, and C. A. Bouman. 2013. High Angle Annular Dark Field- Scanning Transmission Electron Microscope (HAADF-STEM) Tomography. *IEEE Transactions on Image Processing* 22(1) (2013).

[25] X. Wang, K. A. Mohan, S. J. Kisner, C. A. Bouman, and S. P. Midkiff. 2016. Fast Voxel Line Update for Time-Space Image Reconstruction. In *The 41st IEEE International Conference on Acoustics, Speech and Signal Processing*.

[26] X. Wang, A. Sabne, S. J. Kisner, A. Raghunathan, C. A. Bouman, and S. P. Midkiff. 2016. High Performance Model Based Image Reconstruction. *SIGPLAN Notice* 51, 8, Article 2 (Feb. 2016), 12 pages. https://doi.org/10.1145/3016078.2851163

[27] X. Wang, A. Sabne, P. Sakdhnagool, S. J. Kisner, C. A. Bouman, and S. P. Midkiff. 2017. Massively Parallel 3D Image Reconstruction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, Article 3, 12 pages. https://doi.org/10.1145/3126908.3126911

[28] R. Yan, S. V. Venkatakrishnan, J. Liu, C. A. Bouman, and W. Jiang. 2019. MBIR: A Cryo-ET 3D Reconstruction Method that Effectively Minimizes Missing Wedge Artifacts and Restores Missing Information. *Journal of Structural Biology* (Mar 2019).

[29] Z. Yu, J-B. Thibault, C. A. Bouman, K. D. Sauer, and J. Hsieh. 2011. Fast Model-Based X-Ray CT Reconstruction using Spatially Nonhomogeneous ICD Optimization. *IEEE Transactions on Image Processing* 20(1) (2011).

[30] R. Zhang and J. T. Kwok. 2014. Asynchronous Distributed ADMM for Consensus Optimization. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org, II–1701–II–1709.

[31] J. Zheng, S. S. Saquib, K. Sauer, and C. A. Bouman. 2000. Parallelizable Bayesian Tomography Algorithms with Rapid, Guaranteed Convergence. *IEEE Transactions on Image Processing* 9(10) (2000).

[32] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. 2010. Parallelized Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems 23*. Curran Associates, Inc., 2595–2603. http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.pdf

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran all experiments on the NERSC's Cori Knights Landing supercomputer (68 cores per node). The inter-node parallelism is implemented through Intel MPI 2018.up1. The inter-core parallelism is implemented through Intel OpenMP. The communication overhead is collected by the NERSC IPM tool (https://www.nersc.gov/users/software/performance-and-debugging-tools/ipm/). The average computing speed in petaflops is computed by the number of floating point operations divided by the runtime, where the floating point operations are collected by the Intel SDE tool on a single node and then extrapolated to 4320 nodes, and the runtimes are collected by MPI timer. Information about the NERSC Intel SDE tool can be found at https://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: All author-created hardware artifacts are maintained in a public repository under an OSI-approved license.

Data Artifact Availability: Some author-created data artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Proprietary Artifacts: There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

List of URLs and/or DOIs where artifacts are available:

```
High level summary of the NERSC Cori Knights Landing
↪ (KNL) supercomputer configuration:
https://www.nersc.gov/users/computational-systems/co⌋
↪ ri/configuration/cori-intel-xeon-phi-nodes/

KNL supercomputer processor architecture for each
↪ node:
https://www.nersc.gov/users/computational-systems/co⌋
↪ ri/configuration/knl-processor-modes/

KNL interconnect network topology:
https://www.nersc.gov/users/computational-systems/co⌋
↪ ri/configuration/interconnect/
https://www.nersc.gov/users/computational-systems/co⌋
↪ ri/configuration/compute-nodes-topology/
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: NERSC Cori Knights Landing Supercomputer, 68 cores and 128 GB memory per node, each node contains an Intel Xeon Phi Processor 7250 at 1.40GHz

Operating systems and versions: SuSE Linux Enterprise Server version 12.3

Compilers and versions: Intel MPI 2018.up1

Applications and versions: Intel SDE 7.49.0, IPM 2.0.5

Key algorithms: AC-MBIR, NU-PSV, Mini-Batch Gradient Descent

Input datasets and versions: Berkeley Synchrotron Dataset: Ceramic Matrix Composite. Air Force Research Lab Microscopy Dataset: Iron Hydroxide

Paper Modifications: No modification to the hardware, compilers, or performance measurement tools.

Output from scripts that gathers execution environment information.

```
LESSKEY=/etc/lesskey.bin
MODULE_VERSION_STACK=3.2.10.6
KSH_AUTOLOAD=1
ZAP_LIBPATH=/opt/ovis/lib64/ovis-lib
MKLROOT=/opt/intel/compilers_and_libraries_2018.1.16⌋
↪ 3/linux/mkl
PE_LIBSCI_VOLATILE_PRGENV=CRAY GNU INTEL
PE_SMA_DEFAULT_PKGCONFIG_VARIABLES=PE_SMA_COMPFLAG_@⌋
↪ prgenv@
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_mic_knl=160
MANPATH=/opt/intel/compilers_and_libraries_2018.1.16⌋
↪ 3/linux/mpi/man:/usr/common/software/man:/usr/co⌋
↪ mmon/mss/man:/usr/common/nsg/man:/opt/cray/pe/mp⌋
↪ t/7.7.3/gni/man/mpich:/opt/cray/pe/atp/2.1.3/man⌋
↪ :/opt/cray/alps/6.6.43-6.0.7.0_26.4__ga796da3.ar⌋
↪ i/man:/opt/cray/job/2.2.3-6.0.7.0_44.1__g6c4e934⌋
↪ .ari/man:/opt/cray/pe/pmi/5.0.14/man:/opt/cray/p⌋
↪ e/libsci/18.07.1/man:/opt/cray/pe/man/csmlversio⌋
↪ n:/opt/cray/pe/craype/2.5.15/man:/opt/intel/comp⌋
↪ ilers_and_libraries_2018.1.163/linux/man/common:⌋
↪ /usr/syscom/nsg/man:/opt/cray/pe/modules/3.2.10.⌋
↪ 6/share/man:/usr/local/man:/usr/share/man:/opt/c⌋
↪ ray/share/man:/opt/cray/pe/man:/opt/cray/share/m⌋
↪ an
NNTPSERVER=news
PE_PAPI_DEFAULT_ACCEL_FAMILY_LIBS_nvidia=,-lcupti,-l⌋
↪ cudart,-lcuda
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_sandybridge=8.6
PE_PETSC_DEFAULT_GENCOMPS_CRAY_skylake=86
PE_TPSL_DEFAULT_GENCOMPS_INTEL_x86_skylake=160
```

```
PE_CXX_PKGCONFIG_LIBS=mpichcxx
DVS_MAXNODES=1__
XDG_SESSION_ID=102161
HOSTNAME=cori12
CRAY_UDREG_INCLUDE_OPTS=-I/opt/cray/udreg/2.3.2-6.0.
↪ 7.0_33.18__g5196236.ari/include
PE_FFTW_DEFAULT_TARGET_mic_knl=mic_knl
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_mic_knl=16.0
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_interlagos=160
PE_TRILINOS_DEFAULT_GENCOMPS_CRAY_x86_64=87
XKEYSYMDB=/usr/X11R6/lib/X11/XKeysymDB
CRAY_SITE_LIST_DIR=/etc/opt/cray/pe/modules
LIBRARYMODULES=acml:alps:cray-dwarf:cray-fftw:cray-g
↪ a:cray-hdf5:cray-hdf5-parallel:cray-libsci:cray-
↪ libsci_acc:cray-mpich:cray-mpich-abi:cray-mpich2
↪ :cray-netcdf:cray-netcdf-hdf5parallel:cray-paral
↪ lel-netcdf:cray-petsc:cray-petsc-complex:cray-sh
↪ mem:cray-tpsl:cray-trilinos:cudatoolkit:fftw:ga:
↪ hdf5:hdf5-parallel:iobuf:libfast:netcdf:netcdf-h
↪ df5parallel:ntk:onesided:papi:petsc:petsc-comple
↪ x:pmi:tpsl:trilinos:xt-libsci:xt-mpich2:xt-mpt:x
↪ t-papi
PE_NETCDF_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/
↪ pe/netcdf/4.6.1.3/@PRGENV@/@PE_NETCDF_DEFAULT_GE
↪ NCOMPS@/lib/pkgconfig
PE_PARALLEL_NETCDF_DEFAULT_VOLATILE_PKGCONFIG_PATH=/
↪ opt/cray/pe/parallel-netcdf/1.8.1.3/@PRGENV@/@PE
↪ _PARALLEL_NETCDF_DEFAULT_GENCOMPS@/lib/pkgconfig
PE_SMA_DEFAULT_COMPFLAG_GNU=-fcray-pointer
PE_TRILINOS_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cra
↪ y/pe/trilinos/12.12.1.1/@PRGENV@/@PE_TRILINOS_DE
↪ FAULT_GENCOMPS@/@PE_TRILINOS_DEFAULT_TARGET@/lib
↪ /pkgconfig
PE_ENV=INTEL
PE_HDF5_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_MPICH_ALTERNATE_LIBS_dpm=_dpm
PE_SMA_DEFAULT_COMPFLAG=
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
INTEL_LICENSE_FILE=28518@crayintel.licenses.nersc.go
↪ v:28518@intel.licenses.nersc.gov
SHELL=/bin/bash
TERM=xterm-256color
HOST=cori12
PE_TPSL_DEFAULT_GENCOMPS_CRAY_x86_skylake=86
PKGCONFIG_ENABLED=1
HISTSIZE=1000
PROFILEREAD=true
INTEL_MINOR_VERSION=18
PE_PETSC_DEFAULT_GENCOMPS_CRAY_sandybridge=86
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_x86_skylake=8.2 7.1
↪ 6.1
I_MPI_FABRICS=ofi
SSH_CLIENT=134.174.21.2 5709 22
CRAYPE_DIR=/opt/cray/pe/craype/2.5.15
CRAY_UGNI_POST_LINK_OPTS=-L/opt/cray/ugni/6.0.14.0-6
↪ .0.7.0_23.1__gea11d3d.ari/lib64
CRAY_XPMEM_POST_LINK_OPTS=-L/opt/cray/xpmem/2.2.15-6
↪ .0.7.1_5.11__g7549d06.ari/lib64
PE_NETCDF_DEFAULT_VOLATILE_PRGENV=GNU
PE_PARALLEL_NETCDF_DEFAULT_VOLATILE_PRGENV=GNU
PE_PETSC_DEFAULT_GENCOMPS_GNU_haswell=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_INTEL_haswell=160
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_x86_skylake=160
PE_TPSL_DEFAULT_GENCOMPS_GNU_sandybridge=82 71 53 49
PE_TPSL_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_LIBSCI
PE_TRILINOS_DEFAULT_VOLATILE_PRGENV=CRAY GNU INTEL
TMPDIR=/tmp
LIBRARY_PATH=/opt/intel/compilers_and_libraries_2018
↪ .1.163/linux/compiler/lib/intel64:/opt/intel/com
↪ pilers_and_libraries_2018.1.163/linux/mkl/lib/in
↪ tel64
PE_FFTW_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe
↪ /fftw/3.3.8.1/@PE_FFTW_DEFAULT_TARGET@/lib/pkgco
↪ nfig
PE_HDF5_DEFAULT_VOLATILE_PRGENV=GNU
PE_HDF5_PARALLEL_DEFAULT_VOLATILE_PKGCONFIG_PATH=/op
↪ t/cray/pe/hdf5-parallel/1.10.2.0/@PRGENV@/@PE_HD
↪ F5_PARALLEL_DEFAULT_GENCOMPS@/lib/pkgconfig
PE_NETCDF_HDF5PARALLEL_DEFAULT_VOLATILE_PKGCONFIG_PA
↪ TH=/opt/cray/pe/netcdf-hdf5parallel/4.6.1.3/@PRG
↪ ENV@/@PE_NETCDF_HDF5PARALLEL_DEFAULT_GENCOMPS@/l
↪ ib/pkgconfig
PE_PETSC_DEFAULT_GENCOMPS_CRAY_interlagos=86
CRAY_MPICH2_DIR=/opt/cray/pe/mpt/7.7.3/gni/mpich-int
↪ el/16.0
ALTD_SELECT_OFF_USERS=
ALT_LINKER=/usr/common/software/altd/2.0/bin/ld
SITE_MODULE_NAMES=darshan
INTEL_PATH=/opt/intel/compilers_and_libraries_2018.1
↪ .163
PE_GA_DEFAULT_VOLATILE_PRGENV=GNU
PE_LIBSCI_DEFAULT_GENCOMPS_GNU_x86_64=71 61 51 49
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_interlagos=8.6
PE_TPSL_DEFAULT_GENCOMPS_CRAY_mic_knl=86
ALTD_SELECT_ON=0
MORE=-sl
FPATH=:/opt/cray/pe/modules/3.2.10.6/init/sh_funcs/n
↪ o_redirect:/opt/cray/pe/modules/3.2.10.6/init/sh
↪ _funcs/no_redirect
PE_MPICH_DEFAULT_GENCOMPILERS_GNU=7.1 5.1 4.9
PE_PKGCONFIG_PRODUCTS=PE_MPICH:PE_LIBSCI
PE_TPSL_DEFAULT_GENCOMPS_INTEL_x86_64=160
PE_MPICH_GENCOMPS_GNU=71 51 49
I_MPI_PMI_LIBRARY=/usr/lib64/slurmpmi/libpmi.so
PE_PAPI_DEFAULT_ACCEL_LIBS_nvidia35=,-lcupti,-lcudar
↪ t,-lcuda
PE_PETSC_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_LIBSC
↪ I:PE_HDF5_PARALLEL:PE_TPSL
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_haswell=86
```

```
PE_TPSL_64_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/tpsl/18.06.1/@PRGENV@64/@PE_TPSL_64_DEFAULT_GENCOMPS@/@PE_TPSL_64_DEFAULT_TARGET@/lib/pkgconfig
ALTD_VERBOSE=0
PE_CRAY_DEFAULT_FIXED_PKGCONFIG_PATH=/opt/cray/pe/parallel-netcdf/1.8.1.3/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/netcdf-hdf5parallel/4.6.1.3/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/netcdf/4.6.1.3/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/hdf5-parallel/1.10.2.0/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/hdf5/1.10.2.0/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/ga/5.3.0.8/CRAY/8.6/lib/pkgconfig
PE_TRILINOS_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.7
SSH_TTY=/dev/pts/53
PE_LIBSCI_DEFAULT_OMP_REQUIRES_openmp=_mp
PE_PETSC_DEFAULT_GENCOMPS_CRAY_x86_64=86
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_sandybridge=8.6
PE_FORTRAN_PKGCONFIG_LIBS=mpichf90
PE_SMA_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/mpt/7.7.3/gni/sma@PE_SMA_DEFAULT_DIR_DEFAULT64@/lib64/pkgconfig
ALLINEA_QUEUE_DLL=/opt/cray/pe/mpt/7.7.3/gni/mpich-intel/16.0/lib/libtvmpich.so.3.0.1
PE_TRILINOS_DEFAULT_GENCOMPS_INTEL_x86_64=160
CRAY_MPICH_BASEDIR=/opt/cray/pe/mpt/7.7.3/gni
USER=USER
JRE_HOME=/usr/lib64/jvm/java/jre
PE_HDF5_PARALLEL_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_NETCDF_HDF5PARALLEL_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_x86_skylake=86
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_haswell=160
LD_LIBRARY_PATH=/opt/intel/compilers_and_libraries_2018.1.163/linux/mpi/intel64/lib:/opt/cray/job/2.3-6.0.7.0_44.1__g6c4e934.ari/lib64:/opt/intel/compilers_and_libraries_2018.1.163/linux/compiler/lib/intel64:/opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/lib/intel64:/usr/syscom/nsg/lib
LS_COLORS=no=00:fi=00:di=01;34:ln=00;36:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=41;33;01:ex=00;32:*.cmd=00;32:*.exe=01;32:*.com=01;32:*.bat=01;32:*.btm=01;32:*.dll=01;32:*.tar=00;31:*.tbz=00;31:*.tgz=00;31:*.rpm=00;31:*.deb=00;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.lzma=00;31:*.zip=00;31:*.zoo=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.tb2=00;31:*.tz2=00;31:*.tbz2=00;31:*.xz=00;31:*.avi=01;35:*.bmp=01;35:*.fli=01;35:*.gif=01;35:*.jpg=01;35:*.jpeg=01;35:*.mng=01;35:*.mov=01;35:*.mpg=01;35:*.pcx=01;35:*.pbm=01;35:*.pgm=01;35:*.png=01;35:*.ppm=01;35:*.tga=01;35:*.tif=01;35:*.xbm=01;35:*.xpm=01;35:*.dl=01;35:*.gl=01;35:*.wmv=01;35:*.aiff=00;32:*.au=00;32:*.mid=00;32:*.mp3=00;32:*.ogg=00;32:*.voc=00;32:*.wav=00;32:
PE_FFTW_DEFAULT_TARGET_interlagos=interlagos
PE_LIBSCI_DEFAULT_VOLATILE_PRGENV=CRAY GNU INTEL
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_interlagos=16.0
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_mic_knl=16.0
PE_TPSL_DEFAULT_GENCOMPS_CRAY_x86_64=86
PE_TRILINOS_DEFAULT_GENCOMPILERS_GNU_x86_64=8.2 7.3 5.1 4.9
PE_TRILINOS_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
CRAY_RCA_POST_LINK_OPTS=-L/opt/cray/rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari/lib64 -lrca
PE_LIBSCI_PKGCONFIG_VARIABLES=PE_LIBSCI_OMP_REQUIRES_@openmp@:PE_SCI_EXT_LIBPATH:PE_SCI_EXT_LIBNAME
PE_PETSC_DEFAULT_VOLATILE_PRGENV=CRAY CRAY64 GNU GNU64 INTEL INTEL64
PE_PKGCONFIG_LIBS=darshan-runtime:mpich:AtpSigHandler:cray-rca:libsci_mpi:libsci
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_sandybridge=8.2 7.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_haswell=16.0
PE_MPICH_FIXED_PRGENV=INTEL
CSHRCREAD=true
CSCRATCH=/global/cscratch1/sd/USER
XNLSPATH=/usr/share/X11/nls
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_mic_knl=8.6
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_skylake=16.0
PE_PETSC_DEFAULT_GENCOMPS_GNU_interlagos=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_GNU_sandybridge=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_INTEL_interlagos=160
PE_PETSC_DEFAULT_GENCOMPS_INTEL_sandybridge=160
PE_TPSL_DEFAULT_GENCOMPS_GNU_haswell=82 71 53 49
LDAPTLS_REQCERT=never
INTEL_VERSION=18.0.1.163
MPICH_ABORT_ON_ERROR=1
PE_LIBSCI_DEFAULT_GENCOMPS_CRAY_x86_64=86
PE_PAPI_DEFAULT_PKGCONFIG_VARIABLES=PE_PAPI_ACCEL_LIBS_@accelerator@
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_haswell=8.6
PE_PETSC_DEFAULT_GENCOMPS_GNU_mic_knl=53
```

```
PE_PETSC_DEFAULT_GENCOMPS_INTEL_mic_knl=160
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_interlagos=8.2
↪ 7.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_sandybridge=160
MPICH_DIR=/opt/cray/pe/mpt/7.7.3/gni/mpich-intel/16.0
ALTD_ON=1
HOSTTYPE=x86_64
ATP_POST_LINK_OPTS=-Wl,-L/opt/cray/pe/atp/2.1.3/libA
↪ pp/
CPATH=/opt/intel/compilers_and_libraries_2018.1.163/
↪ linux/mkl/include
PE_FFTW_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_FFTW_DEFAULT_TARGET_sandybridge=sandybridge
PE_HDF5_PARALLEL_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_NETCDF_HDF5PARALLEL_DEFAULT_REQUIRED_PRODUCTS=PE_
↪ HDF5_PARALLEL
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_sandybridge=16.0
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_haswell=8.6
PE_MPICH_FORTRAN_PKGCONFIG_LIBS=mpichf90
MPICH_MPIIO_DVS_MAXNODES=32
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_mic_knl=5.3
RCLOCAL_PRGENV=true
FROM_HEADER=
PE_LIBSCI_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
PE_LIBSCI_GENCOMPS_INTEL_x86_64=160
PE_PRODUCT_LIST=CRAYPE_MIC-KNL:CRAY_RCA:CRAY_ALPS:DV
↪ S:CRAY_XPMEM:CRAY_DMAPP:CRAY_PMI:CRAY_UGNI:CRAY_
↪ UDREG:CRAY_LIBSCI:CRAYPE:INTEL
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
PE_TPSL_DEFAULT_GENCOMPS_GNU_interlagos=82 71 53 49
PAGER=less
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_x86_64=7.1 5.3 4.9
PE_TPSL_DEFAULT_GENCOMPS_GNU_x86_skylake=82 71 61
CRAY_MPICH_ROOTDIR=/opt/cray/pe/mpt/7.7.3
CSHEDIT=emacs
PE_LIBSCI_GENCOMPILERS_GNU_x86_64=7.1 6.1 5.1 4.9
PE_PETSC_DEFAULT_GENCOMPS_GNU_skylake=61
PE_PETSC_DEFAULT_GENCOMPS_INTEL_skylake=160
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
PE_MPICH_GENCOMPILERS_CRAY=8.6
PE_MPICH_MODULE_NAME=cray-mpich
XDG_CONFIG_DIRS=/etc/xdg
INTEL_MAJOR_VERSION=18
PE_LIBSCI_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
PE_LIBSCI_GENCOMPS_CRAY_x86_64=86
PE_MPICH_DEFAULT_VOLATILE_PRGENV=CRAY GNU
PE_MPICH_TARGET_VAR_nvidia20=-lcudart
PE_TPSL_64_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_LIB
↪ SCI
PE_TPSL_DEFAULT_GENCOMPS_CRAY_haswell=86
PE_TPSL_DEFAULT_GENCOMPS_CRAY_sandybridge=86
MINICOM=-c on
LIBGL_DEBUG=quiet
USERMODULES=PrgEnv-cray:PrgEnv-gnu:PrgEnv-intel:PrgE
↪ nv-pathscale:PrgEnv-pgi:acml:alps:apprentice:app
↪ rentice2:atp:blcr:cce:chapel:cray-ccdb:cray-fftw
↪ :cray-ga:cray-hdf5:cray-hdf5-parallel:cray-lgdb:
↪ cray-libsci:cray-libsci_acc:cray-mpich:cray-mpic
↪ h-compat:cray-mpich2:cray-netcdf:cray-netcdf-hdf
↪ 5parallel:cray-parallel-netcdf:cray-petsc:cray-p
↪ etsc-complex:cray-shmem:cray-snplauncher:cray-tp
↪ sl:cray-trilinos:craypat:craype:craypkg-gen:cuda
↪ toolkit:ddt:fftw:ga:gcc:hdf5:hdf5-parallel:intel
↪ :iobuf:java:lgdb:libfast:libsci_acc:mpich1:netcd
↪ f:netcdf-hdf5parallel:netcdf-nofsync:netcdf-nofs
↪ ync-hdf5parallel:ntk:onesided:papi:parallel-netc
↪ df:pathscale:perftools:perftools-lite:petsc:pets
↪ c-complex:pgi:pmi:stat:totalview:tpsl:trilinos:x
↪ t-asyncpe:xt-craypat:xt-lgdb:xt-libsci:xt-mpich2
↪ :xt-mpt:xt-papi:xt-shmem:xt-totalview
CRAY_DMAPP_INCLUDE_OPTS=-I/opt/cray/dmapp/7.1.1-6.0.
↪ 7.0_34.3__g5a674e0.ari/include
↪ -I/opt/cray/gni-headers/5.0.12.0-6.0.7.0_24.1__g
↪ 3b1768f.ari/include
CRAY_LIBSCI_BASE_DIR=/opt/cray/pe/libsci/18.07.1
CRAY_LIBSCI_DIR=/opt/cray/pe/libsci/18.07.1
DVS_VERSION=0.9.0
NLSPATH=/opt/intel/compilers_and_libraries_2018.1.16
↪ 3/linux/compiler/lib/intel64/locale/%l_%t/%N:/op
↪ t/intel/compilers_and_libraries_2018.1.163/linux
↪ /mkl/lib/intel64/locale/%l_%t/%N
PE_LIBSCI_PKGCONFIG_LIBS=libsci_mpi:libsci
PE_NETCDF_DEFAULT_GENCOMPS_GNU=
PE_PARALLEL_NETCDF_DEFAULT_GENCOMPS_GNU=51 49
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_mic_knl=71 53
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_x86_64=82 71 53 49
PATH=/opt/intel/compilers_and_libraries_2018.1.163/l
↪ inux/mpi/intel64/bin:/usr/common/software/darsha
↪ n/3.1.4/bin:/usr/common/software/altd/2.0/bin:/u
↪ sr/common/software/bin:/usr/common/mss/bin:/usr/
↪ common/nsg/bin:/opt/cray/pe/mpt/7.7.3/gni/bin:/o
↪ pt/cray/rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari/bi
↪ n:/opt/cray/alps/6.6.43-6.0.7.0_26.4__ga796da3.a
↪ ri/sbin:/opt/cray/job/2.2.3-6.0.7.0_44.1__g6c4e9
↪ 34.ari/bin:/opt/cray/pe/craype/2.5.15/bin:/opt/i
↪ ntel/compilers_and_libraries_2018.1.163/linux/bi
↪ n/intel64:/opt/ovis/bin:/opt/ovis/sbin:/usr/sysc
↪ om/nsg/sbin:/usr/syscom/nsg/bin:/opt/cray/pe/mod
↪ ules/3.2.10.6/bin:/usr/local/bin:/usr/bin:/bin:/
↪ usr/bin/X11:/usr/games:/usr/lib/mit/bin:/usr/lib
↪ /mit/sbin:/opt/cray/pe/bin
MAIL=/var/mail/USER
MODULE_VERSION=3.2.10.6
PE_HDF5_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe
↪ /hdf5/1.10.2.0/@PRGENV@/@PE_HDF5_DEFAULT_GENCOMP
↪ S@/lib/pkgconfig
PE_PKGCONFIG_DEFAULT_PRODUCTS=PE_TRILINOS:PE_TPSL_64
↪ :PE_TPSL:PE_PETSC:PE_PARALLEL_NETCDF:PE_NETCDF_H
↪ DF5PARALLEL:PE_NETCDF:PE_MPICH:PE_LIBSCI:PE_HDF5
↪ _PARALLEL:PE_HDF5:PE_GA:PE_FFTW2:PE_FFTW
```

```
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_x86_64=8.2 7.1 5.3
↪  4.9
PE_TPSL_DEFAULT_GENCOMPS_CRAY_interlagos=86
PE_MPICH_GENCOMPILERS_GNU=7.1 5.1 4.9
LIBGL_ALWAYS_INDIRECT=1
I_MPI_OFI_LIBRARY=/global/common/software/m3169/ofi-
↪  libfabric/1.6.2/lib/libfabric.so
CPU=x86_64
XTPE_NETWORK_TARGET=aries
ATP_IGNORE_SIGTERM=1
PE_FFTW_DEFAULT_TARGET_abudhabi=abudhabi
PE_NETCDF_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_PARALLEL_NETCDF_DEFAULT_GENCOMPILERS_GNU=5.1 4.9
PE_PETSC_DEFAULT_GENCOMPS_CRAY_mic_knl=86
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_x86_skylake=8.2
↪  7.1 6.1
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_haswell=8.2 7.1 5.3
↪  4.9
_=/usr/bin/env
JAVA_BINDIR=/usr/lib64/jvm/java/bin
LDMSD_PLUGIN_LIBPATH=/opt/ovis/lib64/ovis-ldms
PE_HDF5_PARALLEL_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_HDF5_PARALLEL_DEFAULT_GENCOMPS_GNU=
PE_NETCDF_HDF5PARALLEL_DEFAULT_FIXED_PRGENV=CRAY
↪  INTEL
PE_NETCDF_HDF5PARALLEL_DEFAULT_GENCOMPS_GNU=
PE_SMA_DEFAULT_DIR_CRAY_DEFAULT64=64
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_x86_skylake=8.6
CRAY_UDREG_POST_LINK_OPTS=-L/opt/cray/udreg/2.3.2-6.
↪  0.7.0_33.18__g5196236.ari/lib64
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_sandybridge=86
PE_TPSL_64_DEFAULT_VOLATILE_PRGENV=CRAY CRAY64 GNU
↪  GNU64 INTEL INTEL64
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_mic_knl=8.6
PE_TPSL_DEFAULT_GENCOMPS_INTEL_interlagos=160
PWD=/global/homes/w/USER
INPUTRC=/etc/inputrc
CRAYPE_VERSION=2.5.15
CRAY_ALPS_POST_LINK_OPTS=-L/opt/cray/alps/6.6.43-6.0
↪  .7.0_26.4__ga796da3.ari/lib64
PE_TPSL_DEFAULT_GENCOMPS_GNU_mic_knl=71 53
PE_MPICH_VOLATILE_PRGENV=CRAY GNU
I_MPI_FALLBACK=1
JAVA_HOME=/usr/lib64/jvm/java
TARGETMODULES=craype-abudhabi:craype-abudhabi-cu:cra
↪  ype-accel-host:craype-accel-nvidia20:craype-acce
↪  l-nvidia30:craype-accel-nvidia35:craype-barcelon
↪  a:craype-broadwell:craype-haswell:craype-hugepag
↪  es128K:craype-hugepages128M:craype-hugepages16M:
↪  craype-hugepages256M:craype-hugepages2M:craype-h
↪  ugepages32M:craype-hugepages4M:craype-hugepages5
↪  12K:craype-hugepages512M:craype-hugepages64M:cra
↪  ype-hugepages8M:craype-intel-knc:craype-interlag
↪  os:craype-interlagos-cu:craype-istanbul:craype-i
↪  vybridge:craype-mc12:craype-mc8:craype-mic-knl:c
↪  raype-network-aries:craype-network-gemini:craype
↪  -network-infiniband:craype-network-none:craype-ne
↪  twork-seastar:craype-sandybridge:craype-shanghai
↪  :craype-target-compute_node:craype-target-local_
↪  host:craype-target-native:craype-xeon:xtpe-barce
↪  lona:xtpe-interlagos:xtpe-interlagos-cu:xtpe-ist
↪  anbul:xtpe-mc12:xtpe-mc8:xtpe-network-gemini:xtp
↪  e-network-seastar:xtpe-shanghai:xtpe-target-nati
↪  ve:xtpe-xeon
_LMFILES_=/opt/cray/pe/modulefiles/modules/3.2.10.6:
↪  /usr/syscom/nsg/modulefiles/nsg/1.2.0:/opt/modul
↪  efiles/intel/18.0.1.163:/opt/cray/pe/craype/2.5.
↪  15/modulefiles/craype-network-aries:/opt/cray/pe
↪  /modulefiles/craype/2.5.15:/opt/cray/pe/modulefi
↪  les/cray-libsci/18.07.1:/opt/cray/ari/modulefile
↪  s/udreg/2.3.2-6.0.7.0_33.18__g5196236.ari:/opt/c
↪  ray/ari/modulefiles/ugni/6.0.14.0-6.0.7.0_23.1__
↪  gea11d3d.ari:/opt/cray/pe/modulefiles/pmi/5.0.14
↪  :/opt/cray/ari/modulefiles/dmapp/7.1.1-6.0.7.0_3
↪  4.3__g5a674e0.ari:/opt/cray/ari/modulefiles/gni-
↪  headers/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari:/opt
↪  /cray/ari/modulefiles/xpmem/2.2.15-6.0.7.1_5.11_
↪  _g7549d06.ari:/opt/cray/ari/modulefiles/job/2.2.
↪  3-6.0.7.0_44.1__g6c4e934.ari:/opt/cray/ari/modul
↪  efiles/dvs/2.7_2.2.117-6.0.7.1_9.2__gf817677:/op
↪  t/cray/ari/modulefiles/alps/6.6.43-6.0.7.0_26.4_
↪  _ga796da3.ari:/opt/cray/ari/modulefiles/rca/2.2.
↪  18-6.0.7.0_33.3__g2aa4f39.ari:/opt/cray/pe/modul
↪  efiles/atp/2.1.3:/opt/cray/pe/modulefiles/PrgEnv
↪  -intel/6.0.4:/opt/cray/pe/craype/2.5.15/modulefil
↪  es/craype-mic-knl:/opt/cray/pe/modulefiles/cray-
↪  mpich/7.7.3:/usr/common/software/modulefiles/alt
↪  d/2.0:/usr/common/software/modulefiles/darshan/3
↪  .1.4:/usr/common/software/modulefiles/impi/2018.
↪  up1
PE_LIBSCI_DEFAULT_OMP_REQUIRES=
PE_MPICH_DEFAULT_GENCOMPS_CRAY=86
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_sandybridge=7.1
↪  5.3 4.9
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_haswell=16.0
PE_LIBSCI_MODULE_NAME=cray-libsci/18.07.1
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_skylake=8.6
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_interlagos=8.6
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_mic_knl=7.1 5.3
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_x86_skylake=82 71 61
```

```
PE_INTEL_FIXED_PKGCONFIG_PATH=/opt/cray/pe/mpt/7.7.3
↪ /gni/mpich-intel/16.0/lib/pkgconfig
MODULEPATH=/opt/cray/pe/craype/2.5.15/modulefiles:/o
↪ pt/cray/pe/modulefiles:/opt/cray/modulefiles:/op
↪ t/modulefiles:/usr/common/software/modulefiles:/
↪ usr/syscom/nsg/modulefiles:/usr/syscom/nsg/opt/m
↪ odulefiles:/usr/common/das/modulefiles:/usr/comm
↪ on/ftg/modulefiles:/opt/cray/craype/default/modu
↪ lefiles:/opt/cray/ari/modulefiles
PYTHONSTARTUP=/etc/pythonstart
PE_LIBSCI_GENCOMPILERS_CRAY_x86_64=8.6
PE_MPICH_NV_LIBS_nvidia20=-lcudart
PE_MPICH_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/mpt/7.
↪ 7.3/gni/mpich-@PRGENV@@PE_MPICH_DIR_DEFAULT64@/@
↪ PE_MPICH_GENCOMPS@/lib/pkgconfig
SDK_HOME=/usr/lib64/jvm/java
TZ=US/Pacific
LOADEDMODULES=modules/3.2.10.6:nsg/1.2.0:intel/18.0.
↪ 1.163:craype-network-aries:craype/2.5.15:cray-li
↪ bsci/18.07.1:udreg/2.3.2-6.0.7.0_33.18__g5196236
↪ .ari:ugni/6.0.14.0-6.0.7.0_23.1__gea11d3d.ari:pm
↪ i/5.0.14:dmapp/7.1.1-6.0.7.0_34.3__g5a674e0.ari:
↪ gni-headers/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari:
↪ xpmem/2.2.15-6.0.7.1_5.11__g7549d06.ari:job/2.2.
↪ 3-6.0.7.0_44.1__g6c4e934.ari:dvs/2.7_2.2.117-6.0
↪ .7.1_9.2__gf817677:alps/6.6.43-6.0.7.0_26.4__ga7
↪ 96da3.ari:rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari:
↪ atp/2.1.3:PrgEnv-intel/6.0.4:craype-mic-knl:cray
↪ -mpich/7.7.3:altd/2.0:darshan/3.1.4:impi/2018.up1
NSG_HOME=/usr/syscom/nsg
SHMEM_ABORT_ON_ERROR=1
MAN_POSIXLY_CORRECT=1
CRAY_DMAPP_POST_LINK_OPTS=-L/opt/cray/dmapp/7.1.1-6.
↪ 0.7.0_34.3__g5a674e0.ari/lib64
PE_FFTW_DEFAULT_TARGET_ivybridge=ivybridge
PE_FFTW_DEFAULT_TARGET_share=share
PE_FFTW_DEFAULT_TARGET_x86_skylake=x86_skylake
PE_PKG_CONFIG_PATH=/opt/cray/pe/cti/1.0.7/lib/pkgcon
↪ fig:/opt/cray/pe/cti/1.0.6/lib/pkgconfig
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_interlagos=82 71 53
↪ 49
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_mic_knl=16.0
CRAY_RCA_INCLUDE_OPTS=-I/opt/cray/rca/2.2.18-6.0.7.0
↪ _33.3__g2aa4f39.ari/include
↪ -I/opt/cray/krca/2.2.4-6.0.7.1_5.42__g8505b97.ar
↪ i/include
↪ -I/opt/cray-hss-devel/8.0.0/include
PE_LIBSCI_OMP_REQUIRES_openmp=_mp
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_skylake=6.1
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_x86_skylake=8.6
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_mic_knl=86
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
CRAY_MPICH_DIR=/opt/cray/pe/mpt/7.7.3/gni/mpich-inte
↪ l/16.0
PE_MPICH_CXX_PKGCONFIG_LIBS=mpichcxx
PE_LIBSCI_DEFAULT_GENCOMPS_INTEL_x86_64=160
PE_MPICH_PKGCONFIG_VARIABLES=PE_MPICH_NV_LIBS_@accel
↪ erator@:PE_MPICH_ALTERNATE_LIBS_@multithreaded@:
↪ PE_MPICH_ALTERNATE_LIBS_@dpm@
CRAY_PMI_POST_LINK_OPTS=-L/opt/cray/pe/pmi/5.0.14/li
↪ b64
PE_HDF5_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_mic_knl=8.6
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_x86_skylake=16.0
PE_TPSL_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe
↪ /tpsl/18.06.1/@PRGENV@/@PE_TPSL_DEFAULT_GENCOMPS
↪ @/@PE_TPSL_DEFAULT_TARGET@/lib/pkgconfig
CRAY_MPICH2_VER=7.7.3
PE_MPICH_PKGCONFIG_LIBS=mpich
GPG_TTY=/dev/pts/53
PE_GA_DEFAULT_GENCOMPILERS_GNU=5.3 4.9
PE_LIBSCI_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/libsc
↪ i/18.07.1/@PRGENV@/@PE_LIBSCI_GENCOMPS@/@PE_LIBS
↪ CI_TARGET@/lib/pkgconfig
PE_MPICH_ALTERNATE_LIBS_multithreaded=_mt
PE_NETCDF_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_PARALLEL_NETCDF_DEFAULT_FIXED_PRGENV=CRAY INTEL
HOME=/global/homes/w/USER
SHLVL=2
JDK_HOME=/usr/lib64/jvm/java
QT_SYSTEM_DIR=/usr/share/desktop-data
CRAY_LIBSCI_VERSION=18.07.1
PE_HDF5_PARALLEL_DEFAULT_VOLATILE_PRGENV=GNU
PE_MPICH_TARGET_VAR_nvidia35=-lcudart
PE_NETCDF_HDF5PARALLEL_DEFAULT_VOLATILE_PRGENV=GNU
PE_PKGCONFIG_PRODUCTS_DEFAULT=PE_PAPI
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_haswell=82 71 53 49
OSTYPE=linux
LESS_ADVANCED_PREPROCESSOR=no
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_interlagos=16.0
PE_MPICH_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/p
↪ e/mpt/7.7.3/gni/mpich-@PRGENV@@PE_MPICH_DEFAULT_
↪ DIR_DEFAULT64@/@PE_MPICH_DEFAULT_GENCOMPS@/lib/p
↪ kgconfig
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_interlagos=8.6
PE_TPSL_DEFAULT_VOLATILE_PRGENV=CRAY CRAY64 GNU GNU64
↪ INTEL INTEL64
ALTD_PATH=/usr/common/software/altd/2.0
LS_OPTIONS=-N --color=none -T 0
CRAY_PMI_INCLUDE_OPTS=-I/opt/cray/pe/pmi/5.0.14/incl
↪ ude
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_interlagos=86
PE_TPSL_DEFAULT_GENCOMPS_INTEL_sandybridge=160
PRGENVMODULES=PrgEnv-cray:PrgEnv-gnu:PrgEnv-intel:Pr
↪ gEnv-pathscale:PrgEnv-pgi
WINDOWMANAGER=
CRAYPE_NETWORK_TARGET=aries
ATP_MRNET_COMM_PATH=/opt/cray/pe/atp/2.1.3/libexec/a
↪ tp_mrnet_commnode_wrapper
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_haswell=8.6
PKG_CONFIG_PATH_DEFAULT=/opt/cray/pe/papi/5.6.0.3/li
↪ b64/pkgconfig
```

```
PE_MPICH_DIR_CRAY_DEFAULT64=64
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_haswell=7.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_mic_knl=7.1 5.3
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_interlagos=8.2 7.1
↪ 5.3 4.9
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_sandybridge=16.0
BASH_ENV=/global/homes/w/USER/.bashrc
LOGNAME=USER
MACHTYPE=x86_64-suse-linux
LESS=-M -I -R
PYTHONPATH=/opt/ovis/lib/python2.7/site-packages
CRAY_GNI_HEADERS_INCLUDE_OPTS=-I/opt/cray/gni-header┘
↪ s/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari/include
CRAY_LIBSCI_PREFIX_DIR=/opt/cray/pe/libsci/18.07.1/I┘
↪ NTEL/16.0/x86_64
PE_HDF5_DEFAULT_GENCOMPS_GNU=
PE_MPICH_NV_LIBS=
PE_NETCDF_DEFAULT_REQUIRED_PRODUCTS=PE_HDF5
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_haswell=8.2 7.1
↪ 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_sandybridge=16┘
↪ .0
PE_TPSL_DEFAULT_GENCOMPS_GNU_x86_64=82 71 53 49
PE_TRILINOS_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_HD┘
↪ F5_PARALLEL:PE_NETCDF_HDF5PARALLEL:PE_LIBSCI:PE_┘
↪ TPSL
ALTD_SELECT_USERS=
I_MPI_OFI_PROVIDER=gni
CVS_RSH=ssh
DMAPP_ABORT_ON_ERROR=1
PE_LIBSCI_OMP_REQUIRES=
PE_MPICH_DEFAULT_GENCOMPILERS_CRAY=8.6
PE_TRILINOS_DEFAULT_GENCOMPS_GNU_x86_64=82 73 51 49
PE_MPICH_GENCOMPS_CRAY=86
SSH_CONNECTION=134.174.21.2 5709 128.55.209.23 22
TOOLMODULES=apprentice:apprentice2:atp:chapel:cray-l┘
↪ gdb:cray-snplauncher:craypat:craypkg-gen:ddt:gdb┘
↪ :iobuf:papi:perftools:perftools-lite:stat:totalv┘
↪ iew:xt-craypat:xt-lgdb:xt-papi:xt-totalview
XDG_DATA_DIRS=/usr/share
DVS_INCLUDE_OPTS=-I/opt/cray/dvs/2.7_2.2.117-6.0.7.1┘
↪ _9.2__gf817677/include
PE_LIBSCI_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_MPICH_DEFAULT_FIXED_PRGENV=INTEL
PE_MPICH_DEFAULT_GENCOMPS_GNU=71 51 49
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_interlagos=16.0
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_sandybridge=8.6
MODULESHOME=/opt/cray/pe/modules/3.2.10.6
CRAY_PRGENVINTEL=loaded
PE_FFTW2_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_GA_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_LIBSCI_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/┘
↪ pe/libsci/18.07.1/@PRGENV@/@PE_LIBSCI_DEFAULT_GE┘
↪ NCOMPS@/@PE_LIBSCI_DEFAULT_TARGET@/lib/pkgconfig
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_sandybridge=8.2 7.1
↪ 5.3 4.9
LESSOPEN=lessopen.sh %s
PE_MPICH_NV_LIBS_nvidia35=-lcudart
PE_PETSC_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/p┘
↪ e/petsc/3.8.4.0/complex/@PRGENV@/@PE_PETSC_DEFAU┘
↪ LT_GENCOMPS@/@PE_PETSC_DEFAULT_TARGET@/lib/pkgco┘
↪ nfig
PKG_CONFIG_PATH=/usr/common/software/darshan/3.1.4/l┘
↪ ib/pkgconfig:/opt/cray/rca/2.2.18-6.0.7.0_33.3__┘
↪ g2aa4f39.ari/lib64/pkgconfig:/opt/cray/alps/6.6.┘
↪ 43-6.0.7.0_26.4__ga796da3.ari/lib64/pkgconfig:/o┘
↪ pt/cray/xpmem/2.2.15-6.0.7.1_5.11__g7549d06.ari/┘
↪ lib64/pkgconfig:/opt/cray/gni-headers/5.0.12.0-6┘
↪ .0.7.0_24.1__g3b1768f.ari/lib64/pkgconfig:/opt/c┘
↪ ray/dmapp/7.1.1-6.0.7.0_34.3__g5a674e0.ari/lib64┘
↪ /pkgconfig:/opt/cray/pe/pmi/5.0.14/lib64/pkgconf┘
↪ ig:/opt/cray/ugni/6.0.14.0-6.0.7.0_23.1__gea11d3┘
↪ d.ari/lib64/pkgconfig:/opt/cray/udreg/2.3.2-6.0.┘
↪ 7.0_33.18__g5196236.ari/lib64/pkgconfig:/opt/cra┘
↪ y/pe/craype/2.5.15/pkg-config:/opt/cray/pe/iobuf┘
↪ /2.0.8/lib/pkgconfig:/opt/cray/pe/fftw/2.1.5.9/l┘
↪ ib/pkgconfig:/opt/cray/pe/atp/2.1.3/lib/pkgconfig
PELOCAL_PRGENV=true
LIBSCI_BASE_DIR=/opt/cray/pe/libsci/18.07.1
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_x86_64=160
LIBSCI_VERSION=18.07.1
PE_LIBSCI_DEFAULT_PKGCONFIG_VARIABLES=PE_LIBSCI_DEFA┘
↪ ULT_OMP_REQUIRES_@openmp@:PE_SCI_EXT_LIBPATH:PE_┘
↪ SCI_EXT_LIBNAME
PE_MPICH_NV_LIBS_nvidia60=-lcudart
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_sandybridge=82 71 53
↪ 49
PE_TPSL_DEFAULT_GENCOMPS_INTEL_mic_knl=160
DISPLAY=cori12:16.0
XDG_RUNTIME_DIR=/run/user/71130
NERSC_HOST=cori
CRAY_PRE_COMPILE_OPTS=-hnetwork=aries
CRAY_ALPS_INCLUDE_OPTS=-I/opt/cray/alps/6.6.43-6.0.7┘
↪ .0_26.4__ga796da3.ari/include
PE_FFTW_DEFAULT_TARGET_broadwell=broadwell
PE_LIBSCI_GENCOMPILERS_INTEL_x86_64=16.0
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_x86_64=8.2 7.1
↪ 5.3 4.9
CRAY_CPU_TARGET=mic-knl
CRAY_UGNI_INCLUDE_OPTS=-I/opt/cray/ugni/6.0.14.0-6.0┘
↪ .7.0_23.1__gea11d3d.ari/include
CRAY_XPMEM_INCLUDE_OPTS=-I/opt/cray/xpmem/2.2.15-6.0┘
↪ .7.1_5.11__g7549d06.ari/include
PE_LIBSCI_REQUIRED_PRODUCTS=PE_MPICH
PE_PAPI_DEFAULT_ACCELL_FAMILY_LIBS=
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_x86_64=86
craype_already_loaded=0
PE_LIBSCI_DEFAULT_GENCOMPILERS_GNU_x86_64=7.1 6.1 5.1
↪ 4.9
PE_LIBSCI_GENCOMPS_GNU_x86_64=71 61 51 49
PE_TPSL_DEFAULT_GENCOMPS_INTEL_haswell=160
LESSCLOSE=lessclose.sh %s %s
```

```
ATP_HOME=/opt/cray/pe/atp/2.1.3
PE_FFTW_DEFAULT_TARGET_x86_64=x86_64
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
SCRATCH=/global/cscratch1/sd/USER
G_BROKEN_FILENAMES=1
CRAY_LD_LIBRARY_PATH=/usr/common/software/darshan/3.
↪ 1.4/lib:/opt/cray/pe/mpt/7.7.3/gni/mpich-intel/1
↪ 6.0/lib:/opt/cray/rca/2.2.18-6.0.7.0_33.3__g2aa4
↪ f39.ari/lib64:/opt/cray/alps/6.6.43-6.0.7.0_26.4
↪ __ga796da3.ari/lib64:/opt/cray/xpmem/2.2.15-6.0.
↪ 7.1_5.11__g7549d06.ari/lib64:/opt/cray/dmapp/7.1
↪ .1-6.0.7.0_34.3__g5a674e0.ari/lib64:/opt/cray/pe
↪ /pmi/5.0.14/lib64:/opt/cray/ugni/6.0.14.0-6.0.7.
↪ 0_23.1__gea11d3d.ari/lib64:/opt/cray/udreg/2.3.2
↪ -6.0.7.0_33.18__g5196236.ari/lib64:/opt/cray/pe/l
↪ ibsci/18.07.1/INTEL/16.0/x86_64/lib
PE_FFTW_DEFAULT_TARGET_haswell=haswell
PE_GA_DEFAULT_GENCOMPS_GNU=53 49
PE_GA_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/g
↪ a/5.3.0.8/@PRGENV@/@PE_GA_DEFAULT_GENCOMPS@/lib/
↪ pkgconfig
PE_INTEL_DEFAULT_FIXED_PKGCONFIG_PATH=/opt/cray/pe/p
↪ arallel-netcdf/1.8.1.3/INTEL/16.0/lib/pkgconfig:
↪ /opt/cray/pe/netcdf-hdf5parallel/4.6.1.3/INTEL/1
↪ 6.0/lib/pkgconfig:/opt/cray/pe/netcdf/4.6.1.3/IN
↪ TEL/16.0/lib/pkgconfig:/opt/cray/pe/mpt/7.7.3/gn
↪ i/mpich-intel/16.0/lib/pkgconfig:/opt/cray/pe/hd
↪ f5-parallel/1.10.2.0/INTEL/16.0/lib/pkgconfig:/o
↪ pt/cray/pe/hdf5/1.10.2.0/INTEL/16.0/lib/pkgconfi
↪ g:/opt/cray/pe/ga/5.3.0.8/INTEL/18.0/lib/pkgconf
↪ ig
PE_PAPI_DEFAULT_ACCEL_LIBS=
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_interlagos=7.1 5.3
↪ 4.9
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_haswell=16.0
PE_SMA_DEFAULT_DIR_PGI_DEFAULT64=64
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_x86_skylake=16
↪ .0
ALTD_WORKDIR=/global/cscratch1/altd/logs
COLORTERM=1
JAVA_ROOT=/usr/lib64/jvm/java
PE_MPICH_DEFAULT_DIR_CRAY_DEFAULT64=64
PE_PETSC_DEFAULT_GENCOMPS_CRAY_haswell=86
PE_PETSC_DEFAULT_GENCOMPS_GNU_x86_64=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_INTEL_x86_64=160
intel_already_loaded=0
I_MPI_ROOT=/opt/intel/compilers_and_libraries_2018.1
↪ .163/linux/mpi
BASH_FUNC_module%%=() {  eval
↪ `/opt/cray/pe/modules/3.2.10.6/bin/modulecmd
↪ bash $*`
}
+ lsb_release -a
LSB Version:        n/a
Distributor ID:        SUSE
Description:        SUSE Linux Enterprise Server 12
↪ SP3
Release:        12.3
Codename:        n/a
+ uname -a
Linux cori12 4.4.162-94.72-default #1 SMP Mon Nov 12
↪ 18:57:45 UTC 2018 (9de753f) x86_64 x86_64 x86_64
↪ GNU/Linux
+ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2698 v3
↪ @ 2.30GHz
Stepping:              2
CPU MHz:               3114.459
CPU max MHz:           3600.0000
CPU min MHz:           1200.0000
BogoMIPS:              4599.54
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              40960K
NUMA node0 CPU(s):
↪ 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34
↪ ,36,38,40,42,44,46,48,50,52,54,56,58,60,62
NUMA node1 CPU(s):
↪ 1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35
↪ ,37,39,41,43,45,47,49,51,53,55,57,59,61,63
Flags:                 fpu vme de pse tsc msr pae mce
↪ cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪ dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪ pdpe1gb rdtscp lm ibrs flush_l1d constant_tsc
↪ arch_perfmon pebs bts rep_good nopl xtopology
↪ nonstop_tsc aperfmperf eagerfpu pni pclmulqdq
↪ dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg
↪ fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic
↪ movbe popcnt tsc_deadline_timer aes xsave avx
↪ f16c rdrand lahf_lm abm ida arat epb
↪ invpcid_single pln pts dtherm ssbd ibpb stibp
↪ kaiser tpr_shadow vnmi flexpriority ept vpid
↪ fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
↪ invpcid cqm xsaveopt cqm_llc cqm_occup_llc
+ cat /proc/meminfo
MemTotal:       527508144 kB
MemFree:        458551364 kB
```

```
MemAvailable:   456521136 kB
Buffers:         3598880 kB
Cached:         18233820 kB
SwapCached:        71436 kB
Active:         14565564 kB
Inactive:       10904672 kB
Active(anon):    3599260 kB
Inactive(anon):   805088 kB
Active(file):   10966304 kB
Inactive(file): 10099584 kB
Unevictable:     8388688 kB
Mlocked:         8388688 kB
SwapTotal:      33554428 kB
SwapFree:       30743124 kB
Dirty:              1172 kB
Writeback:             0 kB
AnonPages:      11987040 kB
Mapped:          1503364 kB
Shmem:            766792 kB
Slab:           28110080 kB
SReclaimable:    6952468 kB
SUnreclaim:     21157612 kB
KernelStack:       51424 kB
PageTables:       115572 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    297308500 kB
Committed_AS:   22974896 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:   9697280 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:    58489596 kB
DirectMap2M:    464654336 kB
DirectMap1G:    15728640 kB
+ inxi -F -c0
./collect_environment.sh: line 14: inxi: command not
↪  found
+ lsblk -a
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0   2.6G  0 loop
loop1    7:1    0  23.8G  0 loop
↪  /var/opt/cray/imps-image-binding/PE_x86_64/squas┘
↪  h_mounts/squashfs_756IR3_mount_point
loop2    7:2    0         1 loop
loop3    7:3    0         1 loop
loop4    7:4    0         1 loop
loop5    7:5    0         1 loop
loop6    7:6    0         1 loop
loop7    7:7    0         1 loop
loop8    7:8    0         1 loop
loop9    7:9    0         1 loop
loop10   7:10   0         1 loop
loop11   7:11   0         1 loop
loop12   7:12   0         1 loop
loop13   7:13   0         1 loop
loop14   7:14   0         1 loop
loop15   7:15   0         1 loop
loop16   7:16   0         1 loop
loop17   7:17   0         1 loop
sda      8:0    0   200G  0 disk
|-sda1   8:1    0     9M  0 part
|-sda2   8:2    0     2G  0 part /boot
|-sda3   8:3    0    32G  0 part
|-sda4   8:4    0    32G  0 part /tmp
|-sda5   8:5    0    32G  0 part [SWAP]
`-sda6   8:6    0    24G  0 part /var/crash
sdb      8:16   0 917.3G  0 disk
`-sdb1   8:17   0 917.3G  0 part
↪  /var/opt/cray/persistent
sr0     11:0    1  1024M  0 rom
+ lsscsi -s
[0:2:0:0]    disk    DELL     PERC H730 Mini   4.25
↪  /dev/sda    214GB
[0:2:1:0]    disk    DELL     PERC H730 Mini   4.25
↪  /dev/sdb    984GB
[10:0:0:0]   cd/dvd  PLDS     DVD+-RW DS-8ABSH AD51
↪  /dev/sr0        -
+ module list
++ /opt/cray/pe/modules/3.2.10.6/bin/modulecmd bash
↪  list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6
  ↪  13) job/2.2.3-6.0.7.0_44.1__g6c4e934.ari
  2) nsg/1.2.0
  ↪  14) dvs/2.7_2.2.117-6.0.7.1_9.2__gf817677
  3) intel/18.0.1.163
  ↪  15) alps/6.6.43-6.0.7.0_26.4__ga796da3.ari
  4) craype-network-aries
  ↪  16) rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari
  5) craype/2.5.15
  ↪  17) atp/2.1.3
  6) cray-libsci/18.07.1
  ↪  18) PrgEnv-intel/6.0.4
  7) udreg/2.3.2-6.0.7.0_33.18__g5196236.ari
  ↪  19) craype-mic-knl
  8) ugni/6.0.14.0-6.0.7.0_23.1__gea11d3d.ari
  ↪  20) cray-mpich/7.7.3
  9) pmi/5.0.14
  ↪  21) altd/2.0
 10) dmapp/7.1.1-6.0.7.0_34.3__g5a674e0.ari
 ↪  22) darshan/3.1.4
 11) gni-headers/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari
 ↪  23) impi/2018.up1
```

```
 12) xpmem/2.2.15-6.0.7.1_5.11__g7549d06.ari
+ eval
+ nvidia-smi
```

ARTIFACT EVALUATION

Verification and validation studies: The quality of image reconstruction is validated in two ways: (1) convergence comparison with a fully converged gold standard reconstruction, and (2) subjective image quality evaluation by authors and imaging experts. The normalized root-mean-square-error (NRMSE) of all reconstruction results are compared with a fully converged gold standard reconstruction, and all results reported in tables 1 and 2 have a NRMSE less than 2%. Therefore, the computational results are trustworthy in terms of NRMSE. In addition, reconstruction results are subjectively evaluated by authors of this paper and other imaging experts. The consensus among experts who judge our image quality is "good image quality at super-resolution".

Accuracy and precision of timings: All timings reported in this paper are the entire program runtime minus the IO time, and all timings are measured three times with the average taken.

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: Since the cost function to be minimized in Eqn. (6) of this paper is a strictly convex function, the converged results are always the same, no matter what the initial conditions are.

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. This paper controls different variables that influence computational performance. Table 1 controls the same number of sub-volumes, but varies the number of data subsets. Table 2 controls the same number of data subsets, but varies the number of sub-volumes. Memory footprint reduction, runtimes, and strong scaling efficiencies are independently measured for tables 1 and 2. In addition, this paper uses two datasets, one dataset from Berkeley National Lab and another dataset from Air Force Research Lab, for performance evaluations. The results from the two datasets are consistent.