iFDK: A Scalable Framework for Instant High-resolution Image Reconstruction

Peng Chen
Tokyo Institute of Technology
AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory, National Institute of Advanced Industrial Science and Technology
chen.p.aa@m.titech.ac.jp

Mohamed Wahib
AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory, National Institute of Advanced Industrial Science and Technology
mohamed.attia@aist.go.jp

Shinichiro Takizawa
AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory, National Institute of Advanced Industrial Science and Technology
shinichiro.takizawa@aist.go.jp

Ryousei Takano
National Institute of Advanced Industrial Science and Technology
takano-ryousei@aist.go.jp

Satoshi Matsuoka
Tokyo Institute of Technology
RIKEN Center for Computational Science, Hyogo, Japan
matsu@acm.org

ABSTRACT

Computed Tomography (CT) is a widely used technology that requires compute-intense algorithms for image reconstruction. We propose a novel back-projection algorithm that reduces the projection computation cost to 1/6 of the standard algorithm. We also propose an efficient implementation that takes advantage of the heterogeneity of GPU-accelerated systems by overlapping the filtering and back-projection stages on CPUs and GPUs, respectively. Finally, we propose a distributed framework for high-resolution image reconstruction on state-of-the-art GPU-accelerated supercomputers. The framework relies on an elaborate interleave of MPI collective communication steps to achieve scalable communication. Evaluation on a single Tesla V100 GPU demonstrates that our back-projection kernel performs up to 1.6× faster than the standard FDK implementation. We also demonstrate the scalability and instantaneous CT capability of the distributed framework by using up to 2,048 V100 GPUs to solve 4K and 8K problems within 30 seconds and 2 minutes, respectively (including I/O).

CCS CONCEPTS

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

KEYWORDS

Computed Tomography, GPU, Distributed, Heterogeneous Computing

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

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356163

ACM Reference Format:
Peng Chen, Mohamed Wahib, Shinichiro Takizawa, Ryousei Takano, and Satoshi Matsuoka. 2019. iFDK: A Scalable Framework for Instant High-resolution Image Reconstruction. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3295500.3356163

1 INTRODUCTION

High-resolution Compute Tomography (CT) is a technology used in a wide variety of fields, e.g. medical diagnosis, non-invasive inspection [62], and reverse engineering [17, 50]. In the past decades, the size of a single three-dimensional (3D) volume generated by CT systems has increased from hundreds of megabytes (the typical sizes of a volume are 256^3, 512^3) to several gigabytes (i.e. 2048^3, 4096^3) [7, 42, 66]. The increased demand for rapid tomography reconstruction and the associated high computational cost attracted heavy attention and efforts from the HPC community [8, 11, 19, 25, 28, 47, 54, 55, 66, 68, 76]. As illustrated in [48], the FDK [1] algorithm is widely regarded as the primary method to reconstruct 3D images (or volumes) from projections, i.e. X-ray images. The FDK algorithm includes a filtering stage (also known as convolution) and a back-projection stage. The computational complexities of those two stages are $O(N^2 log(N))$ and $O(N^4)$, respectively. Researchers are increasingly relying on the latest accelerators to improve the computational performance of FDK, e.g. Application Specific Integrated Circuits (ASIC) [72], Field-Programming Gate Array (FPGA) [16, 27, 64, 75], Digital Signal Processor (DSP) [37], Intel Xeon-Phi [53], Multi-core CPUs [68], and Graphics Processing Unit (GPU) [51, 73, 77, 78]. This paper focuses on GPU-accelerated supercomputers for two reasons. First, GPUs are dominantly used for tomographic image reconstruction [20, 28, 33, 55, 59, 74]. Second, GPU-accelerated supercomputers are increasingly gaining ground in top-tier HPC systems.

[1] Feldkamp, Davis, and Kress [23] presented a convolution-backprojection formulation (known as FDK algorithm) for CT image reconstruction in 1984. FDK is also known as the Filtered Back Projection (FBP) algorithm.

Instantaneous high-resolution image reconstruction, i.e. generating a volume moments after processing the scanned image projections, has long been the holy grail of CT technologies. The following are some of the challenges one has to consider when targeting instant high-resolution image reconstruction. First, FDK is a well-researched algorithm, yet it remains a fact that FDK, with its high-compute intensity, can benefit from innovation at the algorithm level in order to reduce the cost of computing the projections (while preserving computational precision). Second, CPUs and GPUs have different architectures. A fully heterogeneous solution necessitates careful assignment and orchestration of computing tasks to CPUs and GPUs. Third, high-resolution image reconstruction is limited by GPU memory capacity. Taking a volume of size 4096^3 for instance, the required storage is 256GB, which largely exceeds the memory capacity of a single GPU. Hence a distributed implementation is essential to avoid the GPU memory capacity bottleneck. Fourth, the effective use of MPI over the system hierarchy is critical to optimize data movement within the system. Finally, non-trivial optimizations are required to design an end-to-end pipeline for the computation of the filtering and back-projection stages using thousands of CPUs/GPUs (and this includes the I/O bottleneck of the parallel file system).

We target to enable instantaneous high-resolution image reconstruction while also providing the technical capacity required for advancing to unprecedented resolutions, e.g. 8192^3. We propose a scalable framework, called iFDK, for computing FDK on GPU-accelerated supercomputers. Optimizing the back-projection stage is crucial since back-projection is the computational bottleneck in most of the practical CT image reconstruction algorithms. We propose a novel back-projection algorithm that reduces the number of operations for computing the projection computations to a factor of 1/6 from the standard FDK algorithm. The algorithm is also general and thus can be adopted by iterative reconstruction methods, in which the back-projection is required to be repeated dozens of times, e.g. ART [24], SART [3], MLEM [61], and MBIR [2].

Prevalent approaches in the literature execute both the filtering and back-projection stages on GPUs. Contrarily, we improve the efficiency by taking full advantage of the heterogeneity in GPU-accelerated supercomputers. The filtering stage is executed on CPUs with optimizations for multi-threading and SIMD vectorization [15]. The back-projection stage is executed on GPUs with optimizations at the algorithmic level to reduce the cost of the projection computations while also improving the data locality.

The proposed framework is further optimized for reducing communication. More specifically, we propose a scalable problem decomposition scheme at which several independent sub-tasks are decomposed to a two-dimensional mesh of MPI ranks. The image reconstruction problem is decomposed such that the horizontal MPI ranks handle the input while the vertical ranks generate the output volume. Most importantly, the sub-tasks are overlapped in a pipelined fashion to be computed in parallel.

Using more than 2,000 Nvidia Tesla V100 GPUs, we solve the 4K ($2048\times2048\times4096\rightarrow4096^3$) and 8K ($2048\times2048\times4096\rightarrow8192^3$) high-resolution image reconstruction problems [2] within 30 seconds and 2 minutes, respectively. This includes the end-to-end processing time: loading projections from the Parallel File System (PFS), the filtering stage, the back-projection stage, MPI communication, and finally storing the output 3D volume to the PFS.

The contributions in this paper are as follows:

- We propose a novel back-projection algorithm that reduces the cost to compute the projections and improves cache locality.
- We propose a scalable and distributed framework for high-resolution image reconstruction on heterogeneous supercomputers.
- We demonstrate that high-resolution image reconstruction problems can be solved within tens of seconds by iFDK. To the author's knowledge, this is the first attempt to achieve instant distributed CT image reconstruction for 4K and 8K resolutions.

The rest of this paper is organized as follows. In Section 2, we review the background of CUDA and FDK algorithm. In Section 3, we propose the novel FDK algorithm and CUDA implementation. In Section 4, we propose the distributed framework (namely iFDK) and present the performance model. Section 5 describes the evaluation results. Section 6.1 discusses potential impact of iFDK in real-world. In Section 7, we introduce the related work. Finally, Section 8 concludes.

2 BACKGROUND

In this section, we introduce the basics of CUDA and describe the details of the FDK algorithm.

2.1 CUDA

Nvidia Compute Unified Device Architecture (CUDA) is a parallel computing platform and application programming model. We briefly introduce the concepts of CUDA's architecture and memory hierarchy, more details can be found in [18].

CUDA architecture. CUDA is built on an array of multi-threaded Streaming Processors (SMs). Massive thread-level parallelism is abstracted into a hierarchy of threads running in a single instruction multi-thread (SIMT) fashion. The threads in CUDA are grouped into warps (32 threads execute as a warp), blocks, and grids. Thousands of threads are created, scheduled, and executed concurrently.

CUDA memory hierarchy. To approach the peak performance of GPU, it is essential to implement applications that efficiently utilize CUDA's memory hierarchy. (I) **Global memory**. The largest off-chip memory. A coalesced access pattern is required for the CUDA kernels to achieve the highest bandwidth. (II) **Shared memory**. A fast on-chip scratchpad memory, which shares space with the L1 cache. The access scope is limited to a single CUDA block. (III) **Constant memory**. A read-only constant cache shared by all of the SMs. (IV) **Texture memory**. A read-only on-chip memory which is optimized for the spatial locality. The operation of reading a texture is also called texture fetch. Texture fetch [56] can support efficient sub-pixel interpolation (as will be shown in Alg 3). (V) **Register files**. The fastest on-chip and thread-private memory.

CUDA shuffle intrinsic. CUDA provides efficient intra-warp communication instructions, namely, shuffle. Without using shared or global memory, threads in a single warp can exchange registers directly. In terms of computing efficiency, using shuffle for in-register computation is superior in performance to the other memory types, due to its low latency and high throughput [14].

[2] The image reconstruction problem is defined in Section 2.3.

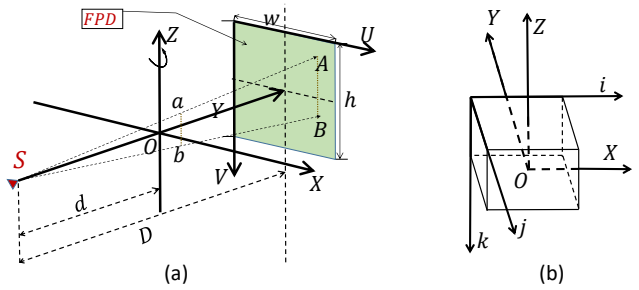


Figure 1: CBCT geometry and trajectory.

Table 1: CBCT parameter list.

Param	Description	Unit
N_p	the number of 2D projections	N/A
N_u, N_v	the width and height of a 2D projection, respectively	pixel
D_u, D_v	FPD pixel pitch in U and V direction, respectively	mm/pixel
E_i, Q_i	the i^{th} projections and the filtered result, respectively	N/A
F_{ramp}	1-D Ramp filter [23]	N/A
F_{cos}	2-D cosine table of size (N_v, N_u) [23]	N/A
P_i	the i^{th} projection matrix of size 3×4 [52, 53]	N/A
d	distance of X-ray source to rotation Z-axis	pixel
D	distance of X-ray source to FPD center	pixel
N_x, N_y, N_z	the number of voxels in X, Y, Z dimension, respectively	voxel
D_x, D_y, D_z	the pitch of volume in X, Y, Z dimension, respectively	mm/voxel
θ	rotation step angle, $\theta = 2 \cdot \pi / N_p$	Rad
I	3D volume	N/A

2.2 FDK algorithm

In this section, we revisit the 3D image reconstruction method (namely FDK) for Cone-Beam Computed Tomography (CBCT) as introduced by Feldkamp et al [23]. We briefly introduce geometry, arithmetic computation, and convolution. More details can be found in [34].

2.2.1 **CBCT Geometry.** Figure 1 illustrates the CBCT geometry in detail and Table 1 lists all of the related parameters. S is a micro-focus X-ray source, FPD (Flat Panel Detector) is a class of x-ray digital radiography detectors, which is principally similar to the image sensors used in digital photography. In addition, both the X-ray source and FPD are fixed relatively in position as Figure 1a shows. Both of them rotate around the Z-axis while scanning objects placed at the center O to express a 3D volume as shown in Figure 1b.

2.2.2 **FDK Computation.** FDK is widely employed to build tomographic images in clinical and medical practice [48]. FDK comprises a filtering stage (or convolution stage) as Algorithm 1 shows, and a back-projection stage shown in Algorithm 2. In Algorithm 1, cosine weighting (F_{cos}) and ramp filter (F_{ramp}) are convolved with projections (E_i) to generate the filtered result (Q_i). The details of F_{cos} and F_{ramp} (including improved versions) can be found in [23, 34]. The shape of the F_{ramp} filter deeply affects the final image quality, yet it has no effect on the compute intensity of the filtering stage.

Algorithm 2 shows the back-projection algorithm. The projection matrix P_i, a well-aligned 3×4 matrix, incorporates all of the geometry information for the back-projection, i.e. d, D, θ. More details on computing the P matrix are presented in literature [52, 69]. It is important to mention the embarrassingly parallel nature of the back-projection computation when utilizing the projection matrix, in comparison to other methods in literature [38, 57]. As Algorithm 3 shows, the *bilinear interpolation* method is adopted by most

Algorithm 1 Filtering stage [34].

Input: E, F_{cos}, F_{ramp}, N_p, N_v, N_u
Output: Q
1: **for** $i \in [0, N_p)$ **do**
2: $\vec{E}_i \leftarrow E_i \cdot F_{cos}$ ▷ · means point-wise multiplication
3: **for** $j \in [0, h)$ **do**
4: $Q_i(j, \dots) \leftarrow \vec{E}_i(j, \dots) \otimes F_{ramp}$ ▷ ⊗ means convolution
5: **end for**
6: **end for**

Algorithm 2 Back-projection stage. This scheme is implemented in RTK [52], RabbitCT [53], etc.

Input: P, Q, N_p, N_x, N_y, N_z
Output: I ▷ generated 3D volume
1: $I \leftarrow 0$ ▷ I initialization
2: **for** $s \in [0, N_p)$ **do**
3: **for** $k \in [0, N_z)$ **do**
4: **for** $j \in [0, N_y)$ **do**
5: **for** $i \in [0, N_x)$ **do**
6: $[x, y, z]^T \leftarrow P_s \cdot [i, j, k, 1]^T$ ▷ 3 inner product
7: $f \leftarrow 1/z$
8: $W_{dis} \leftarrow f^2$ ▷ distance weight
9: $[u, v]^T \leftarrow [x, y]^T \cdot f$ ▷ coordinates in FPD
10: $I(i, j, k) \leftarrow I(i, j, k) + W_{dis} \cdot interp2(Q_s, u, v)$
11: **end for**
12: **end for**
13: **end for**
14: **end for**

Algorithm 3 Bilinear interpolation with sub-pixel precision [32].

1: **function** interp2(X, u, v) ▷ X is 2D matrix, (u, v) is sub-pixel coordinate
2: $[n_u, n_v]^T \leftarrow [\text{int}(u), \text{int}(v)]^T$ ▷ n_u, n_v are integers
3: $[d_u, d_v]^T \leftarrow [u - n_u, v - n_v]^T$ ▷ distance to left points
4: $t_1 \leftarrow X(n_u, n_v) \cdot (1 - d_u) + X(n_u + 1, n_v) \cdot d_u$ ▷ a sub-pixel value
5: $t_2 \leftarrow X(n_u, n_v + 1) \cdot (1 - d_u) + X(n_u + 1, n_v + 1) \cdot d_u$ ▷ a sub-pixel value
6: **return** $t_1 \cdot (1 - d_v) + t_2 \cdot d_v$ ▷ final sub-pixel value
7: **end function**

of the FDK implementations to fetch the intensity value of a 2D matrix with sub-pixel precision for updating each value of I in Algorithm 2 [52, 53].

2.2.3 **Convolution via FFT.** FFT [13] is essential to the filtering stage since one-dimensional FFT is used to perform the convolution operation in Algorithm 1. For large problem sizes, FFT is typically the choice for the convolution computation [12]. Optimized FFT primitives are typically provided by CPU/GPU vendors, e.g. Intel IPP (Intel Integrated Performance Primitives) [65], and Nvidia cuFFT library [18]. In Algorithm 1 line 4, the FFT primitive is employed to perform the convolution as follows. Regarding the two one-dimensional arrays, according to the Convolution Theorem [4], convolution computation in the time domain equals point-wise dot product in the frequency domain. The reader can refer to a detailed illustration in [41].

2.3 Terminology

Throughout this paper, the image reconstruction problem and performance metric GUPS are defined as follows:

(I) $N_u \times N_v \times N_p \rightarrow N_x \times N_y \times N_z$ is defined as the image reconstruction problem, where $N_u \times N_v \times N_p$ denotes the size of projections (**Input**) and $N_x \times N_y \times N_z$ is the size of volume (**Output**).

(II) **GUPS** is defined as giga-updates per second and is used as a performance metric. It is computed as $\text{GUPS} = \frac{N_x * N_y * N_z * N_p}{T * 2^{30}}$, where T is the execution time (in seconds).

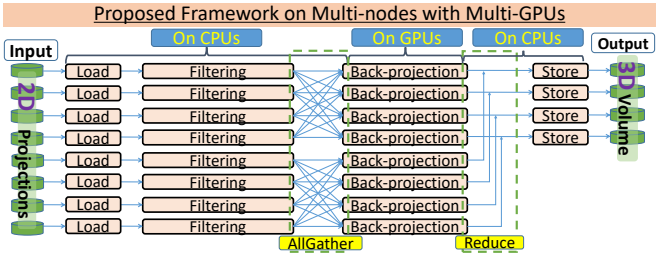


Figure 2: Overview of the proposed framework. Multi-nodes with GPUs accelerators are used such that a 3D volume is generated from 2D projections.

3 PROPOSED NOVEL FDK ALGORITHM

The next section discusses the execution of the filtering stage on the CPUs and the following section discusses the execution of the novel FDK algorithm (back-projection stage) on the GPUs.

3.1 Filtering Stage

Figure 2 shows our heterogeneous computational flow, which is different from the typical method of using only the GPU for all the computation [10, 38, 46]. Utilizing the CPU to perform the filtering stage can be more efficient in comparison to using the GPUs to compute the entire FDK pipeline. **We list the three considerations** that makes the fully heterogeneous model more efficient: (I) **Dedication of GPU to the compute intense stage:** the latency of the filtering stage on the CPU can be hidden by overlapping it with the back-projection offloaded to the GPU. Hence, the GPU can be fully dedicated to the back-projection kernel (see Section 3.3). (II) **Reduce memory pressure on GPU device memory:** in the case that the GPU would be used for filtering, a batch of images have to be filtered at a time in order to fully utilize the GPU. The additional memory required to store the filtered projects would take away from the already limited GPU memory and would force the use of more GPUs in high-resolution problems to have enough aggregate memory capacity. (III) **Efficient communication:** in our communication scheme, an AllGather collective is required after the filtering stage to fetch the filtered projections. When the filtering stage is applied at the GPU, the AllGather collective will be applied on data residing in GPU memory and not on data residing in the CPU memory (as is the case when the filtering is applied on the CPU). Applying AllGather on data residing on the GPU incur the extra cost of moving data across the PCIe interconnect, even when the GPUDirect [36, 49] Technology is enabled.

3.2 Back-projection Stage

We propose a general back-projection algorithm that reduces the computation and improves the data locality (regardless of the target architecture).

3.2.1 **Theorems for Back-projection Algorithm.** This section introduces three theorems which we use to propose a novel version of the original FDK algorithm at Algorithm 2.

- **Theorem-1:** As Figure 1 shows, when two 3D points a $(\tilde{i}, \tilde{j}, \tilde{k})$ and b $(\tilde{i}, \tilde{j}, \tilde{N_z} - k - 1)$ are symmetrical to XY plane, then the corresponding projection points A $(\tilde{u_A}, \tilde{v_A})$ and B $(\tilde{u_B}, \tilde{v_B})$ at FPD are symmetrical to the horizontal center line, namely $\tilde{u_A} = \tilde{u_B}$ and $\tilde{v_A} + \tilde{v_B} = N_v - 1$. (proven by [77])
- **Theorem-2:** For points in the vertical line $\overline{ab}$ (parallel to Z-axis), their projection line $\overline{AB}$ is parallel to V-axis in FPD plane, namely the $\tilde{u}$ of line $\overline{AB}$ has a constant value. (the mathematical proof is trivial, we do not include it due to space constraints)
- **Theorem-3:** When points are in the vertical line $\overline{ab}$ (parallel to Z-axis), such that computing the projection points use Equation 1, then their z is a constant value equal to $d + y_{ab}$, where y_{ab} is the Y coordinate of line $\overline{ab}$. (proof follows).

Proof of Theorem-3. We prove that in the specified rotation angle β (or $i{\cdot}\theta$), if the $\hat{i}$ and $\hat{j}$ are fixed, the z in Equ 1 is a constant value. As Fig 1 shows, given a point a of coordinate $(\tilde{i}, \tilde{j}, \tilde{k})$ in the volume coordinate system, its projection point A of coordinate (u, v) on the FPD can be computed using the projection equation as

$$\begin{cases} [x, y, z]^T = P_i \cdot [\tilde{i}, \tilde{j}, \tilde{k}, 1]^T \\ [u, v]^T = [x, y]^T \cdot 1/z \end{cases} \tag{1}$$

where x,y,z are temporary variables. P_i is a 3×4 projection matrix on the condition that the gantry rotation angle is β. Hence, P_i may be written as

$$\begin{cases} \hat{P}_i = M_1 \cdot M_{rot} \cdot M_0 \\ P_i = \hat{P}_i[0:3] \end{cases} \tag{2}$$

where the shapes of $\hat{P}_i$ and P_i are 4×4 and 4×3, respectively. M_0, M_{rot}, and M_1 are listed as follows

$$M_0 = \begin{pmatrix} D_x & 0 & 0 & 0 \\ 0 & D_y & 0 & 0 \\ 0 & 0 & D_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -(N_x-1)/2 \\ 0 & -1 & 0 & (N_y-1)/2 \\ 0 & 0 & -1 & (N_z-1)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{rot} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & d \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\beta) & -sin(\beta) & 0 & 0 \\ sin(\beta) & cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_1 = \begin{pmatrix} 1/D_u & 0 & 0 & 0 \\ 0 & 1/D_v & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} D & 0 & (N_u-1)\cdot D_u/2 & 0 \\ 0 & D & (N_v-1)\cdot D_v/2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

M_0 represents the transformation of the coordinate system from volume to gantry [71], M_{rot} denotes the gantry rotation along the Z-axis at the angle β plus the transpose distance of d. M_1 indicates the projection point on the FPD plane. Note that all of the variables are listed in Table 1. By expanding Equation 1, the z can be written as

$$z = d + sin(\beta)(\tilde{i} - (N_x - 1)/2) \cdot D_x - cos(\beta) \cdot (\tilde{j} - (N_y - 1)/2) \cdot D_y \tag{3}$$

Clearly, the z is independent of $\hat{k}$ and equals d+y_{ab}, where y_{ab} is the Y coordinate of line $\overline{ab}$, which is parallel to the Z-axis in Figure 1a.

3.2.2 **Reducing the Cost of Computing the Projections.** In this section, we present a method to improve the performance of back-projection by reducing the projections computational cost. Algorithm 2 is adopted in a number of CBCT applications, e.g. open-source libraries (OSCaR [51], RabbitCT [53], RTK [52]), and literature [31, 46, 74, 77]. Based on *Theorem-2* and *Theorem-3*, the

Algorithm 4 Proposed Back-projection algorithm. The optimized variables are highlighted in gray color.

Input: $P_i, Q_i, N_p, N_x, N_y, N_z, i \in [0, N_p)$
Output: I ▷ reconstructed 3-dimensional volume

1: $\tilde{I} \leftarrow 0$ ▷ $\tilde{I}$ initialization
2: **for** $s \in [0, N_p)$ **do**
3: $\tilde{Q_s} \leftarrow Q_s^T$ ▷ transpose 2D matrix
4: **for** $j \in [0, N_y)$ **do**
5: **for** $i \in [0, N_x)$ **do**
6: $t \leftarrow [i, j, 0, 1]$
7: $[x, z] \leftarrow [\langle P_s[0], t\rangle, \langle P_s[2], t\rangle]$ ▷ 2 inner product
8: $f \leftarrow 1.0/z$
9: $u \leftarrow x \cdot f$
10: $W_{dis} \leftarrow f^2$ ▷ weighting
11: **for** $k \in [0, N_z/2)$ **do** ▷ symmetric geometry
12: $y \leftarrow \langle P_s[1], [i, j, k, 1]\rangle$ ▷ 1 inner product
13: $v \leftarrow y \cdot f$ ▷ compute v only
14: $\tilde{I}(k, j, i) \leftarrow \tilde{I}(k, j, i) + W_{dis} \cdot interp2(\tilde{Q_s}, v, u)$
15: $\tilde{k} \leftarrow N_z - 1 - k$ ▷ symmetric geometry
16: $\tilde{v} \leftarrow N_v - 1 - v$ ▷ symmetric geometry
17: $\tilde{I}(\tilde{k}, j, i) \leftarrow \tilde{I}(\tilde{k}, j, i) + W_{dis} \cdot interp2(\tilde{Q_s}, \tilde{v}, u)$
18: **end for**
19: **end for**
20: **end for**
21: **end for**
22: $I \leftarrow reshape(\tilde{I})$ ▷ reshape means changing data layout

improved back-projection is illustrated in Algorithm 4. The computational cost for computing the projections becomes a factor of 1/6 of the original FDK algorithm (Alg 2 line 6), since we only compute half of the N_z dimension (Algorithm 4 line 11), and one of inner products for the two 1×4 vectors (Algorithm 4 line 12) instead of the three inner products in the original algorithm (Algorithm 2 line 6). Algorithm 4 shows the optimized variables highlighted in gray color. More specifically, the values of u and W_{dis} are reused for N_z times, and only half of y is computed directly. Zhao et al. [77] discussed a rotational symmetry in the projection layout. We do not adopt this methodology since it is impractical for pipeline processing in terms of the high latency of collecting the four projections.

3.2.3 **Improving Data Locality: Data Layout & Loops.** This section discusses how we leverage the proposed algorithm to derive an implementation that improves the data locality. To increase the cache hits for accessing the volume (I) and projection (Q) in Algorithm 2, the proposed Algorithm 4 adjusts the memory layout and re-organizes the loops for N_p, N_x, N_y, and N_z. The original I uses an i-major layout as Figure 1b shows. The I becomes k-major in the proposed algorithm. Based on *Theorem-2*, the proposed memory layout is more cache-friendly for data access, since the data buffers of both the projections and the volume can be accessed contiguously, as shown by the marked variables of $\tilde{Q_s}$ and $\tilde{I}$ in Algorithm 4.

Note that this memory layout is general to all kinds of processors, i.e. CPUs, GPUs, and Xeon Phi. To improve the data locality, the authors in [38, 78] implemented the back-projection kernel on GPUs by organizing the loops as Algorithm 4, such that they compute along the z-axis first. However, that method does not optimize for the layout of arrays Q and I. Hence, their implementation becomes further complex since one has to rely on CUDA's 2D-texture cache to improve the data locality. It is noteworthy that the time required to transpose a projection (Algorithm 4 line 3) is a small fraction of

Listing 1: The proposed back-projection CUDA kernel. Constant memory-optimized ProjMat is defined to store the 3×4 projection matrixes. The *dot* function computes inner product, *mad* is the fused-multiply-and-addition intrinsic, *interp2* function is implemented as Algorithm 3. The batch of projections (defined as N_{batch}) is 32.

```
1  __constant float4 ProjMat[32][3];//32 3x4 proj-matrix
2  __global void shflBP(float* vol, int3 vol_size, const float* img
     , int3 img_dim){
3    int laneId = threadIdx.x & 31;
4    //(k,j,i) is the position of a voxel
5    int i = blockIdx.x*blockDim.x + threadIdx.x;
6    int j = blockIdx.y*blockDim.y + threadIdx.y;
7    int k = blockIdx.z*blockDim.z + threadIdx.z;
8    float4 vec = make_float4(k, j, i, 1);
9    //Z, U are 2 registers
10   float Z, U;
11   if (laneId < img_dim.z) {
12      Z = 1/dot(ProjMat[laneId][2], vec); //inner product
13      U = dot(ProjMat[laneId][0], vec)*Z; //inner product
14   }
15   float sum = 0, _sum = 0, v, _v;
16   int _i = vol_dim.z - 1 - i; //geometry symmetric
17   for (int s = 0; s<img_dim.z; s++) {
18     //get register value via shuffle
19     float u = __shfl_sync(0xffffffff, U,  s);
20     float f = __shfl_sync(0xffffffff, Z,  s);
21     float Wdis=f*f;                    //weighting
22      v = dot(ProjMat[s][1], vec)*f;    //inner product
23     _v = img_dim.x - 1 - v;            //geometry symmetric
24     //sub-pixel interpolation & weighting
25     sum=mad(interp2(img,s,img_dim.x,img_dim.y, v,u),Wdis, sum);
26    _sum=mad(interp2(img,s,img_dim.x,img_dim.y,_v,u),Wdis,_sum);
27   }
28   //update voxels
29     pVolOut                              [ i] +=  sum;
30    (pVolOut + (k*vol_dim.y+j)*vol_dim.z)[_i] += _sum;
31 }
```

the filtering (or back-projection) stage and thus, we do not discuss its effect on the overall performance of FDK in the later sections.

3.3 Back-projection on GPU

In this section, we introduce the proposed back-projection implementation in CUDA and elaborate on the optimization of the CUDA kernel using the shuffle intrinsic.

3.3.1 **CUDA Implementation.** This section describes the proposed CUDA implementation. We implement the proposed back-projection kernel (called shflBP in Listing 1), which can be used in all generations of Nvidia GPUs from Kepler to Volta architectures. In Listing 1, the detailed CUDA kernel is presented. We use global memory (as introduced in Section 2.1) to store the 3D volume (see the variable of *vol* in Listing 1) due to its huge size. Though the CUDA unified memory [18] is also an attractive choice for storing 3D volume, we avoid using it due to its unstable performance, which varies from CUDA version to version as reported in [6, 39]. To further reduce the computation cost by sharing data between threads, we take advantage of the shuffle instruction to perform intra-warp communication as introduced in Section 2.1. The shflBP kernel processes a batch of projections in one pass. This benefits the overall performance in many aspects: (I) decreasing the access count of the volume data which is stored in the global memory. (II) increasing in-register accumulation for back-projection computation. (III) eliminating the overhead of launching multiple CUDA kernels. Note that this strategy is also applied in the widely used image library RTK [52]. However, as explained earlier, the proposed

Table 2: iFDK parameter list.

Parameter	Descriptions
R, C	the rows and columns of 2D grid of mpi ranks, respectively
R_i	the i^{th} row of 2D mpi ranks, where $i \in [0, R)$
C_i	the i^{th} column of 2D mpi ranks, where $i \in [0, C)$
$N_{gpu\_per\_node}$	the number of GPUs per compute node
$N_{cpu\_per\_node}$	the number of CPUs per compute node
N_{nodes}	the total number of compute nodes
N_{ranks}	the total number of launched ranks
N_{gpus}	the total number of GPUs
N_{cpus}	the total number of CPUs
N_{PCIe}	the number of PCIe connector per compute node
$N_{proj\_per\_rank}$	the number of loaded and filtered projections per rank
N_{cpu_core}	the total number of cores per CPU
$N_{gpu\_mem\_size}$	the memory capacity of a single GPU

algorithm outperforms RTK by reducing the computational cost and improving data locality.

3.3.2 **Shuffle-based CUDA kernel: shflBP.** In this section, we explain the details of applying shuffle to our CUDA kernel. We employ 2 registers (named Z, U) in the shflBP kernel (Listing 1 line 10) to store the values of f and u (Algorithm 4 line 7~9). Hence, the value of variable W_{dis} (Algorithm 4 line 10) can be easily obtained as *Wdis* in Listing 1 line 21. We avoid the use of shared or global memory all together by taking advantage of the shuffle instruction to realize the data communication between threads (Listing 1 line 19~20). Since the scope of the shuffle instructions is limited to a single CUDA Warp, the strategy of sharing data as Algorithm 4 is adjusted to a single CUDA Warp. To the best of our knowledge, exchanging registers by shuffle is superior in terms of effective throughput, in comparison to using the shared memory [14]. Additionally, the shflBP kernel does not require thread block barriers, which is often required when using shared memory.

4 DISTRIBUTED FRAMEWORK FOR HIGH-RES. IMAGE RECONSTRUCTION

This section presents a distributed framework, namely iFDK, for instant high-resolution image reconstruction. The parameters used in iFDK are summarized in Table 2.

4.1 Design and Implementation of iFDK

We combine the CPU filtering stage, the GPU-optimized back-projection stage, and MPI as a communication library to scale iFDK to the $O(1000)$ GPUs. This section elaborates on the design choices and implementation.

4.1.1 **2D Grid of MPI Ranks.** The compute capability and device memory capacity are limited in a single GPU. It is impractical to use a single GPU to generate large volumes (e.g. volumes of size 4096^3 or 8192^3). Therefore, we scale the proposed method to take advantage of GPU-accelerated supercomputers to solve those problems. This section presents the problem decomposition and orchestration of MPI ranks. To fully utilize the computing resources, i.e. CPUs, GPUs, inter-connectors, we launch multiple MPI ranks within each node (one rank per GPU) to perform the computation and communication concurrently.

The MPI ranks are managed as a 2D-grid of R rows and C columns. The total number of ranks may be expressed as

$$N_{ranks} = C * R \tag{4}$$

In Figure 3a, an example of using iFDK with 32 MPI ranks is presented, where R=8 and C=4. In iFDK, ranks in each column of the 2D-grid load a subset of projections from the PFS independently. Next, we perform the filtering stage on those projections using the CPUs. The number of projections processed by the ranks in each column of the 2D-grid is N_p/C. We use MPI-AllGather to send the filtered projections to neighbor ranks in the same group (namely the same column). Hence, the number of projections which are loaded and filtered by a single rank is

$$N_{proj\_per\_rank} = N_p/N_{ranks} = N_p/(C * R) \tag{5}$$

Each rank in the same row of the 2D-grid (or the R_i^{th} row) computes the same sub-volumes as Fig 3b shows, the final sub-volume that is reconstructed by the R_i^{th} row can be obtained by reducing all of the sub-volumes that are generated by the ranks in the same group (or the R_i^{th} row).

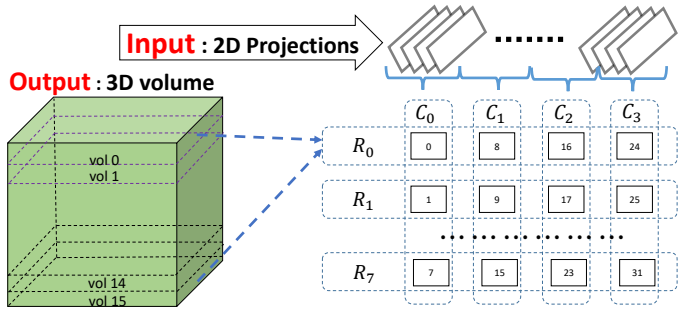
4.1.2 **Multi-GPU Management.** We discuss how the GPUs are managed by MPI ranks in this section. Commonly, in a single compute node, there are multiple GPUs (e.g. ORNL'Summit has six GPUs, LLNL's Sierra and TokyoTech's Tsubame have four GPUs), which are connected to the CPUs by PCIe or NVLink [36]. We launch a number of MPI ranks per compute node equivalent to the number of GPUs, i.e. one MPI rank per GPU. The N_{gpus} may be written as

$$N_{gpus} \equiv N_{ranks} \tag{6}$$

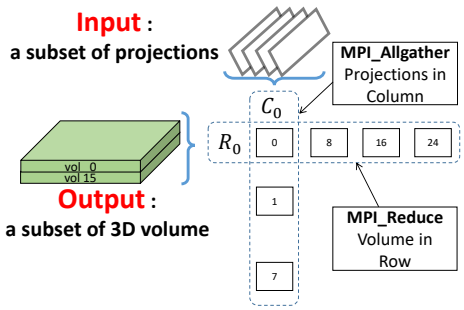
Here, $N_{nodes} = N_{ranks}/N_{gpu\_per\_node}$ is the number of required compute nodes. Each MPI rank manages a single GPU as follows: first, we gather the filtered projections that are processed on the CPU, as explained in Section 3.1. Second, the processed projections are copied from the host to device memory. Third, the back-projection kernel is launched to generate the specified subset of 3D volume. Finally, the computed volume is copied from the device memory to the host. The detailed operations performed inside each rank, by multiple threads, are presented in the next section.

4.1.3 **Multiple Threads in a Rank.** This section presents how multiple threads are orchestrated in each MPI rank. Each rank in iFDK processes several tasks in parallel, e.g. loading projections from PFS, filtering the projections, collective communication, back-projection, and storing the volume to PFS. As Figure 4 shows, we use three threads to execute those tasks, namely *Main-thread*, *Filtering-thread*, and *Bp-thread* (Back-projection thread) . Those threads are created using the *pthread* library, they execute independently and exchange data with each other using circular buffers [70]. The Filtering-thread launches OpenMP threads of number $(N_{cpu\_core} * N_{cpu\_per\_node}/N_{ranks} - 1)$ to load projections and execute the filtering in parallel. For each projection, the load and filtering operations are executed within the same OpenMP thread in the sequence that enables immediate processing.

As Figure 3b shows, we use the MPI-AllGather collective to gather the specified subset of filtered projections in the Main-thread. At the same time, those filtered projections are dispatched to the designated GPU by Bp-thread for the back-projection computation. Note that for each MPI rank, $N_{proj\_per\_rank}$ times of AllGather operations are required since we process one projection at a time by AllGather. When all of the filtered projections are finally processed



(a) iFDK framework. The input is the 2D projections, the output is the generated 3D volume. vol denotes a sub-volume. C_i and R_i are defined in Table 2.



(b) Allgather is performed across the ranks in one column. Reduction is applied only once across the ranks in one row.

Figure 3: Illustrative example of the iFDK framework. 32 MPI ranks are arranged in a 2D grid of size 4×8 (R=8, C=4). The input projections are decomposed into 4 groups corresponding to the columns C. The output volume are aggregated from 16 (2*R) sub-volumes.

by the GPUs, the generated sub-volume on the GPU device memory is copied back to the host. Next, using the MPI-Reduce collective as in Figure 4b, we do a single reduction step to generate the final resulting volume in host memory using the Main-thread (a real example can be found in Figure 7). Finally, the 3D volume is stored in the PFS using multi-ranks with multi-threads. Note that the volume of size $N_x \times N_y \times N_z$ is stored as slices of number N_z, the size of each slice is $N_x \times N_y$. There is room for improvement by tuning the size of each slice to optimize for the throughput of storing to the PFS (i.e. tune slice size to optimize for file striping).

4.1.4 **Orchestration and Overlapping.** In order to achieve the optimal overlap, we use three threads to pipeline the computation as Figure 4a shows. Figure 4c presents a real example of solving a 4K problem using 128 V100 GPUs. To further give insight into the effect of pipelining the computation, the breakdown of the overlapped computation (namely $T_{compute}$, as defined Section 4.2) is listed in Table 5. The value of δ>1 indicates that we achieve the goal of improving the overall performance by overlapping different stages, i.e. filtering, collective communication, and back-projection. This overlapping scheme improves overall performance since back projection is the main bottleneck, and hence we get a streaming benefit from the overlapping. On the other hand, overlapping the tasks after the back-projection (i.e. the device to host copy, reduction, and storing to PFS) does not guarantee any performance improvement (for the price of complexity introduced). Nonetheless, overlapping after the back-projection remains to be one of the points of investigation in future work.

4.1.5 **Configuration of Parameter R.** This section discusses how to select the optimal value for the parameter R: the number of rows in the 2D mesh of MPI ranks. Based on the design of iFDK (in Figure 3a), R may be expressed as

$$R = sizeof(float) * N_x * N_y * N_z / N_{sub\_vol} \quad (7)$$

where N_{sub_vol} is the size of sub-volume. For a specified number of GPUs (or ranks), we minimize the value of R and maximize the value of C for three reasons: (I) We can efficiently use the limited device memory since each GPU would be able to compute a volume of size $sizeof(float) * N_x * N_y * N_z / R$; (II) We can achieve higher computational performance by generating larger volumes. As Table 4 shows, regarding the specified input, bigger output (smaller α) results in better performance for the back-projection kernel; (III) To the specified workload of $N_u \times N_v \times N_p \rightarrow N_x \times N_y \times N_z$, there is a linear relationship between the runtime and N_p. Since we decompose the workload to sub-tasks ($N_u \times N_v \times \frac{N_p}{C} \rightarrow N_x \times N_y \times N_z$) of number C, it is essential to maximize the value of C to decrease the runtime of each sub-task.

In addition, the value of R is often power of two and is constrained by the memory capacity of a GPU as follows

$$sizeof(float) * (\frac{N_x * N_y * N_z}{R} + N_u * N_v * N_{batch}) \leq N_{gpu\_mem\_size}$$

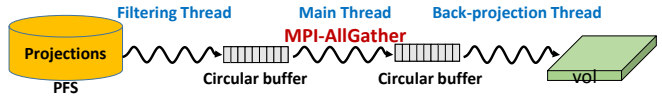
where N_{batch}=32 as Listing 4 shows. Given $N_{gpu\_mem\_size}$=16GB in the GPU generation we use, for the high-resolution reconstruction problems we target, N_{sub_vol}=8GB is adopted.

4.2 Performance Model

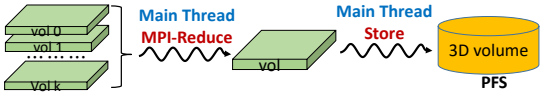
We discuss a performance model intended to analyze the impact of different parameters on performance and predict the potential peak performance.

4.2.1 **Micro-benchmarks.** We use micro-benchmarks to measure peak throughput parameters of constant values in our model (for a given system). BW_{load} and BW_{store} are the aggregate throughput of reading and writing to the PFS, respectively. Both of them are measured by LLNL IOR [58]. TH_{flt} is the throughput of filtering computation that we measure by running the filtering kernel on the target CPU. TH_{bp} is the throughput of our back-projection kernel as measured on the target GPU (we also report it in Table 4 to give perspective to readers interested in only the back projection kernel). TH_{trans} is the throughput of transposing a volume on GPU. The $TH_{AllGather}$ and TH_{Reduce} are the throughput of MPI-AllGather and Reduce APIs, respectively. Both of them are measured by Intel mpi-benchmarks. BW_{PCIe} is the throughput of data transfer between the host and device memory via a single PCI-e and is measured by Nvidia's tool called bandwidthTest.

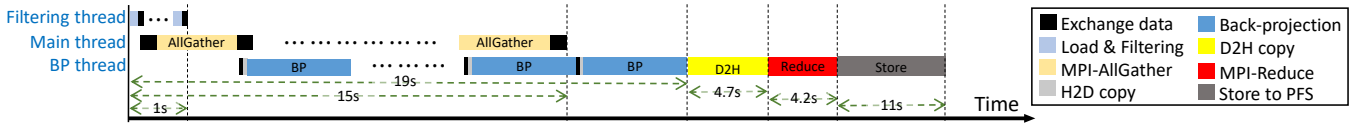
4.2.2 **iFDK Performance Model.** Given execution time required for: reading projections from storage T_{load}, filtering projections



(a) Three threads execute in parallel in a pipeline fashion and exchange data via two queue-buffers. MPI-AllGather is performed the Main thread.



(b) Reduce all the sub-volumes generated in the previous step to a single final volume to be stored in the PFS by the main thread.



(c) Example of pipeline to solve $2048^2 \times 4096 \rightarrow 4096^3$ problem using 128 V100 GPUs. R=32 and C=4. Measured execution time for each task is displayed. The Filtering thread processes 32 projections. The Main thread sends 32 projections and gathers 1024 projections using MPI-AllGather. The Bp-thread processes 1024 projections.

Figure 4: Orchestration and Overlapping in iFDK.

T_{flt}, communicating all projections by MPI-AllGather $T_{AllGather}$, copying filtered projections from host to device T_{H2D}, back-projection T_{bp}, transposing the sub-volume T_{trans}, moving the sub-volume from device memory to host T_{D2H}, reducing the sub-volume T_{Reduce}, and storing volume to PFS T_{store}. Those variables can be written as

$$T_{load} = sizeof(float) * N_u * N_v * N_p / BW_{load} \tag{8}$$

$$T_{flt} = \frac{N_p}{N_{nodes} * TH_{flt}} = \frac{N_p * N_{gpu\_per\_node}}{C * R * TH_{flt}} \tag{9}$$

$$T_{AllGather} = N_p / (C * R * TH_{AllGather}) \tag{10}$$

$$T_{H2D} = \frac{sizeof(float) * N_{gpu\_per\_node} * N_u * N_v * N_p}{C * BW_{PCIe} * N_{PCIe}} \tag{11}$$

$$T_{bp} = T_{H2D} + N_p / (C * TH_{bp}) \tag{12}$$

$$T_{trans} = sizeof(float) * N_x * N_y * N_z / (R * TH_{trans}) \tag{13}$$

$$T_{D2H} = \frac{sizeof(float) * N_{gpu\_per\_node} * N_x * N_y * N_z}{R * BW_{PCIe} * N_{PCIe}} \tag{14}$$

$$T_{reduce} = sizeof(float) * N_x * N_y * N_z / (R * TH_{Reduce}) \tag{15}$$

$$T_{store} = sizeof(float) * N_x * N_y * N_z / (BW_{store}) \tag{16}$$

As Figure 4 shows, the three threads compute in parallel such that most of the computation and data movement is overlapped. Additionally, the filtering thread launches multiple OpenMP threads to perform the loading and filtering operations concurrently as Section 4.1 explained. The filtering operation can be perfectly overlapped since $T_{load} + T_{flt} \ll T_{bp}$, as the example shows in Figure 4c. Here, the execution time required by the three threads may be expressed as

$$T_{compute} = \max(T_{load}, T_{flt}, T_{AllGather}, T_{bp}) \tag{17}$$

The time required to transpose, copy and reduce the volume is

$$\begin{aligned} T_{post} &= T_{trans} + T_{D2H} + T_{reduce} + T_{store} \\ &\approx T_{D2H} + T_{reduce} + T_{store} \end{aligned} \tag{18}$$

Note that the T_{trans} is a small value ($T_{trans} \ll T_{D2H}/10$ is observed) that could be ignored . The total execution time is

$$\begin{aligned} T_{runtime} &= T_{compute} + T_{post} \\ &\approx T_{compute} + T_{D2H} + T_{reduce} + T_{store} \end{aligned} \tag{19}$$

4.2.3 **Conclusions from the Performance Model.** We conclude the discussion of our performance model as follows. (I) **Scalability:** The performance of iFDK scales with the number of GPUs (N_{gpus}) since T_{flt}, $T_{AllGather}$ and T_{bp} are inversely proportional to C (as in Equation 9~12), C is proportional to N_{gpus} (as in Equation 4 and Equation 6), where R is a value that is minimized to meet the constraints described in Section 4.1.5, where T_{post} is a constant value in $T_{runtime}$. (II) **Potential peak performance:** According to Equation 19, the potential peak performance of the computation ($T_{runtime}$) can be used to quantify the efficiency of our implementation.

5 EVALUATION

This section lists the experimental environment, reports the performance of the proposed algorithms and discusses the scalability of the framework.

5.1 How Performance Was Measured

This section lists the experimental environment and discusses how the performance was measured.

HPC system and environment. AIST's ABCI [3] supercomputer is used for our evaluation. Each compute node is equipped with two Intel Xeon Gold 6148 CPUs (20 Cores), 384GB memory, four Tesla V100 GPUs (16GB RAM) through PCIe gen3×16 and two InfiniBand EDR HCAs. ABCI uses CentOS 7.4 for the operating system and mounts a 6.6PB GPFS shared storage. The iFDK framework is implemented by Nvidia CUDA 9.0 (CUDA driver version: 410.104), Intel Performance Libraries 2018.2.199 (includes MPI and IPP). The compilers nvcc-9.0 and mpicc (included in MPI library) are used to compile the CUDA kernel and host code, respectively. The compiler option (-gencode arch=compute_70,code=sm_70) is applied in nvcc-9.0 for Nvidia's Volta architecture.

Measurement methodology. Since the performance of image reconstruction is independent of the content of projections or volume, we apply the standard Shepp-Logan phantom [60] to generate a variety of projections by the forward-projection tool in RTK library. An example that is reconstructed by our framework can be found in Fig. 7. We use single precision for all projections, volumes,

[3] System is ranked 8^{th} on the TOP500 list as of June 2019.

Table 3: Back-projection kernel characteristics. "Texture cache" and "L1 cache" mean accessing projections via 2D-Layered texture and L1 cache, respectively. "Transpose projection" and "Transpose volume" mean transposing the projections and volume, respectively. RTK-32 is the imrpoved RTK kernel. The other kernels are shflBP kernel (as in Listing 1) with different characteristics.

	Texture cache	L1 cache	Transpose projection	Transpose Volume
RTK-32	✓	✗	✗	✗
Bp-Tex	✓	✗	✗	✓
Tex-Tran	✓	✗	✓	✓
Bp-L1	✗	✗	✓	✓
L1-Tran	✗	✓	✓	✓

and runs. For output verification, we use the image processing tool ImageJ [1] to render the generated 3D volumes, then inspecte them manually. We also use profiled runs to investigate the density value of each voxel. Finally, we compare the output with the volumes generated by the RTK library on the CPU, the Root Mean Square Error (RMSE) is less than 10e-5. We use the OS-independent function *cudaEvent* to measure the execution time of CUDA kernels and employ *MPI_Wtime* to measure the application's runtime. Each of the reported results is averaged by 100 runs. Finally, we report the performance in the unit of GUPS, as defined in Section 2.3.

5.2 Performance of the Back-projection Kernel

This section reports the performance of the proposed back-projection kernel on Tesla V100 GPU by comparisons with a collection of kernels (listed in Table 3). The latest RTK 1.4.0 implementation (called RTK-32 in Table 4) is used with 32-bit precision (versus the default 8-bit precision). Note that our target is to instantly generate high-resolution volumes. We demonstrate that we could achieve this goal while using high image quality: *we do not sacrifice the quality by using lower precision.*

The kernel function kernel_fdk_3Dgrid of RTK is strictly implemented as defined in Algorithm 4. The original RTK limits the maximum count of projections to 16, we extend it to 32. Also, we adjust its interpolation function (Algorithm 3) to the precision of 32-bits that uses 2D-Layered cache without linear interpolation: namely using the cudaFilterModePoint parameter for the texture function. Regarding L1 cache-optimized access for projections in Table 3, the __ldg intrinsic is applied.

The characteristics of the proposed kernels are listed in Table 3. A variety of image reconstruction problems are evaluated on those kernels, the performance in GUPS is listed in Table 4. The time required to move data between host and device is not included in the execution time for the calculation of GUPS. Note that in most applications, the value of α is typically very small, often less than 1. As seen in Table 4, the proposed CUDA kernel, namely "L1-Trans", outperforms the other kernels. The performance advantage is due to the improved data locality and efficient intra-warp communication. As Table 4 shows, the size of the output of RTK cannot be bigger than 8GB since RTK employs a dual buffer technique to store the volume while the maximum memory capacity of Tesla V100 is 16GB in our testbed. By inspecting the performance in Table 4 we can observe the following: (I) When comparing the performance difference between Bp-Tex and Tex-Trans, it appears that the transpose operation of the projection has a minor effect on the hit-rate of the 2D-Layered texture cache. (II) However, the transpose operation noticeably contributes to the Bp-L1 cache hit-rate as observed by comparing the performance of Bp-L1 and L1-Trans. (III) The proposed kernel outperforms the most commonly used production library, for high-resolution image reconstruction.

Table 4: Back-projection kernel performance on Tesla V100 GPU. 1k and 2k mean 1024 and 2048, respectively. α is defined as the ratio of input to output problem size. The characteristics of evaluated CUDA kernels are listed in Table 3.

FDK poblems ($pixel \rightarrow voxel$)	α	RTK-32 (GUPS)	Bp-Tex (GUPS)	Tex-Tran (GUPS)	Bp-L1 (GUPS)	L1-Tran (GUPS)
$512^2 \times 1k \rightarrow 128^3$	128	65.3	38.8	46.5	23.7	**118.0**
$512^2 \times 1k \rightarrow 256^3$	16	107.4	96.2	98.9	28.0	**188.6**
$512^2 \times 1k \rightarrow 512^3$	2	115.1	105.8	106.1	34.0	**206.0**
$512^2 \times 1k \rightarrow (1k)^3$	1	118.1	107.3	107.3	64.9	**211.4**
$512^2 \times 1k \rightarrow (1k)^2 \times 2k$	1/8	N/A	107.4	107.6	112.1	**212.7**
$(1k)^3 \rightarrow 128^3$	512	41.9	13.8	13.5	5.7	**27.2**
$(1k)^3 \rightarrow 256^3$	64	77.4	35.9	43.2	12.8	**83.7**
$(1k)^3 \rightarrow 512^3$	8	115.7	95.5	98.1	25.1	**190.3**
$(1k)^3 \rightarrow (1k)^3$	1	117.9	105.8	105.8	34.0	**205.7**
$(1k)^3 \rightarrow (1k)^2 \times 2k$	1/2	N/A	106.3	106.5	65.0	**207.9**
$(2k)^2 \times 1k \rightarrow 128^3$	1024	16.1	5.8	8.5	2.8	**7.7**
$(2k)^2 \times 1k \rightarrow 256^3$	256	38.6	12.7	12.6	4.4	**24.1**
$(2k)^2 \times 1k \rightarrow 512^3$	32	80.2	35.5	42.5	13.9	**81.6**
$(2k)^2 \times 1k \rightarrow (1k)^3$	4	116.9	94.4	97.8	23.9	**186.9**
$(2k)^2 \times 1k \rightarrow (1k)^2 \times 2k$	1	N/A	102.9	104.1	33.4	**198.7**

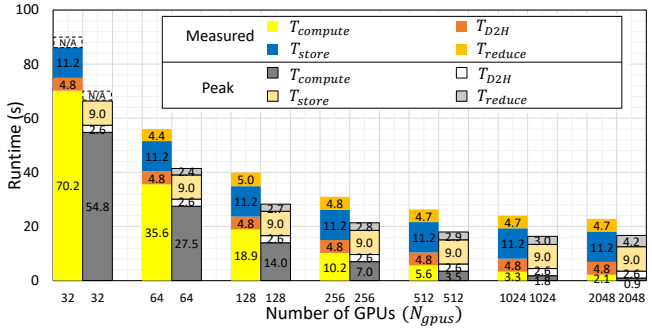
5.3 Scaling and Performance

This section presents the performance and scalability of iFDK. In the scaling experiments, each GPU computes a sub-volume of 8GB (namely N_{sub_vol}). According to Equation 7, R=32 and R=256 are used for generating volumes of 4096^3 and 8192^3, respectively. In the special case of iFDK that C=1, T_{reduce} becomes zero since the volume reduction is not required (shown as "N/A" in each sub-figure of Figure 5).

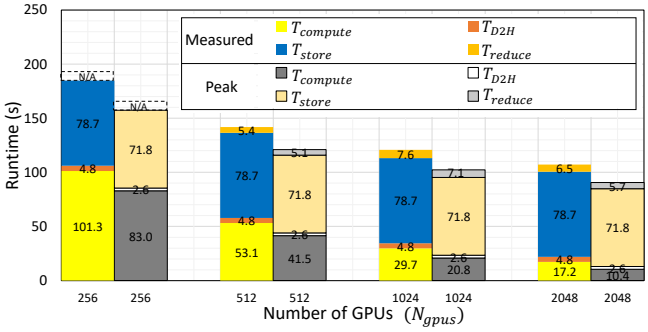
Note that we focus on discussing the impact of the volume parameters N_x, N_y, N_z and number of projections N_p on the performance and scalability in the following sections. The reason is that we cannot target weak scaling by increasing the input image sizes N_u or N_v. As Algorithm 1 shows, only the filtering computation is dependent on N_u and N_v. However, the back-projection, which is the main kernel that we scale, is independent of N_u or N_v (see Algorithm 2). This becomes clear from the performance metrics definition of GUPS as in Section 2.3, i.e. no dependency on N_u or N_v.

5.3.1 **Strong Scaling.** In Figure 5a and Figure 5b, the stacked execution time of $T_{compute}$, T_{post} (namely T_{reduce}+T_{D2H}) and T_{store} are displayed. Note that all T_{load}, T_{flt} and $T_{AllGather}$ are included in $T_{compute}$ as Equation 17 shows. Since the value of R is fixed, the value of C in iFDK increases in proportion to N_{gpus}. This results in $T_{compute}$ decreasing inversely in proportion to N_{gpus}. To give further insight of the computational behaviour, a breakdown of $T_{compute}$ is listed in Table 5. As the figure demonstrates, iFDK follows the same scaling behavior of the potential peak performance.

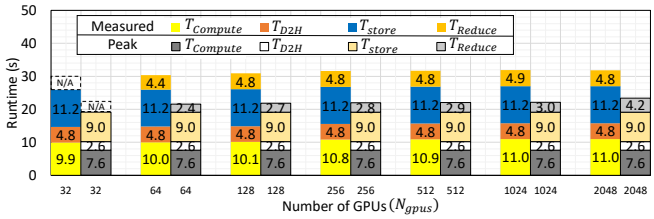
5.3.2 **Weak Scaling.** Figure 5c and Figure 5d show the weak scaling. The evaluated number of GPUs (namely N_{gpus}) is up to 2,048. In both figures, each rank loads and processes 16 and 4 projections, respectively. We use an MPI-AllGather operation to get filtered



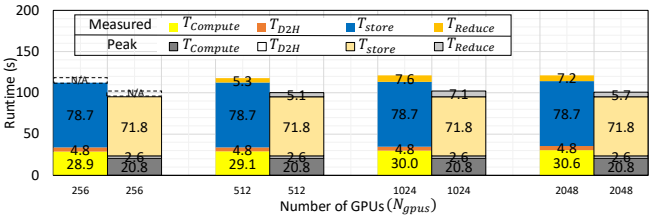
(a) **Strong scaling for** $2048^2\times4096\rightarrow4096^3$. **R=32, C=**$N_{gpus}/32$.



(b) **Strong scaling for** $2048^2\times4096\rightarrow8192^3$. **R=256, C=**$N_{gpus}/256$.



(c) **Weak scaling for** $2048^2\times N_p\rightarrow4096^3$. N_p**=16*** N_{gpus}, **R=32, C=**$N_{gpus}/32$.



(d) **Weak scaling for** $2048^2\times N_p\rightarrow8192^3$. N_p**=4*** N_{gpus}, **R=256, C=**$N_{gpus}/256$.

Figure 5: Scaling iFDK. "Reduce" denotes volume reduction. "store" indicates storing the volume to PFS. "D2H" means GPU→CPU copy. Loading projections from PFS, AllGather communication are overlapped with $T_{compute}$. CPU→GPU copy is included with $T_{compute}$. T_{reduce} is N/A when C=1 (no inter-rank reduction occurs). The high performance of the proposed back-projection kernel exposes other bottlenecks (e.g. I/O).

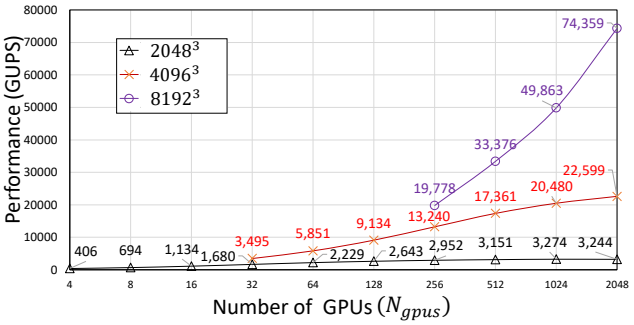


Figure 6: Performance in a unit of GUPS for problem with input size $2048^2\times4096$ **and three different output sizes** 2048^3, 4096^3 **and** 8192^3.

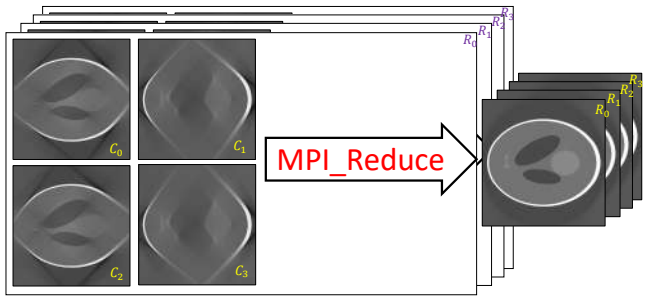


Figure 7: Results of volume reduction in an experiment done in iFDK. The problem is $2048^2\times4096\rightarrow2048^3$. **iFDK parameters are R=4 and C=4.** N_{gpus} **is 16, performance is 1,134 GUPS.**

projections from the ranks in the same column group (as in Figure 3b). In the back-projection stage, each rank processes 128 and 1024 projections, respectively.

5.3.3 **Performance.** The potential peak and achieved performances are displayed in Figure 5. Note that T_{trans} is as small as 0.29s and thus, it is included in T_{D2H} for simplifying the stacked bar figures. The peak performance is projected by our performance model. For example, the overall potential peak performance (namely $T_{runtime}$) is obtained by the benchmark values as follows. The peak bandwidth of a single PCIe×16 (namely BW_{PCIe}) is 11.9GB/s, the projected time required to copy data of size 32GB (8G*4) from device memory to the host by dual PCI-e connectors is ≈2.6s (namely T_{D2H}). The projected time required to reduce 8GB of data by dual InfiniBand per node is ≈2.7s (namely T_{reduce}). The peak sequential write bandwidth of GPFS (namely BW_{store}) is 28.5GB/s and thus, the projected time required to store data of size 256GB and 2TB is ≈ 9.0s and 87.7s (namely T_{store}), respectively.

On average, we can achieve 76% of the potential peak performance. Upon analyzing the performance gap, we found the following. For $T_{compute}$, the data exchange between the three threads orchestrating the workflow, as Figure 4 shows, can have some overhead that contributes to the gap. The overhead of intra-thread data movement within the back-projection thread also has an overhead: before copying data from host to GPU memory, it is necessary to gather at least 32 projections as a batch. In addition, memory management, building the circular buffers (as in Figure 4a) also contribute to the gap in a minor way. For T_{reduce}, we confirmed

Table 5: Details of $T_{compute}$ in Fig. 5a and Fig. 5b. T_{load} is included in T_{flt}. δ is defined as $\delta = \frac{T_{flt}+T_{AllGather}+T_{bp}}{T_{compute}}$.

volume (voxel)	N_{gpus}	N_{cpus}	T_{flt} (s)	$T_{AllGather}$ (s)	T_{bp} (s)	$T_{compute}$ (s)	δ
4096^3	32	16	1.4	31.4	54.8	70.2	**1.2**
	64	32	0.8	20.7	27.5	35.6	**1.4**
	128	64	<0.7	15.2	14.0	18.9	**1.6**
	256	128	<0.7	7.4	7.0	10.2	**1.5**
8192^3	256	128	<0.7	46.9	83.0	101.3	**1.3**
	512	256	<0.7	26.9	41.5	53.1	**1.3**
	1024	512	<0.7	17.0	20.8	29.7	**1.3**
	2048	1024	<0.7	8.6	10.4	17.2	**1.2**

that the gap is due to MPI-Reduce being called only once: the first call to the collective is typically slower, which is why benchmarks, like the one we use, pre-run few iterations before measurements. For T_{D2H}, the architecture of the compute node can cause contention on the PCIe switch feeding two GPUs (two PCIe switches for four GPUs). For T_{store}, it appears from our investigation that the minor gap is the effect of volume slices written to PFS not tuned to the ideal stripe size. Figure 6 shows the overall performance of iFDK in GUPS. We use the entire end-to-end execution time as defined in Equation 19. Figure 7 shows an example that uses MPI-Reduce primitive to generate a volume of 2048^3. iFDK scales better in generating the volume of 8192^3 than 4096^3. It is due to better device utilization in the former. This becomes clear from observing the relationship between the coefficient α and performance, in Table 4. Note that the $2048^2\times4096\rightarrow4096^3$ and $2048^2\times4096\rightarrow8192^3$ problems can be instantly solved within 30 seconds and 2 minutes, respectively. Finally, we report that the performance of the back-projection kernel on a single GPU is ≈200 GUPS (without using mixed precision), as shown in Table 3. This performance of the back-projection kernel is the main reason a time-to-solution in $O(10)$ seconds for high-resolution problems on $O(1000)$ GPUs becomes possible. It is noteworthy that all computation, running on both CPUs and GPUs, use single precision to maintain high fidelity solutions.

5.3.4 **Scaling to 8K FDK problems.** The iFDK is general to any general sizes of FDK problems, due to the two-dimensional problem decomposition methodology. At present, the common perspective in CT imaging is that processing 8192^3 volumes is highly demanded but not feasible (as discussed extensively by Martz et al. [21]). Nonetheless, we conducted experiments with iFDK on 8K volumes. Using iFDK, with 2,048 GPUs, the $2048^2\times4096\rightarrow8192^3$ problem is solved within 2 minutes. This time-to-solution is inclusive of I/O: it includes storing the volume of size *2TB* to PFS in 79s. Note that it is rare to find high precision scanners for capturing images with the quality necessary for 8K resolution, and hence it is rare to find a CT system generating the 3D volume of 8K. Yet we conducted 8K experiments for the sake of evaluating the scalability of iFDK and demonstrating that 8K image reconstruction is technically attainable.

5.3.5 **Performance Model Accuracy.** Table 5 lists the detailed execution time of each component in the framework. Our analysis of the results and the resulting observations are as follows:

(i) $\delta > 1$. This indicates that the orchestration and pipelining methodology as introduced in Section 4.1.4 is effective in improving the overall performance.

(ii) $T_{AllGather} < T_{bp}$. MPI-AllGather is employed to share the filtered projections within the C_i group as in Figure 3a, $T_{AllGather}$ can be considered as the overhead of filtering computation. Fortunately, it can be overlapped with the back-projection stage as Figure 4 shows. Furthermore, this justifies the strategy of minimizing R (as discussed in Section 4.1.5) to generate as large as possible sub-volumes on the GPUs.

6 DISCUSSION

In this section, we discuss the impact of iFDK on several real-world applications and show the platforms available for iFDK.

6.1 Relevance of iFDK to Real-world Applications

In one of the main reference textbooks of image reconstruction [21] (Section 10.7.1), Martz et al. address in detail the necessity of high-resolution image reconstruction, and give an open question about challenge of high-resolution image reconstruction (we quote): **"What happens if we start manipulating $(6k)^3$ and $(8k)^3$ volumes?"**. The work in this paper tackles this challenge by paving the way for generating high-resolution volumes instantly using algorithmic innovation and HPC best practices.

High-resolution image reconstruction is essential since it can expose more detailed information (i.e. geometry, intensity) and provide benefits to the throughput of the industrial inspection and scientific research. The remainder of this section gives concrete examples for the state-of-art innovation in commercial products for high-resolution image reconstruction.

In [17], the industrial CT (named GOM CT) is equipped with FPD of resolution 3008×2512 and is available to reconstruct 3D volumes larger than $3000^2 \times 5000$ (note the specification: the voxel is 2um~80um and the measuring area is 240mm×240mm×400mm). In [43], the CT system (named XTH450) is equipped with FPD of variant resolutions (i.e. 2000^2, 4000^2) and is built for turbine blade and casting inspection. In [62], the CT system (named inspeXio SMX-225CT) is equipped with FPD of resolution (4096^2) and is used for defect inspection.

Scientific research is also witnessing a boom of interest in high-resolution image reconstruction. For instance, Bice et al. [9] proposed rapid tomographic image reconstruction by large scale parallelization (up to 32k cores) to meet the critical demands from scientists. since they require *quasi-instant* feedback in their experiments for rapidly checking results and adjusting experimental configurations while using CT images. This instant capability is critically demanded in scanning objects with huge volumes (e.g. motor engine, human brain [44]) and complex details (e.g. the body of insects).

In the industrial field, the CT is widely used for defect inspection and reverse engineering [22, 29]. Asadizanjani et al. [5] employed micro CT for non-destructive PCB reverse engineering. The importance of high-resolution in reverse engineering is critical for huge objects with complex structures.

6.2 Platforms for iFDK

This section discusses the potential for the practical use case of iFDK in many fields, i.e. medical, industrial, and scientific. The

proposed back-projection algorithm and CUDA implementation can be applied in a number of iterative solvers (i.e. ART, MLEM, MBIR), which are popular methodologies in medical imaging for low dose image reconstruction. In addition, it can provide benefits for real-time CT systems, e.g. 4D-CT [35].

6.2.1 AWS HPC. The methods proposed by iFDK are not limited in use to top-tier HPC systems accessible only to a limited number of researchers. A simple calculation shows that generating a 4K volume as in Figure 5a can be done, for example, on Amazon's AWS HPC offerings for the cost of less than $100. That is when using 256 *p3.8xlarge* EC2 instances (with four V100 GPUs per node, similar to the system used in this paper). This uses on-demand pricing with billing timed by seconds at the price of $12.24 per hour (March 2019 US east Ohio region). This accounts for the low-performance network in AWS (10Gbps) by assuming factors of a slow down over the reported performance. Note that we do not consider moving the volumes out of the cloud since the volume visualization and data analysis can be performed in the same cloud platform. We acknowledge that policy and privacy issues would complicate the use of public clouds in medical and industrial CT. Yet we emphasize that, from the technical perspective, the methods used in iFDK make the practical and affordable instant high-resolution image reconstruction feasible.

6.2.2 Nvidia DGX-2. In order to avoid the privacy issues mentioned earlier, one could use an on-premise dense GPU box to achieve the performance reported by iFDK, for reasonably high resolution. For instance, the Nvidia DGX-2 [45] is an appropriate alternative considering it is equipped with 16 Tesla V100 GPUs, the total GPU memory of 512GB, system memory of 1.5TB, and internal storage of 30TB. Those specifications would allow iFDK, for instance, to tackle 4K problems within a minute (projected by the results shown in Figure 5a) without privacy concern. In addition, the DGX-2 has the fast NVSwitch [36] interconnect between GPUs, and a high capacity SSD: iFDK would even perform better due to improvements in the communication and I/O. Finally, it is important to mention that the price of DGX-2 is relatively low when considering the prices of high-end CT instruments.

7 RELATED WORK

Researchers have been working for a long time to optimize CT image reconstruction algorithms. Wu et al. presented an Application Specific Integrated Circuits (ASIC) solution, which is efficient in computation but expensive in development [72]. The works in [16, 27, 64, 75] applied Field-Programmable Gate Array (FPGA) to accelerate the image reconstruction computation. Programming FPGA by HDL language is complex and thus, the trend in recent years is to use high-level synthesis approaches, e.g. OpenCL [26, 40, 63]. Both Treibig et al. and Hofmann et al. optimized FDK by SIMD instruction set extensions and achieved record performance for CPUs [30, 67]. The performance engineering for *RabbitCT* [53] on the Intel Xeon Phi accelerator by Johannes et al. demonstrated a promising approach for optimizing back-projection algorithms [31]. Mostly, the prior work focused on parallelizing image reconstruction algorithms on the specified accelerators.

The prior work in [38, 77] focused on decomposing the output problems into several sub-volumes in order to solve the problem out-of-core. Wang et al. [68] used a 69,632-core distributed system to accelerate the iterative reconstruction and achieved 1,665× speed-up over the baseline. The authors in [10] use up to 6 GPU to reconstruct the volume of 2K and 4K via the FDK method. Lack of details, e.g. computational precision and I/O, complicates direct comparisons with the work mentioned above. An iterative image reconstruction algorithm called Distributed MLEM [19] scaled up to 16 GPUs on multi-GPU clusters. Hidayetoglu et. al [28] present a massively-parallel solver (iterative method) for tomographic image reconstruction, their solution scales up to 4,096 GPUs.

8 CONCLUSION

The filtered back-projection method is indispensable in most of the practical CT systems. In this work, we propose a novel and general FDK algorithm that reduces the computational cost and improves data locality. Using CUDA, we implement an efficient back-projection kernel. We further propose a distributed framework that leverages the heterogeneity in modern systems to solve high-resolution image reconstruction problems (e.g. 4K, 8K) in tens of seconds. More specifically, we optimize the filtering computation and back-projection on CPUs and GPUs, respectively. We perform the inter-node hierarchical computation by MPI collective communication primitive. The experimental results demonstrate the performance and scalability of iFDK, and also validate our performance model. For future work, we intend to investigate compression and visualization of the high-resolution volumes. We also plan to provide a real-time image reconstruction cloud service.

ACKNOWLEDGMENT

This work was partially supported by JST-CREST under Grant Number JPMJCR1687. Computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by National Institute of Advanced Industrial Science and Technology (AIST) was used. A part of the numerical calculations were carried out on the TSUBAME3.0 supercomputer at Tokyo Institute of Technology. We would like to thank Endo Lab at Tokyo Institute of Technology for providing a Tesla V100 GPU Server.

REFERENCES

[1] Michael D Abràmoff, Paulo J Magalhães, and Sunanda J Ram. 2004. Image processing with ImageJ. *Biophotonics international* 11, 7 (2004), 36–42.
[2] K. Aditya Mohan, S. V. Venkatakrishnan, J. W. Gibbs, E. B. Gulsoy, X. Xiao, M. De Graef, P. W. Voorhees, and C. A. Bouman. 2015. TIMBIR: A Method for Time-Space Reconstruction From Interlaced Views. *IEEE Transactions on Computational Imaging* 1, 2 (June 2015), 96–111.
[3] Anders H Andersen and Avinash C Kak. 1984. Simultaneous algebraic reconstruction technique (SART): a superior implementation of the ART algorithm. *Ultrasonic imaging* 6, 1 (1984), 81–94.
[4] George B Arfken and Hans J Weber. 1999. Mathematical methods for physicists.
[5] Navid Asadizanjani, Sina Shahbazmohamadi, Mark Tehranipoor, and Domenic Forte. 2015. Non-destructive pcb reverse engineering using x-ray micro computed tomography. In *41st International symposium for testing and failure analysis, ASM.* 1–5.
[6] Ammar Ahmad Awan, Ching-Hsiang Chu, Hari Subramoni, Xiaoyi Lu, and Dhabaleswar K Panda. 2018. OC-DNN: Exploiting Advanced Unified Memory Capabilities in CUDA 9 and Volta GPUs for Out-of-Core DNN Training. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC).* IEEE, 143–152.

[7] Benjamin Betz, Steffen Kie, Michael Krumm, Gunnar Knupe, Tsegaye Eshete, and Sven Simon. 2018. Efficient Data Structures for the Fast 3D Reconstruction of Voxel Volumes with Inhomogeneous Spatial Resolution. (2018).

[8] Tekin Bicer, Doga Gursoy, Raj Kettimuthu, Francesco De Carlo, Gagan Agrawal, and Ian Foster. 2015. Rapid Tomographic Image Reconstruction via Large-Scale Parallelization, Vol. 9233. 289–302. https://doi.org/10.1007/978-3-662-48096-0_23

[9] Tekin Bicer, Doga Gursoy, Rajkumar Kettimuthu, Francesco De Carlo, Gagan Agrawal, and Ian T Foster. 2015. Rapid tomographic image reconstruction via large-scale parallelization. In *European Conference on Parallel Processing*. Springer, 289–302.

[10] Javier Garcia Blas, Monica Abella, Florin Isaila, Jesus Carretero, and Manuel Desco. 2014. Surfing the optimization space of a multiple-GPU parallel implementation of a X-ray tomography reconstruction algorithm. *Journal of Systems and Software* 95 (2014), 166–175.

[11] Javier Garcia Blas, Florin Isaila, Monica Abella, Jesus Carretero, Ernesto Liria, and Manuel Desco. 2013. Parallel Implementation of a X-ray Tomography Reconstruction Algorithm Based on MPI and CUDA. In *Proceedings of the 20th European MPI Users' Group Meeting (EuroMPI '13)*. ACM, New York, NY, USA, 217–222.

[12] Ronald Newbold Bracewell. 1995. *Two-dimensional imaging*. Vol. 247. Prentice Hall Englewood Cliffs.

[13] E Oran Brigham and E Oran Brigham. 1988. *The fast Fourier transform and its applications*. Vol. 448. prentice Hall Englewood Cliffs, NJ.

[14] Peng Chen, Mohamed Wahib, Shinichiro Takizawa, Ryousei Takano, and Satoshi Matsuoka. 2018. Efficient Algorithms for the Summed Area Tables Primitive on GPUs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 482–493.

[15] Paul Cockshott and Kenneth Renfrew. 2013. *SIMD programming manual for Linux and Windows*. Springer Science & Business Media.

[16] Srdjan Coric, Miriam Leeser, Eric Miller, and Marc Trepanier. 2002. Parallel-beam backprojection: an FPGA implementation optimized for medical imaging. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. ACM, 217–226.

[17] GOM CT. 2019. Industrial Computed Tomography. https://www.gom.com/metrology-systems/gom-ct.html [Online; accessed 13-May-2019].

[18] NVIDIA CUDA. 2019. CUDA Toolkit Documentation. *NVIDIA Developer Zone. http://docs.nvidia.com/cuda/index.html* (2019).

[19] Jingyu Cui, Guillem Pratx, Bowen Meng, and Craig S Levin. 2013. Distributed MLEM: An iterative tomographic image reconstruction algorithm for distributed memory architectures. *IEEE transactions on medical imaging* 32, 5 (2013), 957–967.

[20] Philippe Despres and Xun Jia. 2017. A review of GPU-based medical image reconstruction. *Physica Medica* 42 (2017), 76–92.

[21] Harry E. Martz, Clint M. Logan, Daniel J. Schneberk, and Peter J. Shull. 2017. *X-ray imaging: fundamentals, industrial techniques, and applications*. Boca Raton: CRC Press, Taylor & Francis Group.

[22] Liyong Fang, Hui Li, Jinping Bai, and Bailin Li. 2013. Application of industrial CT in reverse engineering technology. *High Power Laser and Particle Beams* 25, 7 (2013), 1620–1624.

[23] LA Feldkamp, LC Davis, and JW Kress. 1984. Practical cone-beam algorithm. *JOSA A* 1, 6 (1984), 612–619.

[24] Richard Gordon, Robert Bender, and Gabor T Herman. 1970. Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography. *Journal of theoretical Biology* 29, 3 (1970), 471–481.

[25] Jens Gregor. 2011. Distributed CPU multi-core implementation of SIRT with vectorized matrix kernel for micro-CT. *Proceedings of the 11th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine* (2011).

[26] Devin Held. 2016. Analysis of 3D Cone-Beam CT Image Reconstruction Performance on a FPGA. (2016).

[27] I Henry and Ming Chen. 2012. *An FPGA Architecture for Real-Time 3-D Tomographic Reconstruction*. Ph.D. Dissertation. University of California, Los Angeles.

[28] M. Hidayetoglu, C. Pearson, I. El Hajj, L. Gurel, W. C. Chew, and W. Hwu. 2018. A Fast and Massively-Parallel Inverse Solver for Multiple-Scattering Tomographic Image Reconstruction. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 64–74. https://doi.org/10.1109/IPDPS.2018.00017

[29] Akira Hirakimoto. 2002. Microfocus X-ray computed tomography and it's industrial applications. In *Analytical Sciences/Supplements Proceedings of IUPAC International Congress on Analytical Sciences 2001 (ICAS 2001)*. The Japan Society for Analytical Chemistry, i123–i125.

[30] Johannes Hofmann, Jan Treibig, Georg Hager, and Gerhard Wellein. 2014. Comparing the performance of different x86 SIMD instruction sets for a medical imaging application on modern multi-and manycore chips. In *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing*. ACM, 57–64.

[31] Johannes Hofmann, Jan Treibig, Georg Hager, and Gerhard Wellein. 2014. Performance engineering for a medical imaging application on the Intel Xeon Phi accelerator. In *Architecture of Computing Systems (ARCS), 2014 Workshop Proceedings*. VDE, 1–8.

[32] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. 1995. *Machine vision*. Vol. 5. McGraw-Hill New York.

[33] Xun Jia, Bin Dong, Yifei Lou, and Steve B Jiang. 2011. GPU-based iterative cone-beam CT reconstruction using tight frame regularization. *Physics in Medicine & Biology* 56, 13 (2011), 3787.

[34] Avinash C.. Kak and Malcolm Slaney. 1988. *Principles of computerized tomographic imaging*. IEEE press New York.

[35] PJ Keall, G Starkschall, HEE Shukla, KM Forster, V Ortiz, CW Stevens, SS Vedam, R George, T Guerrero, and R Mohan. 2004. Acquiring 4D thoracic CT scans using a multislice helical method. *Physics in Medicine & Biology* 49, 10 (2004), 2053.

[36] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan Tallent, and Kevin Barker. 2019. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *arXiv preprint arXiv:1903.04611* (2019).

[37] Wenxuan Liang, Hui Zhang, and Guangshu Hu. 2010. Optimized implementation of the FDK algorithm on one digital signal processor. *Tsinghua Science and Technology* 15, 1 (2010), 108–113.

[38] Yuechao Lu, Fumihiko Ino, and Kenichi Hagihara. 2016. Cache-aware GPU optimization for out-of-core cone beam CT reconstruction of high-resolution volumes. *IEICE TRANSACTIONS on Information and Systems* 99, 12 (2016), 3060–3071.

[39] KV Manian, AA Ammar, A Ruhela, C-H Chu, H Subramoni, and DK Panda. 2019. Characterizing CUDA Unified Memory (UM)-Aware MPI Designs on Modern GPU Architectures. In *Proceedings of the 12th Workshop on General Purpose Processing Using GPUs*. ACM, 43–52.

[40] Maxime Martelli, Nicolas Gag, Alain Mérigot, and Cyrille Enderli. 2017. 3D Tomography back-projection parallelization on FPGAs using OpenCL. In *Design and Architectures for Signal and Image Processing (DASIP), 2017 Conference on*. IEEE, 1–6.

[41] Wolfram Mathworld. 2019. Convolution Theorem. http://mathworld.wolfram.com/ConvolutionTheorem.html [Online; accessed 13-May-2019].

[42] Glenn R Myers, Shane J Latham, Andrew M Kingston, Jan Kolomazník, Václav Krajíček, Tomáš Krupka, Trond K Varslot, and Adrian P Sheppard. 2016. High cone-angle x-ray computed micro-tomography with 186 gigavoxel datasets. In *Developments in X-Ray Tomography X*, Vol. 9967. International Society for Optics and Photonics, 99670U.

[43] Nikon. 2019. Computed Tomography Product. https://www.nikonmetrology.com/en-us/product/xt-h-450 [Online; accessed 13-May-2019].

[44] Anna Nowogrodzki. 2018. The world's strongest MRI machines are pushing human imaging to new limits.

[45] Nvidia. 2019. NVIDIA DGX-2. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-2-datasheet-us-nvidia-955420-r2-web-new.pdf [Online; accessed 13-May-2019].

[46] Yusuke Okitsu, Fumihiko Ino, and Kenichi Hagihara. 2010. High-performance cone beam reconstruction using CUDA compatible GPUs. *Parallel Comput.* 36, 2-3 (2010), 129–141.

[47] Willem Jan Palenstijn, Jeroen Bédorf, and K Joost Batenburg. 2015. A distributed SIRT implementation for the ASTRA toolbox. In *Proc. Fully Three-Dimensional Image Reconstruct. Radiol. Nucl. Med.* 166–169.

[48] Xiaochuan Pan, Emil Y Sidky, and Michael Vannier. 2009. Why do commercial CT scanners still employ traditional, filtered back-projection for image reconstruction? *Inverse problems* 25, 12 (2009), 123009.

[49] Sreeram Potluri, Khaled Hamidouche, Akshay Venkatesh, Devendar Bureddy, and Dhabaleswar K Panda. 2013. Efficient inter-node MPI communication using GPUDirect RDMA for InfiniBand clusters with NVIDIA GPUs. In *2013 42nd International Conference on Parallel Processing*. IEEE, 80–89.

[50] CT ProScan. 2019. Reverse Engineering Services. https://3dproscan.com/industrial-ct-scanning-services-reverse-engineering/.

[51] N Rezvani, D Aruliah, K Jackson, D Moseley, and J Siewerdsen. 2007. SU-FF-I-16: OSCaR: An open-source cone-beam CT reconstruction tool for imaging research. *Medical Physics* 34, 6Part2 (2007), 2341–2341.

[52] Simon Rit, M Vila Oliva, Sébastien Brousmiche, Rudi Labarbe, David Sarrut, and Gregory C Sharp. 2014. The Reconstruction Toolkit (RTK), an open-source cone-beam CT reconstruction toolkit based on the Insight Toolkit (ITK). In *Journal of Physics: Conference Series*, Vol. 489. IOP Publishing, 012079.

[53] Christopher Rohkohl, Benjamin Keck, HG Hofmann, and Joachim Hornegger. 2009. RabbitCT-an open platform for benchmarking 3D cone-beam reconstruction algorithms. *Medical Physics* 36, 9Part1 (2009), 3940–3944.

[54] Jeffrey M Rosen, Junjie Wu, TF Wenisch, and JA Fessler. 2013. Iterative helical CT reconstruction in the cloud for ten dollars in five minutes. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med.* 241–4.

[55] Amit Sabne, Xiao Wang, Sherman J Kisner, Charles A Bouman, Anand Raghunathan, and Samuel P Midkiff. 2017. Model-based iterative CT image reconstruction on GPUs. *ACM SIGPLAN Notices* 52, 8 (2017), 207–220.

[56] Ing Karl Schwarz. 2011. The Texture Unit as Performance Booster in CT-Reconstruction. In *NVIDIA Tesla GPU Computing at the International Supercomputing Conference 2011*.

[57] Estefania Serrano, Guzman Bermejo, Javier Garcia Blas, and Jesus Carretero. 2014. High-performance X-ray tomography reconstruction algorithm based on heterogeneous accelerated computing systems. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 331–338.

[58] Hongzhang Shan and John Shalf. 2007. Using IOR to analyze the I/O performance for HPC platforms. (2007).
[59] GC Sharp, N Kandasamy, H Singh, and Michael Folkert. 2007. GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration. *Physics in Medicine & Biology* 52, 19 (2007), 5771.
[60] Lawrence A Shepp and Benjamin F Logan. 1974. The Fourier reconstruction of a head section. *IEEE Transactions on nuclear science* 21, 3 (1974), 21–43.
[61] Lawrence A Shepp and Yehuda Vardi. 1982. Maximum likelihood reconstruction for emission tomography. *IEEE transactions on medical imaging* 1, 2 (1982), 113–122.
[62] CT shimadzu. 2019. High Resolution CT Image. https://www.ssi.shimadzu.com/products/non-destructive-x-ray-testing/inspexio-smx-225ct-features.html.
[63] Christian Siegl, HG Hofmann, B Keck, M Prümmer, and J Hornegger. 2011. OpenCL: a viable solution for high-performance medical image reconstruction?. In *Medical Imaging 2011: Physics of Medical Imaging*, Vol. 7961. International Society for Optics and Photonics, 79612Q.
[64] Nikhil Subramanian. 2009. *A C-to-FPGA solution for accelerating tomographic reconstruction.* Ph.D. Dissertation. University of Washington.
[65] Stewart Taylor. 2007. *Optimizing applications for multi-core processors, using the Intel integrated performance primitives.* Intel Press.
[66] Darren Thompson, Ya I Nesterets, TE Gureyev, Arthur Sakellariou, Alex Khassapov, and John Taylor. 2011. Rapid CT reconstruction on GPU-enabled HPC clusters. *MODSIM 2011, December 12-16 2011, Perth, Australia* (2011).
[67] Jan Treibig, Georg Hager, Hannes G Hofmann, Joachim Hornegger, and Gerhard Wellein. 2013. Pushing the limits for medical image reconstruction on recent standard multicore processors. *The International Journal of High Performance Computing Applications* 27, 2 (2013), 162–177.
[68] Xiao Wang, Amit Sabne, Putt Sakdhnagool, Sherman J Kisner, Charles A Bouman, and Samuel P Midkiff. 2017. Massively parallel 3D image reconstruction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 3.
[69] Karl Wiesent, Karl Barth, Nassir Navab, Peter Durlak, Thomas Brunner, Oliver Schuetz, and Wolfgang Seissler. 2000. Enhanced 3-D-reconstruction algorithm for C-arm systems suitable for interventional procedures. *IEEE transactions on medical imaging* 19, 5 (2000), 391–403.
[70] Wikipedia. 2019. Circular buffer — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Circular%20buffer&oldid=888744485. [Online; accessed 02-April-2019].
[71] Wikipedia. 2019. Gantry (medical) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Gantry%20(medical)&oldid=881118523. [Online; accessed 23-March-2019].
[72] Michael A Wu. 1991. ASIC applications in computed tomography systems. In *ASIC Conference and Exhibit, 1991. Proceedings., Fourth Annual IEEE International*. IEEE, P1–3.
[73] Fang Xu and Klaus Mueller. 2005. Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware. *IEEE Transactions on nuclear science* 52, 3 (2005), 654–663.
[74] Fang Xu and Klaus Mueller. 2007. Real-time 3D computed tomographic reconstruction using commodity graphics hardware. *Physics in Medicine & Biology* 52, 12 (2007), 3405.
[75] Xinwei Xue, Arvi Cheryauka, and David Tubbs. 2006. Acceleration of fluoro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: a simulation study. In *Medical Imaging 2006: Physics of Medical Imaging*, Vol. 6142. International Society for Optics and Photonics, 61424L.
[76] Jiansheng Yang, Xiaohu Guo, Qiang Kong, Tie Zhou, and Ming Jiang. 2006. Parallel implementation of Katsevich's FBP algorithm. *International journal of biomedical imaging* 2006 (2006).
[77] Xing Zhao, Jing-jing Hu, and Peng Zhang. 2009. GPU-based 3D cone-beam CT image reconstruction for large data volume. *Journal of Biomedical Imaging* 2009 (2009), 8.
[78] Timo Zinsser and Benjamin Keck. 2013. Systematic performance optimization of cone-beam back-projection on the Kepler architecture. *Proceedings of the 12th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine* (2013), 225–228.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Our framework, namely iFDK, is implemented by Nvidia CUDA 9.0, Intel Performance Libraries 2018.2.199 (includes MPI and IPP). Our implementation is evaluated on ABCI supercomputer (https://abci.ai/ja/), which is ranked 8th on the TOP500 list as of June 2019. Each compute node is equipped with two Intel Xeon Gold 6148 CPUs (20 Cores, hyper-threading is enabled), 384GB memory, four Tesla V100 GPUs (16GB RAM) through PCIe gen3×16 and two InfiniBand EDR HCAs. ABCI uses CentOS 7.4 for the operating system and mounts three 6.6PB GPFS shared storage.

We apply the standard Shepp-Logan phantom to generate a variety of projections by the forward-projection tool in RTK 1.4.0 library. We use single-precision for all projections, volumes, and runs.

All of the results reported in this paper can be generated by our scripts as follows: (1) Get our source code from our GitHub and build all of the binaries (namely ./build.sh). (2) Generate a 3D Shepp-Logan phantom by the Matlab script. (3) In the Root-folder, run the script (./generate-projections.sh) to generate the projections. Note that the input is t 3D Shepp-Logan phantom path, the output is a set of 2D projections. (4) In the Root-folder, run script (./run.sh) to get all result in Figure 5, Figure 6, Figure 7 and Table 5. (5)login to a single compute node (e.g. rt_F=1 option is adopted by the qrsh command), run script (./eva-kernel.sh) to generate the result of kernel performance as in Table 4. Note that, we use global storage to store the projections, and output of each computation. The storage size we used is 10TB. The maximal number of GPUs we can use in our experiment is 2048 (namely 512 compute nodes).

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: There are no author-created data artifacts.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

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

`https://github.com/pengdada/iFDK-sc19`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: AI Bridging Cloud Infrastructure(ABCI)

Operating systems and versions: CentOS 7.4 running Linux kernel 3.10.0

Compilers and versions: gcc 4.8.5, nvcc 9.0, intel mpicc/icc(2018.2.199), cmake 3.13.1

Applications and versions: RTK 1.4.0, ITK-4.12.2, Nvidia bandwidthTest, python 2.7, IOR version 3.0.0, Matlab R2018a

Libraries and versions: IntelMPI (2018.2.199), Intel Integrated Performance Primitives (IPP 2018.2.199), CUDA 9.0

Key algorithms: FDK, convolution, bilinear interpolation, back-projection

Input datasets and versions: Shepp-Logan phantom

Paper Modifications: (1) The Os is CentOS 7.4, CUDA Driver Version: 410.104, ECC option is ON. (2) The original FDK (also called FBP) implementation is from RTK 1.4.0, which is open-source and cross-platform software for fast circular cone-beam CT reconstruction. We improve RTK1.4.0 back-projection kernel from mixed precision to single precision in our experiment. gcc 4.8.5, cmake 3.13.1 are used to build RTK1.4.0 library. RTK is depended on InsightToolkit (we use v4.12.2). The forward projection module is based on RTK140 library, we use forward projection tool to generate a set of 2D projections. (3) The Nvidia tool, called bandwidthTest, is used for measuring the peak bandwidth of a single PCI-e connector. The bandwidthTest can be found in the Sample of CUDA Toolkit. (4) We use MPI benchmark to measure the peak throughput of MPI communication (5) The OS-independent function, cudaEvent, is employed to measure the performance of the back-projection kernel. The MPI-Time function is used to measure the execution time. (6) Our framework is developed by CUDA 9.0 Toolkit, which is not the latest version but is stable in our experiment. To build our CUDA kernels, the compiler option (-gencode arch=compute_70,code=sm_70) is employed. (7) The description on Shepp-Logan phantom can be found in WIKIPEDIA as (https://en.wikipedia.org/wiki/Shepp%E2%80%93Logan_phantom)

Output from scripts that gathers execution environment information.

```
MANPATH=/bb/system/uge/latest/man:/usr/share/man:/us⌋
↪ r/local/share/man
_fzf_orig_completion_tee=complete -F %s tee #_longopt
_fzf_orig_completion_diff=complete -F %s diff
↪ #_longopt
SGE_TASK_STEPSIZE=undefined
SGE_JOB_SERVICE_TYPE=On-demand
SGE_POSIX_PRIORITY=-450
SHELL=/bin/bash
TERM=xterm
NHOSTS=1
HISTSIZE=1000
TMPDIR=/tmp/493672.1.gpu
SGE_O_WORKDIR=/home/USER/sc19/doc
_fzf_orig_completion_mv=complete -F %s mv #_longopt
SGE_O_HOME=/home/USER
```

```
SGE_CELL=abci
SGE_ARCH=lx-amd64
QTDIR=/usr/lib64/qt-3.3
_fzf_orig_completion_du=complete -F %s du #_longopt
SGE_HGR_rt_Clarge=0
QTINC=/usr/lib64/qt-3.3/include
_fzf_orig_completion_bunzip2=complete -F %s bunzip2
↪  #_filedir_xspec
RESTARTED=0
_fzf_orig_completion_less=complete -F %s less
↪  #_longopt
SGE_CLIENT_NAME=qrsh
_fzf_orig_completion_tail=complete -F %s tail
↪  #_longopt
_fzf_orig_completion_rmdir=complete -F %s rmdir
↪  #_longopt
_fzf_orig_completion_head=complete -F %s head
↪  #_longopt
ARC=lx-amd64
QT_GRAPHICSSYSTEM_CHECKED=1
SGE_HGR_rt_Glarge=0
USER=USER
_fzf_orig_completion_telnet=complete -F %s telnet
↪  #_known_hosts
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0
↪  1;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;0
↪  1:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=3
↪  0;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tg
↪  z=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lh
↪  a=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.t
↪  lz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.z
↪  ip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;
↪  31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31
↪  :*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
↪  :*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:
↪  *.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:
↪  *.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31
↪  :*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*
↪  .jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:
↪  *.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:
↪  *.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35
↪  :*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;3
↪  5:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;
↪  35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01
↪  ;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01
↪  ;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;
↪  35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01
↪  ;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;3
↪  5:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;3
↪  5:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;3
↪  5:*.aac=01;36:*.au=01;36:*.flac=01;36:*.mid=01;3
↪  6:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;
↪  36:*.ogg=01;36:*.ra=01;36:*.wav=01;36:*.axa=01;3
↪  6:*.oga=01;36:*.spx=01;36:*.xspf=01;36:
LD_LIBRARY_PATH=/home/USER/local/gdb-8.2/lib:/home/U
↪  SER/local/gcc-5.4/lib64:/home/USER/local/bison-3
↪  .0.4/lib:/home/USER/local/llvm-7.0.1/lib:/home/U
↪  SER/local/tmux/lib:
INTEL_HOME=/apps/intel/2018.2/compilers_and_librarie
↪  s_2018.2.199/linux
SGE_HGR_TASK_rt_Glarge=0
_fzf_orig_completion_wc=complete -F %s wc #_longopt
SGE_TASK_LAST=undefined
QUEUE=gpu
_fzf_orig_completion_ftp=complete -F %s ftp
↪  #_known_hosts
_fzf_orig_completion_view=complete -F %s view
↪  #_filedir_xspec
SGE_WALLTIME_SEC=14400
SGE_HGR_rt_Gsmall=0 1 2 3
LOCAL_HOME=/home/USER/local
_fzf_orig_completion_grep=complete -F %s grep
↪  #_longopt
SGE_RESOURCE_NUM=1
SGE_HGR_TASK_rt_F=0
SGE_TASK_ID=undefined
SGE_JOB_HOSTLIST=/var/spool/uge/execd/g0008/active_j
↪  obs/493672.1/hostlist
_fzf_orig_completion_gvim=complete -F %s gvim
↪  #_filedir_xspec
SGE_HGR_rt_F=0
SGE_BINARY_PATH=/bb/system/uge/8.6.3/bin/lx-amd64
PATH=/home/USER/local/gdb-8.2/bin:/home/USER/local/c
↪  tages-5.8/bin:/home/USER/local/bash-stepping-xtr
↪  ace:/home/USER/local/bison-3.0.4/bin:/home/USER/
↪  local/llvm-7.0.1/bin:/home/USER/local/bin:/home/
↪  USER/local/sshpass-1.05:/home/USER/local/htop-2.
↪  2.0:/home/USER/local/cmake-3.13.1/bin:/home/USER
↪  /local/tmux/bin:/usr/lib64/qt-3.3/bin:/bin:/usr/
↪  bin:/usr/local/sbin:/usr/sbin:/bb/system/uge/lat
↪  est/bin/lx-amd64:/fs3/home/USER/.fzf/bin:/home/U
↪  SER/.local/bin:/home/USER/bin
MAIL=/var/spool/mail/USER
_fzf_orig_completion_unzip=complete -F %s unzip
↪  #_filedir_xspec
_=/bin/env
SGE_HGR_TASK_rt_Gsmall=0 1 2 3
SGE_STDERR_PATH=/dev/null
PWD=/home/USER/sc19/doc/Author-Kit
GDB_HOME=/home/USER/local/gdb-8.2
_fzf_orig_completion_ls=complete -F %s ls #_longopt
_fzf_orig_completion_rm=complete -F %s rm #_longopt
_fzf_orig_completion_uniq=complete -F %s uniq
↪  #_longopt
SGE_EXECD_PORT=16445
SGE_RSH_COMMAND=builtin
SGE_ACCOUNT=gaa50004
SGE_STDOUT_PATH=/dev/null
_fzf_orig_completion_cat=complete -F %s cat #_longopt
LANG=en_US.UTF-8
```

```
SGE_QMASTER_PORT=16444
SGE_HGR_rt_Csmall=0 1 2 3
JOB_NAME=QRLOGIN
JOB_SCRIPT=QRLOGIN
SGE_RERUN_REQUESTED=2
MODULEPATH=/apps/modules/modulefiles/compilers:/apps
↪ /modules/modulefiles/devtools:/apps/modules/modu
↪ lefiles/gpgpu:/apps/modules/modulefiles/mpi:/app
↪ s/modules/modulefiles/runtimes
SGE_ROOT=/bb/system/uge/latest
LOADEDMODULES=
SGE_HGR_TASK_rt_Clarge=0
_fzf_orig_completion_cd=complete -o nospace -F %s cd
↪ #_cd
SGE_RESOURCE_TYPE=rt_F
REQNAME=QRLOGIN
USE_SYSTEMD=true
_fzf_orig_completion_vi=complete -F %s vi
↪ #_filedir_xspec
ENVIRONMENT=BATCH
SGE_JOB_SPOOL_DIR=/var/spool/uge/execd/g0008/active_
↪ jobs/493672.1
HISTCONTROL=ignoredups
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
PE_HOSTFILE=/var/spool/uge/execd/g0008/active_jobs/4
↪ 93672.1/pe_hostfile
HOME=/home/USER
SHLVL=2
NQUEUES=1
SGE_CWD_PATH=/home/USER
SGE_O_LOGNAME=USER
SGE_RERUN_JOB=0
SGE_O_MAIL=/var/spool/mail/USER
SGE_HGR_TASK_rt_Csmall=0 1 2 3
JOB_ID=493672
LOGNAME=USER
FFTW_HOME=/home/USER/local/fftw-3.3.8
_fzf_orig_completion_vim=complete -F %s vim
↪ #_filedir_xspec
SGE_LOCALDIR=/local/493672.1.gpu
CVS_RSH=ssh
QTLIB=/usr/lib64/qt-3.3/lib
_fzf_orig_completion_awk=complete -F %s awk #_longopt
XDG_DATA_DIRS=/home/USER/.local/share/flatpak/export
↪ s/share/:/var/lib/flatpak/exports/share/:/usr/lo
↪ cal/share/:/usr/share/
_fzf_orig_completion_ld=complete -F %s ld #_longopt
_fzf_orig_completion_sort=complete -F %s sort
↪ #_longopt
SGE_TASK_FIRST=undefined
MODULESHOME=/usr/share/Modules
_fzf_orig_completion_gunzip=complete -F %s gunzip
↪ #_filedir_xspec
LESSOPEN=||/usr/bin/lesspipe.sh %s
SGE_O_PATH=/home/USER/local/gdb-8.2/bin:/home/USER/l
↪ ocal/ctages-5.8/bin:/home/USER/local/bash-steppi
↪ ng-xtrace:/home/USER/local/bison-3.0.4/bin:/home
↪ /USER/local/llvm-7.0.1/bin:/home/USER/local/bin:
↪ /home/USER/local/sshpass-1.05:/home/USER/local/h
↪ top-2.2.0:/home/USER/local/cmake-3.13.1/bin:/hom
↪ e/USER/local/tmux/bin:/usr/lib64/qt-3.3/bin:/usr
↪ /local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/b
↪ b/system/uge/latest/bin/lx-amd64:/fs3/home/USER/
↪ .fzf/bin:/home/USER/.local/bin:/home/USER/bin
SGE_O_SHELL=/bin/bash
SGE_O_HOST=es1
SGE_CLUSTER_NAME=abci
_fzf_orig_completion_emacs=complete -F %s emacs
↪ #_filedir_xspec
_fzf_orig_completion_hg=complete -o bashdefault -o
↪ default -F %s hg #_hg
REQUEST=QRLOGIN
LLVM_HOME=/home/USER/local/llvm-7.0.1
_fzf_orig_completion_cp=complete -F %s cp #_longopt
NSLOTS=80
SGE_STDIN_PATH=/dev/null
_fzf_orig_completion_sed=complete -F %s sed #_longopt
_fzf_orig_completion_ln=complete -F %s ln #_longopt
_fzf_orig_completion_git=complete -o bashdefault -o
↪ default -o nospace -F %s git #__git_wrap__git_main
BASH_FUNC_module()=() {  eval `/usr/bin/modulecmd
↪ bash $*`
}
Linux g0008.abci.local 3.10.0-862.el7.x86_64 #1 SMP
↪ Fri Apr 20 16:44:24 UTC 2018 x86_64 x86_64 x86_64
↪ GNU/Linux
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                80
On-line CPU(s) list:   0-79
Thread(s) per core:    2
Core(s) per socket:    20
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6148 CPU
↪ @ 2.40GHz
Stepping:              4
CPU MHz:               1000.048
CPU max MHz:           3700.0000
CPU min MHz:           1000.0000
BogoMIPS:              4800.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              28160K
```

```
NUMA node0 CPU(s):     0-19,40-59
NUMA node1 CPU(s):     20-39,60-79
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 constant_tsc art 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 3dnowprefetch epb cat_l3 cdp_l3
↪  intel_ppin intel_pt mba tpr_shadow vnmi
↪  flexpriority ept vpid fsgsbase tsc_adjust bmi1
↪  hle avx2 smep bmi2 erms invpcid rtm cqm mpx rdt_a
↪  avx512f avx512dq rdseed adx smap clflushopt clwb
↪  avx512cd avx512bw avx512vl xsaveopt xsavec
↪  xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total
↪  cqm_mbm_local dtherm ida arat pln pts hwp
↪  hwp_act_window hwp_epp hwp_pkg_req pku ospke
MemTotal:       394480496 kB
MemFree:        380100844 kB
MemAvailable:   382007356 kB
Buffers:            8044 kB
Cached:          3315036 kB
SwapCached:            0 kB
Active:          2669204 kB
Inactive:        1922824 kB
Active(anon):    1674780 kB
Inactive(anon):   225340 kB
Active(file):     994424 kB
Inactive(file):  1697484 kB
Unevictable:     4194304 kB
Mlocked:         4194304 kB
SwapTotal:       4200960 kB
SwapFree:        4200960 kB
Dirty:                76 kB
Writeback:             0 kB
AnonPages:       5463380 kB
Mapped:           582100 kB
Shmem:            631160 kB
Slab:            1097352 kB
SReclaimable:     352664 kB
SUnreclaim:       744688 kB
KernelStack:       67520 kB
PageTables:        22536 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    201441208 kB
Committed_AS:    8091168 kB
VmallocTotal:   34359738367 kB
VmallocUsed:     1808208 kB
VmallocChunk:   34156384196 kB
HardwareCorrupted:     0 kB
AnonHugePages:   5064704 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:     6613660 kB
DirectMap2M:    225695744 kB
DirectMap1G:    170917888 kB
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda       8:0    0 119.2G  0 disk
├─sda1    8:1    0     2G  0 part /boot
├─sda2    8:2    0    16G  0 part /
├─sda3    8:3    0    48G  0 part /var
├─sda4    8:4    0     1K  0 part
├─sda5    8:5    0     4G  0 part [SWAP]
├─sda6    8:6    0    32G  0 part /var/dumpsavearea
└─sda7    8:7    0  17.2G  0 part /tmp
loop0     7:0    0         1 loop
loop1     7:1    0         1 loop
nvme0n1 259:0    0   1.5T  0 disk /local
[2:0:0:0]    disk    ATA      M.2 (S42) 3ME4   01U
↪  /dev/sda    128GB
Wed Apr 10 13:21:04 2019
+-----------------------------------------------┐
↪  --------------------------+
| NVIDIA-SMI 410.104      Driver Version: 410.104
↪  CUDA Version: 10.0     |
|-------------------------------+------------------┐
↪  ---+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A
↪  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage
↪  | GPU-Util  Compute M. |
|===============================+==================┐
↪  ===+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:3D:00.0 Off
↪  |                    0 |
| N/A   29C    P0    43W / 300W |      0MiB / 16130MiB
↪  |      0%      Default |
+-------------------------------+------------------┐
↪  ---+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:3E:00.0 Off
↪  |                    0 |
| N/A   28C    P0    41W / 300W |      0MiB / 16130MiB
↪  |      0%      Default |
+-------------------------------+------------------┐
↪  ---+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:B1:00.0 Off
↪  |                    0 |
| N/A   27C    P0    46W / 300W |      0MiB / 16130MiB
↪  |      0%      Default |
+-------------------------------+------------------┐
↪  ---+----------------------+
```

```
|   3  Tesla V100-SXM2...  On   | 00000000:B2:00.0 Off
↪  |                    0 |
| N/A   29C    P0    44W / 300W |      0MiB / 16130MiB
↪  |      0%      Default |
+-------------------------------+----------------------┐
↪  ---+----------------------+

+-----------------------------------------------------┐
↪  --------------------------+
| Processes:
↪             GPU Memory |
|  GPU       PID   Type   Process name
↪              Usage      |
|=====================================================┐
↪  ==========================|
|  No running processes found
↪                          |
+-----------------------------------------------------┐
↪  --------------------------+
H/W path       Device  Class          Description
================================================
                       system         Computer
/0                     bus            Motherboard
/0/0                   memory         376GiB System
↪  memory
/0/2                   processor      Intel(R) Xeon(R)
↪  Gold 6148 CPU @ 2.40GHz
/0/3                   processor      Intel(R) Xeon(R)
↪  Gold 6148 CPU @ 2.40GHz
/0/100                 bridge         Sky Lake-E DMI3
↪  Registers
/0/100/4               generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.1             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.2             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.3             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.4             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.5             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.6             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/4.7             generic        Sky Lake-E CBDMA
↪  Registers
/0/100/5               generic        Sky Lake-E
↪  MM/Vt-d Configuration Registers
/0/100/5.2             generic        Intel Corporation
/0/100/8               generic        Sky Lake-E Ubox
↪  Registers
/0/100/8.1             generic        Sky Lake-E Ubox
↪  Registers
/0/100/8.2             generic        Sky Lake-E Ubox
↪  Registers
/0/100/11              generic        Intel Corporation
/0/100/11.5            storage        Lewisburg SSATA
↪  Controller [AHCI mode]
/0/100/14              bus            Lewisburg USB 3.0
↪  xHCI Controller
/0/100/14.2            generic        Lewisburg
↪  Thermal Subsystem
/0/100/16              communication  Lewisburg CSME:
↪  HECI #1
/0/100/16.1            communication  Lewisburg CSME:
↪  HECI #2
/0/100/16.4            communication  Lewisburg CSME:
↪  HECI #3
/0/100/17              storage        Lewisburg SATA
↪  Controller [AHCI mode]
/0/100/1c              bridge         Lewisburg PCI
↪  Express Root Port #1
/0/100/1c/0            bridge         x1 PCIe Gen2
↪  Bridge[Pilot4]
/0/100/1c/0/0          display        MGA G200e
↪  [Pilot] ServerEngines (SEP1)
/0/100/1c/0/1          processor      Fujitsu
↪  Technology Solutions
/0/100/1c.4            bridge         Lewisburg PCI
↪  Express Root Port #5
/0/100/1c.4/0  eth0    network        I210 Gigabit
↪  Network Connection
/0/100/1f              bridge         Lewisburg LPC
↪  Controller
/0/100/1f.2            memory         Memory controller
/0/100/1f.4            bus            Lewisburg SMBus
/0/100/1f.5            bus            Lewisburg SPI
↪  Controller
/0/101                 bridge         Sky Lake-E PCI
↪  Express Root Port A
/0/101/0       ib0     network        MT27700 Family
↪  [ConnectX-4]
/0/6                   generic        Intel Corporation
/0/7                   generic        Sky Lake-E RAS
↪  Configuration Registers
/0/9                   generic        Sky Lake-E CHA
↪  Registers
/0/a                   generic        Sky Lake-E CHA
↪  Registers
/0/b                   generic        Sky Lake-E CHA
↪  Registers
/0/c                   generic        Sky Lake-E CHA
↪  Registers
/0/d                   generic        Sky Lake-E CHA
↪  Registers
/0/e                   generic        Sky Lake-E CHA
↪  Registers
```

```
/0/f                    generic         Sky Lake-E CHA
↪ Registers
/0/10                   generic         Sky Lake-E CHA
↪ Registers
/0/11                   generic         Sky Lake-E CHA
↪ Registers
/0/12                   generic         Sky Lake-E CHA
↪ Registers
/0/13                   generic         Sky Lake-E CHA
↪ Registers
/0/14                   generic         Sky Lake-E CHA
↪ Registers
/0/15                   generic         Sky Lake-E CHA
↪ Registers
/0/16                   generic         Sky Lake-E CHA
↪ Registers
/0/17                   generic         Sky Lake-E CHA
↪ Registers
/0/18                   generic         Sky Lake-E CHA
↪ Registers
/0/19                   generic         Sky Lake-E CHA
↪ Registers
/0/1a                   generic         Sky Lake-E CHA
↪ Registers
/0/1b                   generic         Sky Lake-E CHA
↪ Registers
/0/1c                   generic         Sky Lake-E CHA
↪ Registers
/0/1d                   generic         Sky Lake-E CHA
↪ Registers
/0/1e                   generic         Sky Lake-E CHA
↪ Registers
/0/1f                   generic         Sky Lake-E CHA
↪ Registers
/0/20                   generic         Sky Lake-E CHA
↪ Registers
/0/21                   generic         Sky Lake-E CHA
↪ Registers
/0/22                   generic         Sky Lake-E CHA
↪ Registers
/0/23                   generic         Sky Lake-E CHA
↪ Registers
/0/24                   generic         Sky Lake-E CHA
↪ Registers
/0/25                   generic         Sky Lake-E CHA
↪ Registers
/0/26                   generic         Sky Lake-E CHA
↪ Registers
/0/27                   generic         Sky Lake-E CHA
↪ Registers
/0/28                   generic         Sky Lake-E CHA
↪ Registers
/0/29                   generic         Sky Lake-E CHA
↪ Registers
/0/2a                   generic         Sky Lake-E CHA
↪ Registers
/0/2b                   generic         Sky Lake-E CHA
↪ Registers
/0/2c                   generic         Sky Lake-E CHA
↪ Registers
/0/2d                   generic         Sky Lake-E CHA
↪ Registers
/0/2e                   generic         Sky Lake-E CHA
↪ Registers
/0/2f                   generic         Sky Lake-E CHA
↪ Registers
/0/30                   generic         Sky Lake-E CHA
↪ Registers
/0/31                   generic         Sky Lake-E CHA
↪ Registers
/0/32                   generic         Sky Lake-E CHA
↪ Registers
/0/33                   generic         Sky Lake-E CHA
↪ Registers
/0/34                   generic         Sky Lake-E CHA
↪ Registers
/0/35                   generic         Sky Lake-E CHA
↪ Registers
/0/36                   generic         Sky Lake-E CHA
↪ Registers
/0/37                   generic         Sky Lake-E CHA
↪ Registers
/0/38                   generic         Sky Lake-E CHA
↪ Registers
/0/39                   generic         Sky Lake-E CHA
↪ Registers
/0/3a                   generic         Sky Lake-E CHA
↪ Registers
/0/3b                   generic         Sky Lake-E CHA
↪ Registers
/0/3c                   generic         Sky Lake-E CHA
↪ Registers
/0/3d                   generic         Sky Lake-E CHA
↪ Registers
/0/3e                   generic         Sky Lake-E CHA
↪ Registers
/0/3f                   generic         Sky Lake-E CHA
↪ Registers
/0/40                   generic         Sky Lake-E CHA
↪ Registers
/0/41                   generic         Sky Lake-E CHA
↪ Registers
/0/42                   generic         Sky Lake-E CHA
↪ Registers
/0/43                   generic         Sky Lake-E CHA
↪ Registers
/0/44                   generic         Sky Lake-E CHA
↪ Registers
```

```
/0/45                 generic        Sky Lake-E PCU
↪ Registers
/0/46                 generic        Sky Lake-E PCU
↪ Registers
/0/47                 generic        Sky Lake-E PCU
↪ Registers
/0/48                 generic        Sky Lake-E PCU
↪ Registers
/0/49                 generic        Sky Lake-E PCU
↪ Registers
/0/4a                 generic        Sky Lake-E PCU
↪ Registers
/0/4b                 generic        Sky Lake-E PCU
↪ Registers
/0/102                bridge         Sky Lake-E PCI
↪ Express Root Port A
/0/102/0              bridge         PEX 8747 48-Lane,
↪ 5-Port PCI Express Gen 3 (8.0 GT/s) Switch
/0/102/0/0            bridge         PEX 8747 48-Lane,
↪ 5-Port PCI Express Gen 3 (8.0 GT/s) Switch
/0/102/0/0/0           display        GV100GL [Tesla
↪ V100 SXM2]
/0/102/0/10           bridge         PEX 8747 48-Lane,
↪ 5-Port PCI Express Gen 3 (8.0 GT/s) Switch
/0/102/0/10/0          display        GV100GL [Tesla
↪ V100 SXM2]
/0/4c                 generic        Intel Corporation
/0/4d                  generic        Sky Lake-E RAS
↪ Configuration Registers
/0/4e                 generic        Intel Corporation
/0/4f                 generic        Intel Corporation
/0/50                 generic        Intel Corporation
/0/51                 generic        Intel Corporation
/0/52                 generic        Intel Corporation
/0/53                 generic        Intel Corporation
/0/54                 generic        Intel Corporation
/0/55                 generic        Intel Corporation
/0/56                 generic        Intel Corporation
/0/57                 generic        Intel Corporation
/0/58                 generic        Intel Corporation
/0/59                 generic        Intel Corporation
/0/5a                 generic        Intel Corporation
/0/5b                 generic        Intel Corporation
/0/5c                 generic        Intel Corporation
/0/5d                 generic        Intel Corporation
/0/5e                 generic        Intel Corporation
/0/5f                 generic        Intel Corporation
/0/60                 generic        Intel Corporation
/0/61                 generic        Intel Corporation
/0/62                 generic        Intel Corporation
/0/63                 generic        Intel Corporation
/0/64                 generic        Intel Corporation
/0/65                 generic        Intel Corporation
/0/66                 generic        Intel Corporation
/0/67                 generic        Intel Corporation
/0/103                bridge         Sky Lake-E PCI
↪ Express Root Port A
/0/68                 generic        Intel Corporation
/0/69                  generic        Sky Lake-E RAS
↪ Configuration Registers
/0/6a                 generic        Intel Corporation
/0/6b                 generic        Intel Corporation
/0/6c                 generic        Intel Corporation
/0/6d                 generic        Intel Corporation
/0/6e                 generic        Intel Corporation
/0/6f                 generic        Intel Corporation
/0/70                  generic        Sky Lake-E M3KTI
↪ Registers
/0/71                  generic        Sky Lake-E M3KTI
↪ Registers
/0/72                  generic        Sky Lake-E M3KTI
↪ Registers
/0/73                  generic        Sky Lake-E M3KTI
↪ Registers
/0/74                  generic        Sky Lake-E M2PCI
↪ Registers
/0/75                  generic        Sky Lake-E M2PCI
↪ Registers
/0/76                  generic        Sky Lake-E M2PCI
↪ Registers
/0/77                  generic        Sky Lake-E M2PCI
↪ Registers
/0/4                   generic        Sky Lake-E CBDMA
↪ Registers
/0/4.1                 generic        Sky Lake-E CBDMA
↪ Registers
/0/4.2                 generic        Sky Lake-E CBDMA
↪ Registers
/0/4.3                 generic        Sky Lake-E CBDMA
↪ Registers
/0/4.4                 generic        Sky Lake-E CBDMA
↪ Registers
/0/4.5                 generic        Sky Lake-E CBDMA
↪ Registers
/0/4.6                 generic        Sky Lake-E CBDMA
↪ Registers
/0/4.7                 generic        Sky Lake-E CBDMA
↪ Registers
/0/78                  generic        Sky Lake-E
↪ MM/Vt-d Configuration Registers
/0/79                 generic        Intel Corporation
/0/7a                  generic        Sky Lake-E Ubox
↪ Registers
/0/7b                  generic        Sky Lake-E Ubox
↪ Registers
/0/7c                  generic        Sky Lake-E Ubox
↪ Registers
/0/104                bridge         Sky Lake-E PCI
↪ Express Root Port A
```

```
/0/104/0       ib1     network        MT27700 Family
↪  [ConnectX-4]
/0/7d                 generic        Intel Corporation
/0/7e                  generic        Sky Lake-E RAS
↪  Configuration Registers
/0/7f                  generic        Sky Lake-E CHA
↪  Registers
/0/8.1                 generic        Sky Lake-E CHA
↪  Registers
/0/8.2                 generic        Sky Lake-E CHA
↪  Registers
/0/80                  generic        Sky Lake-E CHA
↪  Registers
/0/81                  generic        Sky Lake-E CHA
↪  Registers
/0/82                  generic        Sky Lake-E CHA
↪  Registers
/0/83                  generic        Sky Lake-E CHA
↪  Registers
/0/84                  generic        Sky Lake-E CHA
↪  Registers
/0/85                  generic        Sky Lake-E CHA
↪  Registers
/0/86                  generic        Sky Lake-E CHA
↪  Registers
/0/87                  generic        Sky Lake-E CHA
↪  Registers
/0/88                  generic        Sky Lake-E CHA
↪  Registers
/0/89                  generic        Sky Lake-E CHA
↪  Registers
/0/8a                  generic        Sky Lake-E CHA
↪  Registers
/0/8b                  generic        Sky Lake-E CHA
↪  Registers
/0/8c                  generic        Sky Lake-E CHA
↪  Registers
/0/8d                  generic        Sky Lake-E CHA
↪  Registers
/0/8e                  generic        Sky Lake-E CHA
↪  Registers
/0/8f                  generic        Sky Lake-E CHA
↪  Registers
/0/90                  generic        Sky Lake-E CHA
↪  Registers
/0/91                  generic        Sky Lake-E CHA
↪  Registers
/0/92                  generic        Sky Lake-E CHA
↪  Registers
/0/93                  generic        Sky Lake-E CHA
↪  Registers
/0/94                  generic        Sky Lake-E CHA
↪  Registers
/0/95                  generic        Sky Lake-E CHA
↪  Registers
/0/96                  generic        Sky Lake-E CHA
↪  Registers
/0/97                  generic        Sky Lake-E CHA
↪  Registers
/0/98                  generic        Sky Lake-E CHA
↪  Registers
/0/99                  generic        Sky Lake-E CHA
↪  Registers
/0/9a                  generic        Sky Lake-E CHA
↪  Registers
/0/9b                  generic        Sky Lake-E CHA
↪  Registers
/0/9c                  generic        Sky Lake-E CHA
↪  Registers
/0/9d                  generic        Sky Lake-E CHA
↪  Registers
/0/9e                  generic        Sky Lake-E CHA
↪  Registers
/0/9f                  generic        Sky Lake-E CHA
↪  Registers
/0/a0                  generic        Sky Lake-E CHA
↪  Registers
/0/a1                  generic        Sky Lake-E CHA
↪  Registers
/0/a2                  generic        Sky Lake-E CHA
↪  Registers
/0/a3                  generic        Sky Lake-E CHA
↪  Registers
/0/a4                  generic        Sky Lake-E CHA
↪  Registers
/0/a5                  generic        Sky Lake-E CHA
↪  Registers
/0/a6                  generic        Sky Lake-E CHA
↪  Registers
/0/a7                  generic        Sky Lake-E CHA
↪  Registers
/0/a8                  generic        Sky Lake-E CHA
↪  Registers
/0/a9                  generic        Sky Lake-E CHA
↪  Registers
/0/aa                  generic        Sky Lake-E CHA
↪  Registers
/0/ab                  generic        Sky Lake-E CHA
↪  Registers
/0/ac                  generic        Sky Lake-E CHA
↪  Registers
/0/ad                  generic        Sky Lake-E CHA
↪  Registers
/0/ae                  generic        Sky Lake-E CHA
↪  Registers
/0/af                  generic        Sky Lake-E CHA
↪  Registers
```

```
/0/b0                 generic        Sky Lake-E CHA
↪ Registers
/0/b1                 generic        Sky Lake-E CHA
↪ Registers
/0/b2                 generic        Sky Lake-E CHA
↪ Registers
/0/b3                 generic        Sky Lake-E CHA
↪ Registers
/0/b4                 generic        Sky Lake-E CHA
↪ Registers
/0/b5                 generic        Sky Lake-E CHA
↪ Registers
/0/b6                 generic        Sky Lake-E CHA
↪ Registers
/0/b7                 generic        Sky Lake-E CHA
↪ Registers
/0/b8                 generic        Sky Lake-E CHA
↪ Registers
/0/b9                 generic        Sky Lake-E PCU
↪ Registers
/0/ba                 generic        Sky Lake-E PCU
↪ Registers
/0/bb                 generic        Sky Lake-E PCU
↪ Registers
/0/bc                 generic        Sky Lake-E PCU
↪ Registers
/0/bd                 generic        Sky Lake-E PCU
↪ Registers
/0/be                 generic        Sky Lake-E PCU
↪ Registers
/0/bf                 generic        Sky Lake-E PCU
↪ Registers
/0/105                bridge         Sky Lake-E PCI
↪ Express Root Port A
/0/105/0              bridge         PLX Technology,
↪ Inc.
/0/105/0/0            bridge         PLX Technology,
↪ Inc.
/0/105/0/0/0          display        GV100GL [Tesla
↪ V100 SXM2]
/0/105/0/4            bridge         PLX Technology,
↪ Inc.
/0/105/0/4/0          display        GV100GL [Tesla
↪ V100 SXM2]
/0/c0                 generic        Intel Corporation
/0/c1                 generic        Sky Lake-E RAS
↪ Configuration Registers
/0/8                  generic        Intel Corporation
/0/c2                 generic        Intel Corporation
/0/c3                 generic        Intel Corporation
/0/c4                 generic        Intel Corporation
/0/c5                 generic        Intel Corporation
/0/c6                 generic        Intel Corporation
/0/c7                 generic        Intel Corporation
/0/c8                 generic        Intel Corporation
/0/c9                 generic        Intel Corporation
/0/ca                 generic        Intel Corporation
/0/cb                 generic        Intel Corporation
/0/cc                 generic        Intel Corporation
/0/cd                 generic        Intel Corporation
/0/ce                 generic        Intel Corporation
/0/cf                 generic        Intel Corporation
/0/d0                 generic        Intel Corporation
/0/d1                 generic        Intel Corporation
/0/d2                 generic        Intel Corporation
/0/d3                 generic        Intel Corporation
/0/d4                 generic        Intel Corporation
/0/d5                 generic        Intel Corporation
/0/d6                 generic        Intel Corporation
/0/d7                 generic        Intel Corporation
/0/d8                 generic        Intel Corporation
/0/d9                 generic        Intel Corporation
/0/da                 generic        Intel Corporation
/0/106                bridge         Sky Lake-E PCI
↪ Express Root Port A
/0/106/0              storage        Express Flash
↪ NVMe P4500
/0/1                  bridge         Sky Lake-E PCI
↪ Express Root Port B
/0/5                  generic        Intel Corporation
/0/5.2                generic        Sky Lake-E RAS
↪ Configuration Registers
/0/db                 generic        Intel Corporation
/0/dc                 generic        Intel Corporation
/0/dd                 generic        Intel Corporation
/0/de                 generic        Intel Corporation
/0/df                 generic        Intel Corporation
/0/e0                 generic        Intel Corporation
/0/e1                 generic        Sky Lake-E M3KTI
↪ Registers
/0/e2                 generic        Sky Lake-E M3KTI
↪ Registers
/0/e3                 generic        Sky Lake-E M3KTI
↪ Registers
/0/e4                 generic        Sky Lake-E M3KTI
↪ Registers
/0/e5                 generic        Sky Lake-E M2PCI
↪ Registers
/0/e6                 generic        Sky Lake-E M2PCI
↪ Registers
/0/e7                 generic        Sky Lake-E M2PCI
↪ Registers
/0/e8                 generic        Sky Lake-E M2PCI
↪ Registers
/1             bond0  network        interface
```

ARTIFACT EVALUATION

Verification and validation studies: For output verification, (1) we used the image processing tool ImageJ (https://imagej.nih.gov/ij/)

to render the generated 3D volumes, then inspected them manually. (2) We also used profiled runs to investigate the density value of each voxel. (3) Finally, we compared the output with the volumes generated by the RTK library on the CPU, the per-projection Root Mean Square Error (RMSE) is less than 10e-5.

Accuracy and precision of timings: We use MPI_Time to measure the execution time in run-time. Regarding the same execution job on HPC, we run 100 times and report the mean value. We use the CPU clock to measure the execution time of different components running on the CPU, and GPU profiling tool (nvprof) to measure the GPU kernel execution time. Note that we submit the job one by one to avoid the I/O and communication contention.

Used manufactured solutions or spectral properties: N/A

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: The assigned compute node to us on the supercomputer varies from job to job (depending on the job scheduler). The condition of each compute node in the system can be different from run to run.

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. (1) For each job, we run a hundred of time, the execution time we reported is the averaged value. (2) we submit execution jobs to HPC one by one to avoid Parallel File System (PSF) contention, since we generate and write large data (e.g. the size of volume 8kx8kx8k is 2TB) in tens of seconds to Parallel File System (PSF) in each execution. The script is as follows:

```
################################################
#!/bin/sh
JOBS=$(ls jobs/run.*)
echo $JOBS
REPEAT_TIMES=1
if [ "$#" -gt 0 ]; then
   REPEAT_TIMES=$1
fi
echo "repeat count : Each job repeat times="$REPEAT_TIMES
function has_jobs(){
   s=$(qstat | grep $(whoami))
   if [ -z "$s" ] ; then
      echo "1"
   else
      echo "0"
   fi
}
function wait_jobs(){
   count=4
   sleep 1
   while [ $(has_jobs) = 0 ]; do
      printf "/rwait ...$count s"
      sleep 4
      count=$((count+4))
   done
   printf "/n"
}
for ((i=1; i<=$REPEAT_TIMES; i++)); do
   echo # $i
   for S in $JOBS; do
      qsub -g GROUPID $S
      wait_jobs
   done
done
################################################
```