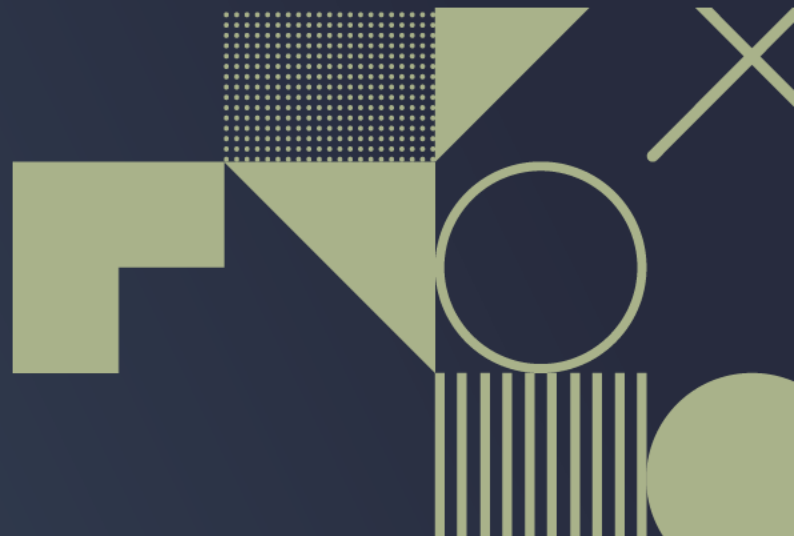
Scalable FBP Decomposition for Cone-Beam CT Reconstruction

Peng Chen,[1,2] Mohamed Wahib,[1,2] Xiao Wang,[6,7] Takahiro Hirofuchi,[1] Hirotaka Ogawa,[1]
Ander Biguri,[4] Richard Boardman,[5] Thomas Blumensath,[5] Satoshi Matsuoka [2,3]

1. National Institute of Advanced Industrial Science and Technology, Japan
2. RIKEN Center for Computational Science, Japan
3. Tokyo Institute of Technology, Japan
4. Institute of Nuclear Medicine, University College London, UK
5. μ-VIS X-Ray Imaging Centre, University of Southampton, UK
6. Oak Ridge National Laboratory, US
7. Boston Children's Hospital, US



Outline

1. Introduction
2. Problem Statement
3. Novelty of our work
4. Proposed projection & volume decomposition methodology
5. Implementation of the proposed framework on a distributed system
6. Evaluation
7. Conclusion
8. Future work

Introduction

- Computed Tomography (CT) is a widely used 3D imaging technology
  - Medical diagnosis
  - Non-invasive inspection
  - Reverse engineering
- Possibility of obtaining a high-resolution image
  - Rapid development in CT manufacturing
  - CMOS-based Flat Panel Detector (FPD, X-ray imaging sensor) become larger
    - 2048×2048, 4096×4096, etc.
  - Micro focus x-ray become better and cheaper
- Complex computation for 3D image reconstruction
  - Filtering computation (or convolution)
  - Back-projection
- The commonly used resolution : 256^3, 512^3, 1024^3

Problem Statement

- Problems in High-resolution CT image reconstruction
  1. Intensive computation
  2. Critical timing demanding for image reconstruction
  3. Huge memory capacity
     - 1024^3 : 4GB
     - 2048^3 : 32GB
     - 4096^3 : 256GB
     - …… : ……
- Challenges of using GPU-accelerated supercomputers to solve this problem
  1. GPU is powerful in computation, but **memory capacity** is limited
  2. How to optimize algorithms on **GPU**?
  3. How to use the **heterogeneous architecture** (CPUs, GPUs) ?
  4. How to optimally perform inter-process communication by **MPI** ?
  5. How to achieve **high performance** and **scaling**?

Novelty of Our Work

- The parallel-beam (Fig. 1a) based algorithms decompose input problems in 2D/3D dimensions
- This is the first work to decompose the input problem in 2D dimension using cone-beam (Fig. 1c)
- Our system can perform out-of-core image reconstruction

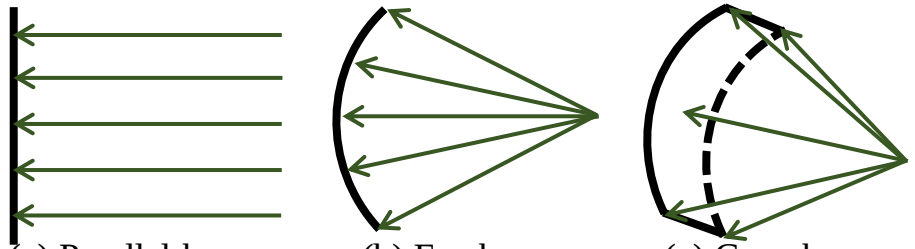


Fig. 1: Different geometries for X-ray sources and detectors. Cone-beam is the geometry used in the latest (7^{th} generation) of CT.

Implementation	Algorithm	Beam Shape	Decomposition		Lower-bound Input Size	Out-of-Core Capability	Multiple		Communication (MPI)
			Input	Output			GPUs	Nodes	
Trace [5]	IR	Parallel	2D	1D	$O(N_p)$	✓	✗	✓	$O(log(N))$
NU-PSV [63]	IR	Parallel	2D	1D	$O(N_p)$	✓	✗	✓	$O(log(N))$
MemXCT [27]	IR	Parallel	2D	2D	$O(N_p)$	✗	✓	✓	$O(N)$
Peta-scale XCT [28]	IR	Parallel	3D	3D	$O(N_p)$	✗	✓	✓	$O(N)$
Consensus Equilibrium [64]	IR	Parallel	2D	1D	$O(N_p)$	✓	✗	✓	$O(Nlog(N))$
DMLEM [12]	IR	Cone	1D	✗	$O(N_u \times N_v)$	✗	✓	✓	$O(Nlog(N))$
Palenstijn et al. [44]	IR	Cone	1D	1D	$O(N_u \times N_p)$	✗	✓	✓	$O(Nlog(N))$
TIGRE [6]	IR	Cone	1D	1D	$O(N_u \times N_v)$	✗	✓	✗	✗
Lu et al. [38]	FBP	Cone	1D	1D	$O(N_u \times N_v)$	✓	✗	✗	✗
iFDK [9]	FBP	Cone	1D	1D	$O(N_u \times N_v)$	✗	✓	✓	$O(Nlog(N))$
This work	**FBP**	**Cone**	**2D**	**1D**	$O(N_u)$	✓	✓	✓	$O(log(N))$

Table 1: State-of-the-art image reconstruction solutions by FBP and Iterative Reconstruction (IR) algorithms.

Introduction of Compute Tomography

- Cone Beam Compute Tomography (CBCT): Geometry & Parameter
- FBP algorithm for CBCT was Presented by Feldkamp, Davis, and Kress (FDK) in 1984 (37 years ago)
  - FBP method is indispensable in most of the practical CT systems
- Intensive computation for 3D image reconstruction
  - Filtering computation: $O(Log(N)N^2)$
  - Back-projection computation: $O(N^4)$

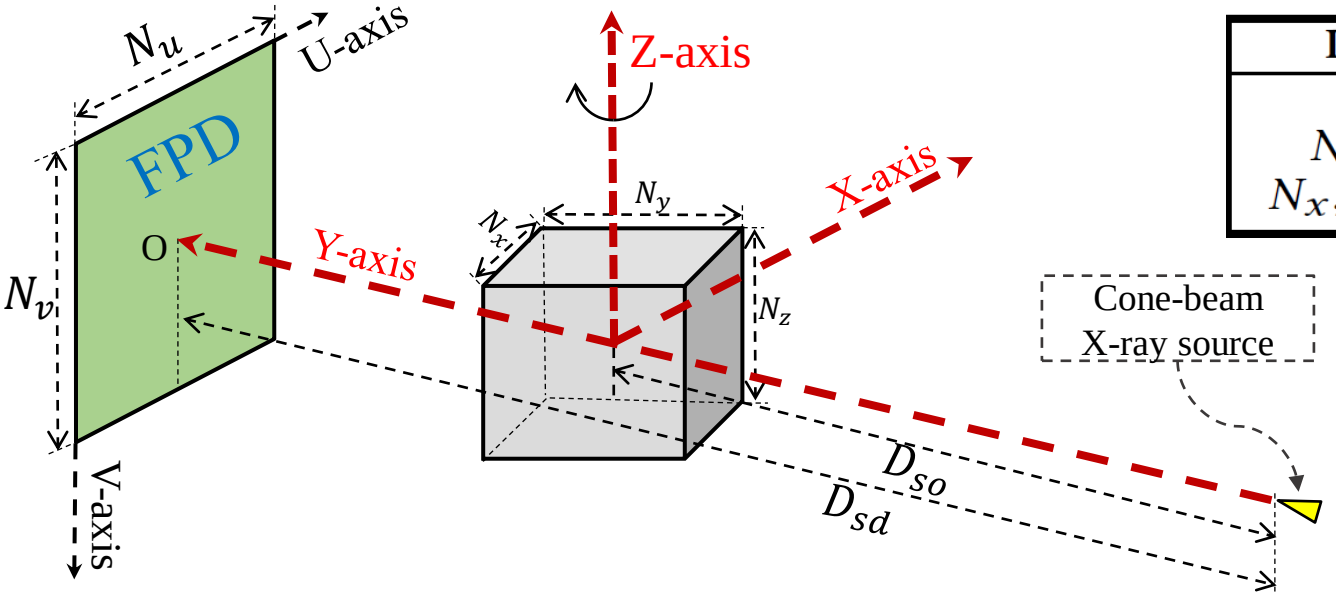


Fig. 1: Cone-beam CT (CBCT) with a Flat Panel Detector (FPD).

Param	Description
N_p	the number of 2D projections
N_u, N_v	the width and height of a 2D projection, respectively
N_x, N_y, N_z	the number of voxels in X, Y, Z dimension, respectively

- Reconstruction Problem Definition :

$$N_u \times N_v \times N_p \rightarrow N_x \times N_y \times N_z$$

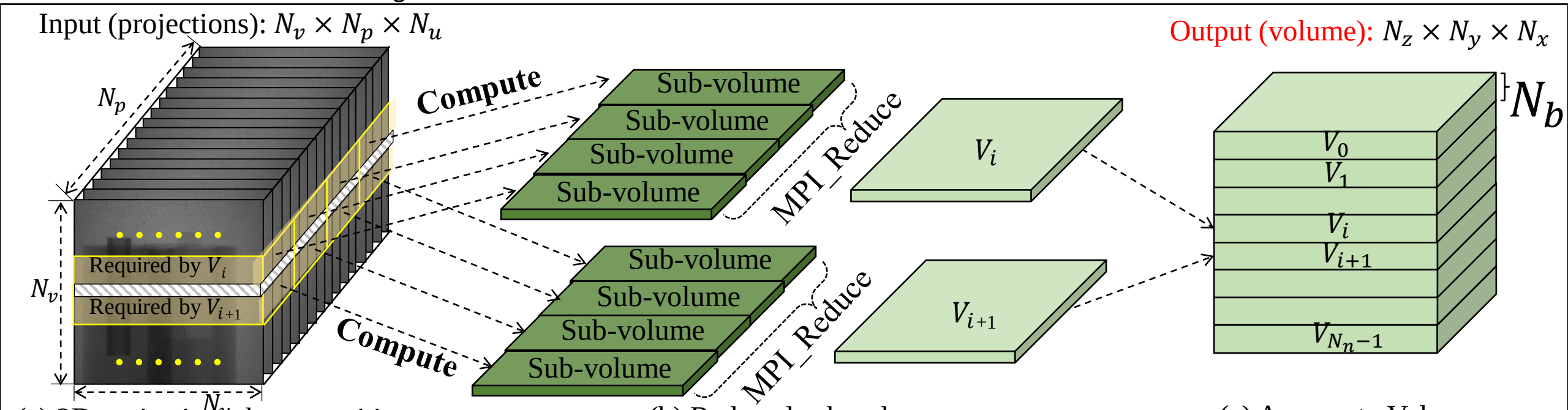
- Performance metrics :

$$GUPS = N_x * N_y * N_z * N_p / T$$

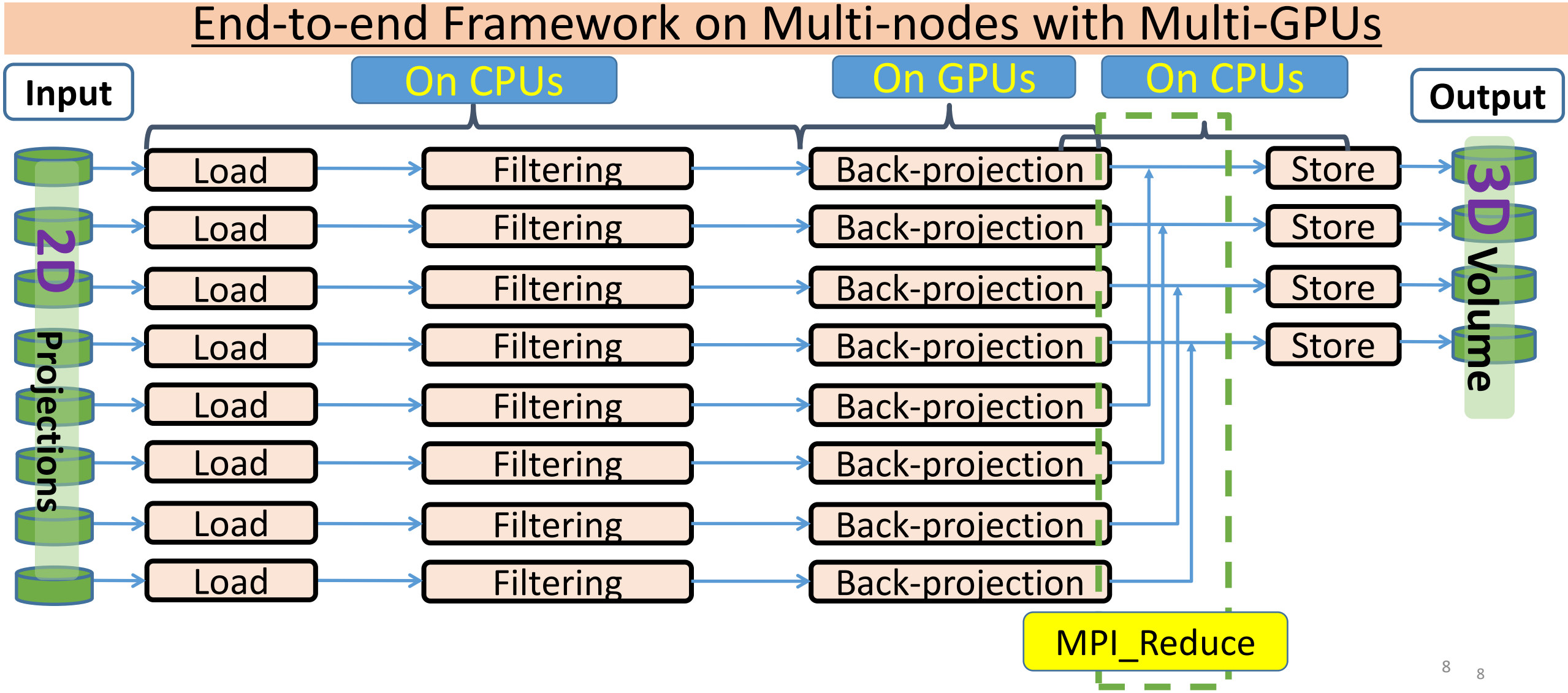
where T is execution time in a unit of second.

Projection and Volume Decomposition Methodology

- Volume data is decomposed in 1D, each sub-volume has slices of N_b (as Fig. c)
- Projection is decomposed in 2D (as Fig. a)
  - N_p dimension is spited averagely
  - N_v dimension is spited with overlapped area
- Benefits of the proposed algorithm
  - Streaming/pipeline processing available
  - Out-of-core image reconstruction available



Overview of the Proposed FBP Framework



A General Projection Matrix

- Correcting the geometric offset is required to reconstruct tomographic images
  - Good spatial resolution
  - Low artifact content
- The proposed matrix (M_ϕ) is general and can be reused for most CBCT systems
  - Offset of FPD at U- and V-axis (Fig. 1)
  - Microscope CT system with rotation center offset (Fig. 2).

Description	Parameter	Unit
Rotation angle	ϕ	degree
Distance from source to object (Z-axis)	D_{so}	mm
Distance from source to detector (FPD)	D_{sd}	mm
The number of 2D projections	N_p	–
The width and height of a 2D projection	N_u, N_v	pixel
Pixel pitch at U- and V-axis	$\triangle_u, \triangle_v$	mm/pixel
The number of voxels in X-, Y-, Z-axis	N_x, N_y, N_z	voxel
Voxel pitch at X-, Y-, and Z-axis	$\triangle_x, \triangle_y, \triangle_z$	mm/voxel
Offset of FPD at U- and V-axis ⟸ Fig. 1	σ_u, σ_v	pixel
Rotation center offset ⟸ Fig. 2	σ_{cor}	mm

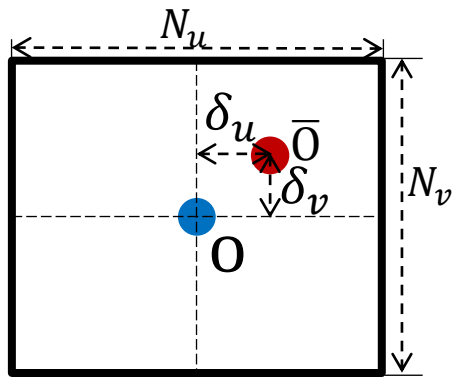


Fig. 1: FPD center offset.

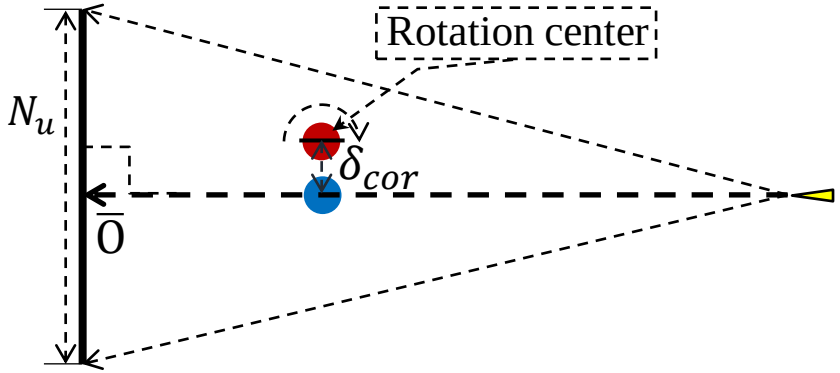


Fig. 2: Rotation center offset (top-view).

$$M_\phi = \begin{bmatrix} \frac{D_{sd}}{\triangle_u} & 0 & \frac{N_u-1}{2} + \sigma_u & 0 \\ 0 & \frac{D_{sd}}{\triangle_v} & \frac{N_v-1}{2} + \sigma_v & 0 \\ 0 & 0 & 0 & D_{sd} \end{bmatrix} \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 & \sigma_{cor} \\ 0 & 0 & -1 & 0 \\ sin(\phi) & cos(\phi) & 0 & D_{so} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \triangle_x & 0 & 0 & \frac{1-\triangle_x \cdot N_x}{2} \\ 0 & -\triangle_y & 0 & \frac{\triangle_y \cdot N_y}{2} \\ 0 & 0 & \triangle_z & \frac{\triangle_z \cdot N_z}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Operation for a Voxel

- A projection matrix ($M_{\emptyset}$) is of size 3×4
- Three inner product operations are required for each projection operation
- (i, j, k) is the index of a voxel, where $\mathrm{i} \in [0, N_x - 1], \mathrm{k} \in [0, N_y - 1], \mathrm{k} \in [0, N_z - 1]$

$$[x, y] = Projection(M_{\phi}, [i, j, k]) \quad (1)$$

$$\begin{cases} z \leftarrow \langle M_{\phi}[2], [i, j, k, 1] \rangle \\ x \leftarrow \langle M_{\phi}[0], [i, j, k, 1] \rangle / z \\ y \leftarrow \langle M_{\phi}[1], [i, j, k, 1] \rangle / z \end{cases}$$

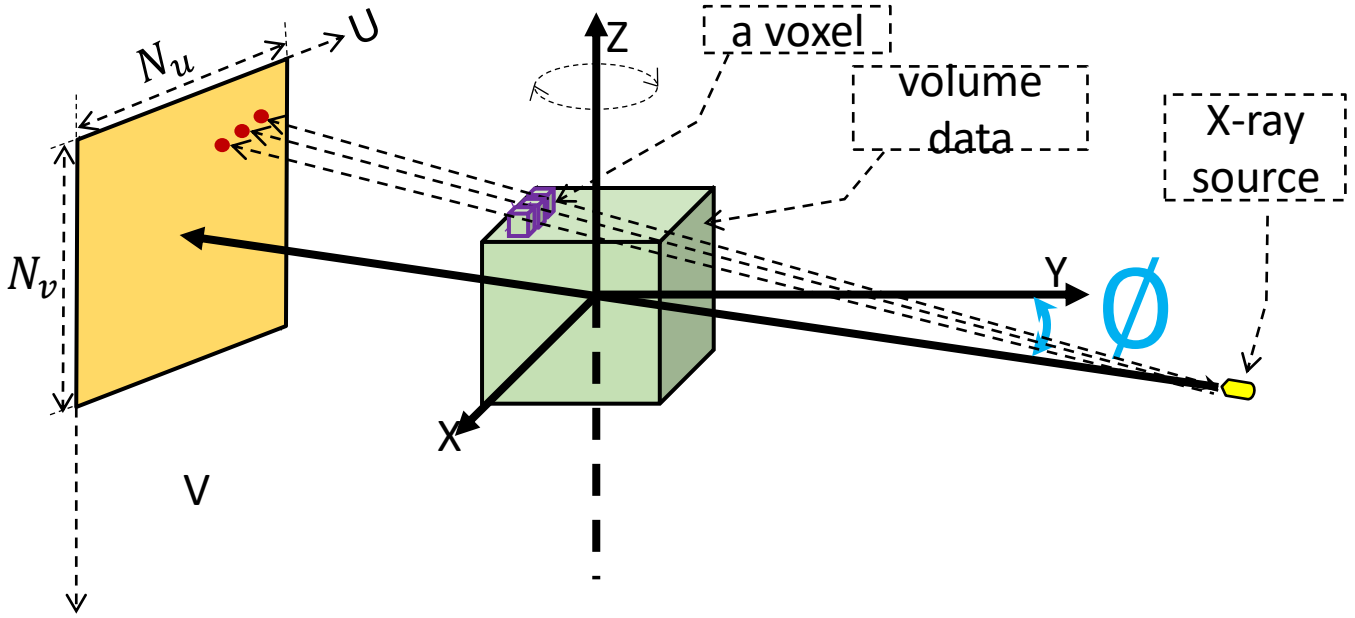


Fig. 1: three examples of projection operations.

Projections Decomposition in N_v Dimension

- The volume data is decomposed in 1D
- The projections are decomposed in 2D (N_p and N_v dimensions)
  - N_p dimension is spited averagely
  - N_v dimension is spited (as in Fig. 1)
- Four projection operations for computing $\overline{a_i b_i}$ (as in Eqn 1)

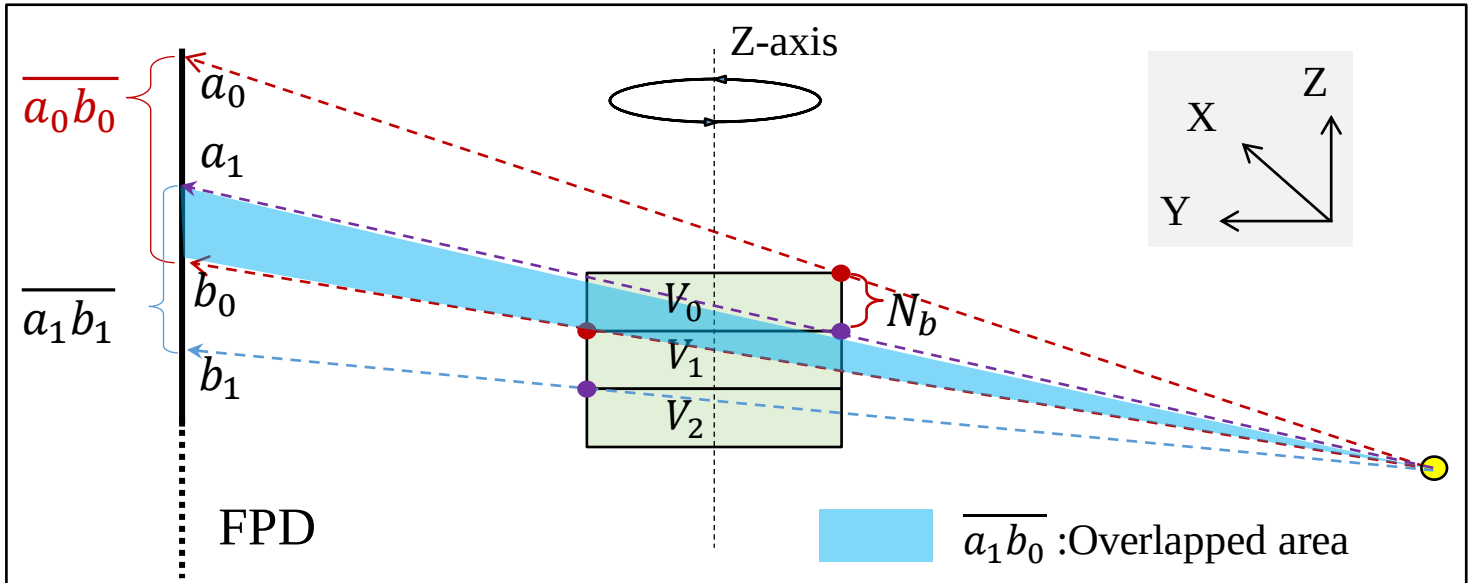


Fig. 1: Schematic YZ-view of overlapped projections.

Function *ComputeAB(begin_idx, end_idx)*

- [-, y_0] = $Projection(M_{135°}, [0, 0, begin\_idx])$
- [-, y_1] = $Projection(M_{315°}, [0, 0, begin\_idx])$
- [-, y_2] = $Projection(M_{135°}, [0, 0, end\_idx - 1])$
- [-, y_3] = $Projection(M_{315°}, [0, 0, end\_idx - 1])$
- a = (int)floor(**min4**(y_0, y_1, y_3, y_4)) ▹ floor operation
- b = (int)ceil(**max4**(y_0, y_1, y_3, y_4)) ▹ ceil operation
- **return** $\overline{ab}$ ▹ integer values

$$\overline{a_i b_i} = ComputeAB(i \cdot N_b, (i+1) \cdot N_b) \quad (1)$$

A Novel Out-of-core Back-projection Kernel

```
texture<float,cudaTextureType3D> tex;//3D texture memory
__global void kernelBackProjection(float* volume, int
    offset_volume_z, int3 vol_dim, const float4* proj_mat,
     int3 proj_dim, int offset_proj_y) {
    int i = blockIdx.x*blockDim.x+threadIdx.x;
    int j = blockIdx.y*blockDim.y+threadIdx.y;
    int k = blockIdx.z*blockDim.z+threadIdx.z;
    if (i>=vol_dim.x || j>=vol_dim.y || k>=vol_dim.z)
        return;
    float sum = 0;
    int K = k+offset_volume_z //offset k
    float4 ijk = make_float4(i,j, K, 1.0f);
    for (int s = 0; s < proj_dim.y; s++, proj_mat += 3) {
        float z = dot(__ldg(&proj_mat[2]), ijk);
        float x = dot(__ldg(&proj_mat[0]), ijk)/z;
        float y = dot(__ldg(&proj_mat[1]), ijk)/z;
        float Y = y-offset_proj_y;//offset y
        sum += 1.f/(z*z)*devSubPixel(x, Y, s, proj_dim);
    }
    //update a voxel of the volume data
    volume[vol_dim.x*vol_dim.y*k+vol_dim.x*j+i] = sum;
}
```

Back-projection operation

- Partial projections are cached by 3D texture memory
- Volume data are stored via global memory
- Back-projection is conducted for each voxel
- The projection matrices are accessed via cache-optimized intrinsic __ldg
- We compute a sub-volume by launching the CUDA kernel
- The projections are moved from host to device only once

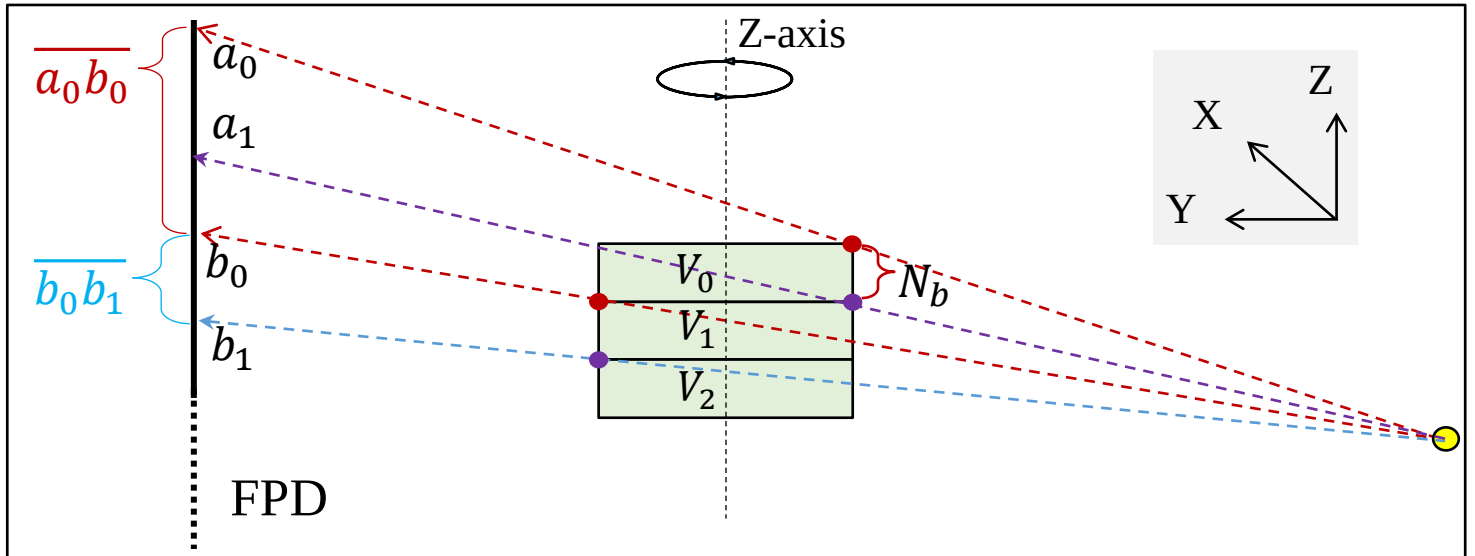


Fig. 1: Schematic YZ-view of overlapped projections.

Orchestration and Pipelining in our Framework

- Each MPI rank launches four extra-threads by std::thread library
- Filtering thread launches multiple OpenMP threads for filtering computation

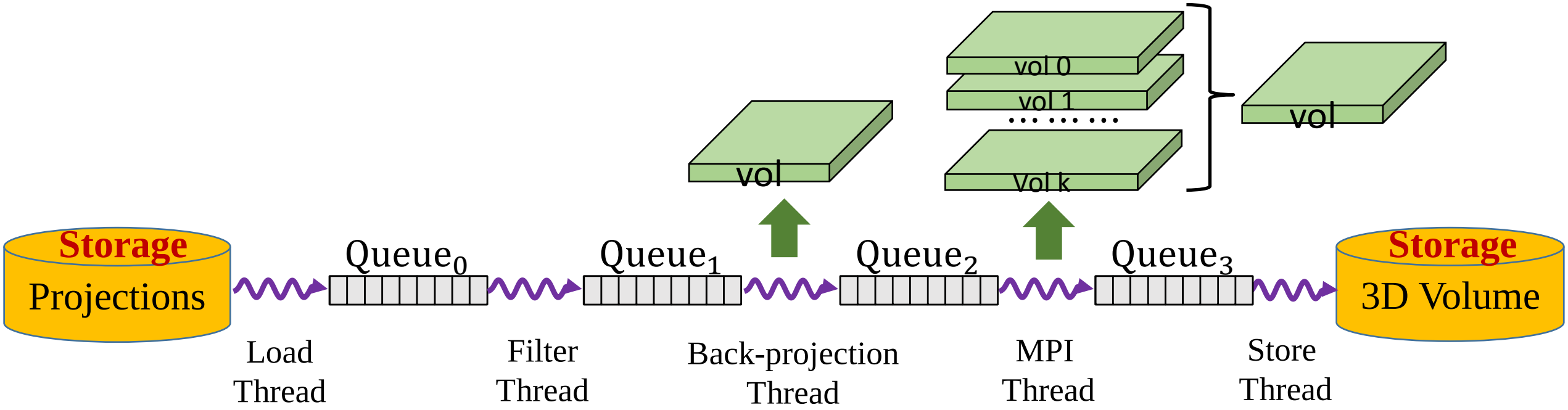


Fig. 1: A end-to-end view of the pipeline in a single MPI rank. Queues correspond to the stages of the pipeline.

Evaluation Environment

- ABCI supercomputer
  - Constructed and operated by AIST
  - 1,088 computing nodes, 4,352 Tesla V100 GPUs
- Software
  - CentOS 7.4
  - CUDA 10.2
  - Intel library 2020.4.304 (MPI, IPP)
  - RTK (Reconstruction Toolkit) 1.4.0 (**https://www.openrtk.org/**)

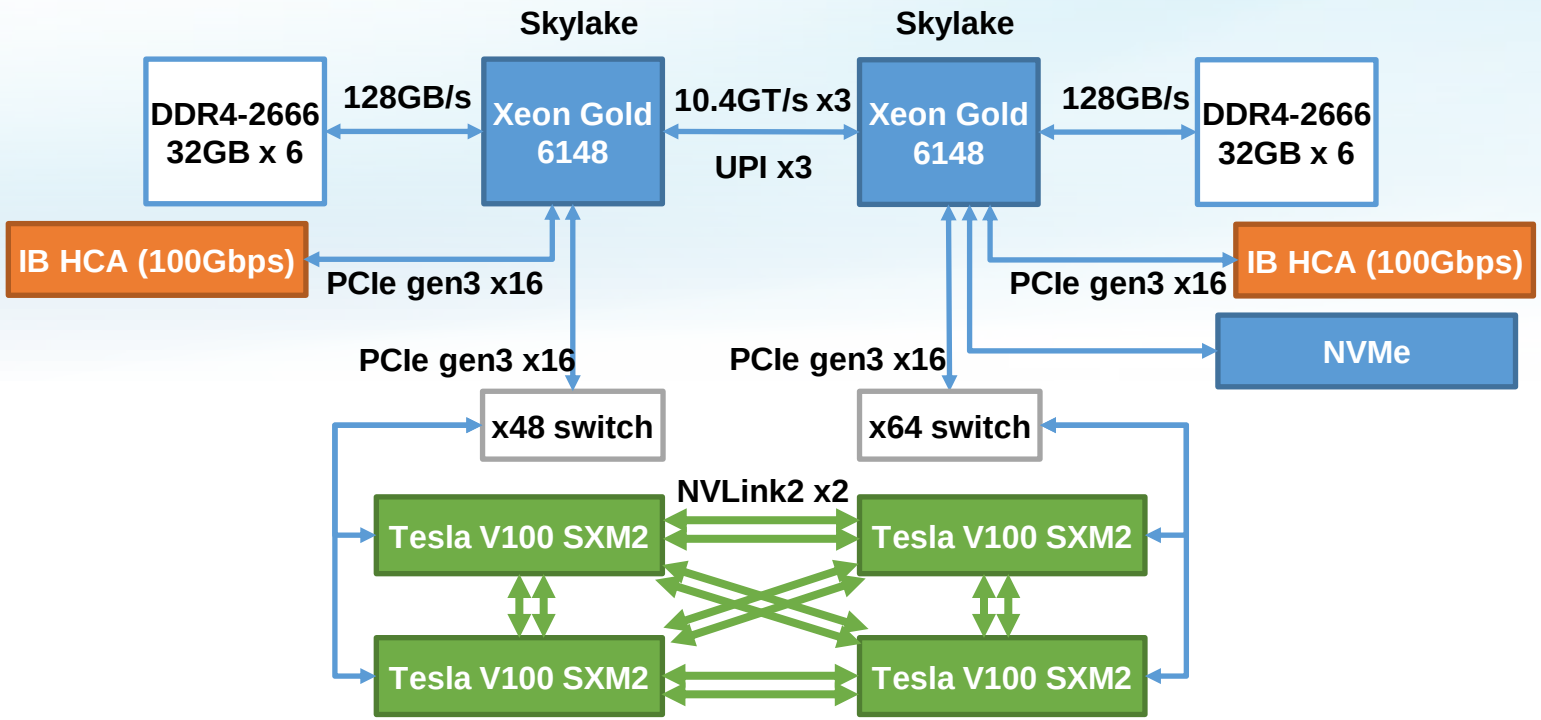
- Evaluation dataset
  - Coffee bean
  - Bumblebee
  - Four TomoBank datasets (**https://tomobank.readthedocs.io/en/latest/#**)

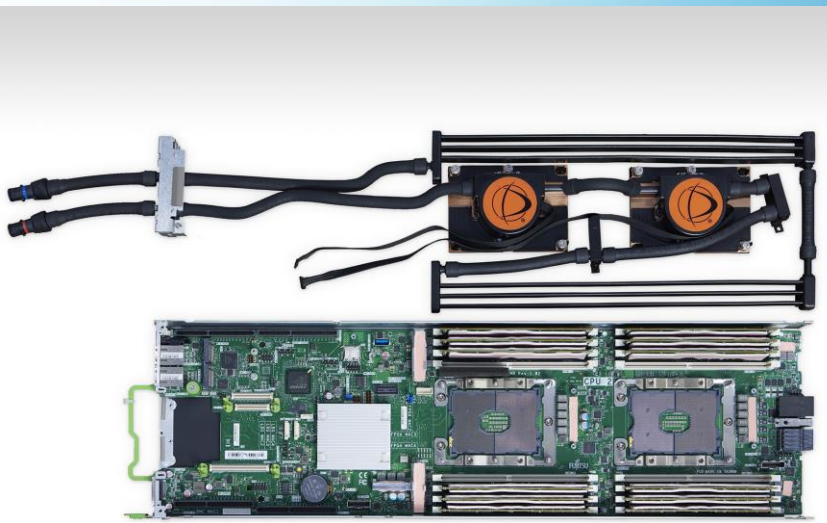
	Coffee bean	Bumblebee	tomo_ID			
			00027	00028	00029	00030
Size	117.8GB	46.8GB	17.9GB	17.9GB	17.9GB	816MB
N_u	3728	2000	2004	2004	2004	668
N_v	2000	2000	1335	1335	1335	445
N_p	6401	3142	1800	1800	1800	720
σ_u	0	0	25	26	27	-10
σ_v	0	0	0.25	0.25	0.2	0.2
σ_{cor}	-0.0021	1.03	0	0	0	0

Table 1: Datasets with geometric offset.

ABCI Compute Node

FUJITSU PRIMERGY Server (2 servers in 2U)	
CPU	Xeon Gold 6148 (27.5M Cache, 2.40 GHz, 20 Core) x2
GPU	NVIDIA Tesla V100 (SXM2) x4
Memory	384GiB DDR4 2666MHz RDIMM
Local Storage	1.6TB NVMe SSD (Intel SSD DC P4600 u.2) x1
Interconnect	InfiniBand EDR x2





Out-of-core Image Reconstruction on a GPU

	Input tomo_ID	Output ($voxel^3$)	$T_{runtime}$ (s)
V100 GPU (16GB)	00030 (816MB)	$\mathbf{512^3}$(512MB)	1.4
		$\mathbf{1024^3}$(4GB)	7.9
		$\mathbf{2048^3}$(32GB)	60.2
		$\mathbf{4096^3}$(256GB)	475.0
	00029 (17.9GB)	$\mathbf{512^3}$(512MB)	19.7
		$\mathbf{1024^3}$(4GB)	25.4
		$\mathbf{2048^3}$(32GB)	137.7
		$\mathbf{4096^3}$(256GB)	1028.8
A100 GPU (40GB)	00030 (816MB)	$\mathbf{512^3}$(512MB)	1.1
		$\mathbf{1024^3}$(4GB)	6.853
		$\mathbf{2048^3}$(32GB)	52.4
		$\mathbf{4096^3}$(256GB)	347.1
	00029 (17.9GB)	$\mathbf{512^3}$(512MB)	10.1
		$\mathbf{1024^3}$(4GB)	19.7
		$\mathbf{2048^3}$(32GB)	114.9
		$\mathbf{4096^3}$(256GB)	807.2

- A single Tesla V100/A100 GPUs
- Use tomo_00029 and tomo_00030 datasets
- We can generate volume of 2048^3 and 4096^3 beyond the memory capacity of a single GPU

Image Reconstruction Performance on a GPU

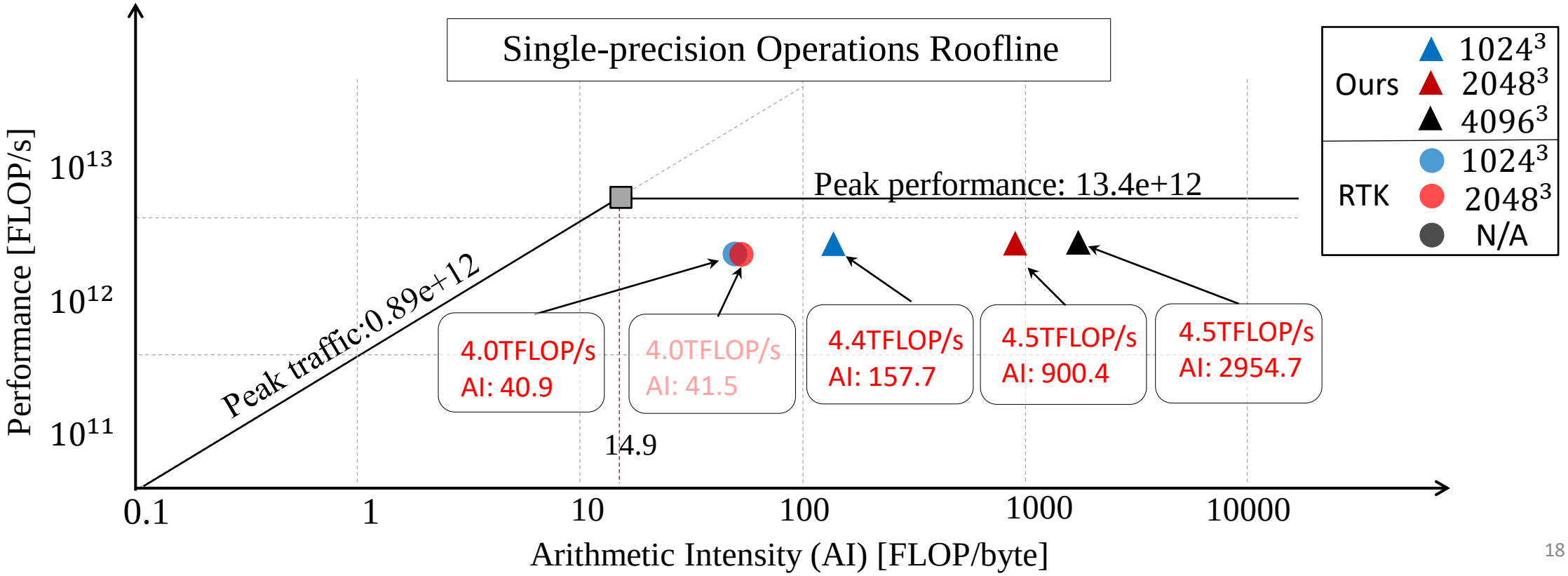
	Input tomo_ID	Output ($voxel^3$)	Perf. (GUPS) Ours	Perf. (GUPS) RTK
V100 GPU (16GB)	00030 (816MB)	512^3(512MB)	111.6	110.8
		1024^3(4GB)	115.7	113.7
		2048^3(32GB)	117.2	✗
		4096^3(256GB)	120.1	✗
	00029 (17.9GB)	512^3(512MB)	29.5	104.7
		1024^3(4GB)	107.0	107.7
		2048^3(32GB)	125.1	✗
		4096^3(256GB)	129.2	✗
A100 GPU (40GB)	00030 (816MB)	512^3(512MB)	111.6	125.4
		1024^3(4GB)	152.0	127.4
		2048^3(32GB)	155.3	✗
		4096^3(256GB)	159.7	✗
	00029 (17.9GB)	512^3(512MB)	87.8	122.0
		1024^3(4GB)	137.5	124.3
		2048^3(32GB)	158.2	✗
		4096^3(256GB)	166.4	✗

- A single Tesla V100/A100 GPUs
- Use tomo_00029 and tomo_00030 datasets
- We achieve competitive performance comparing to the state-of-the-art RTK library (baseline)
- RTK is uncapable of out-of-core computations

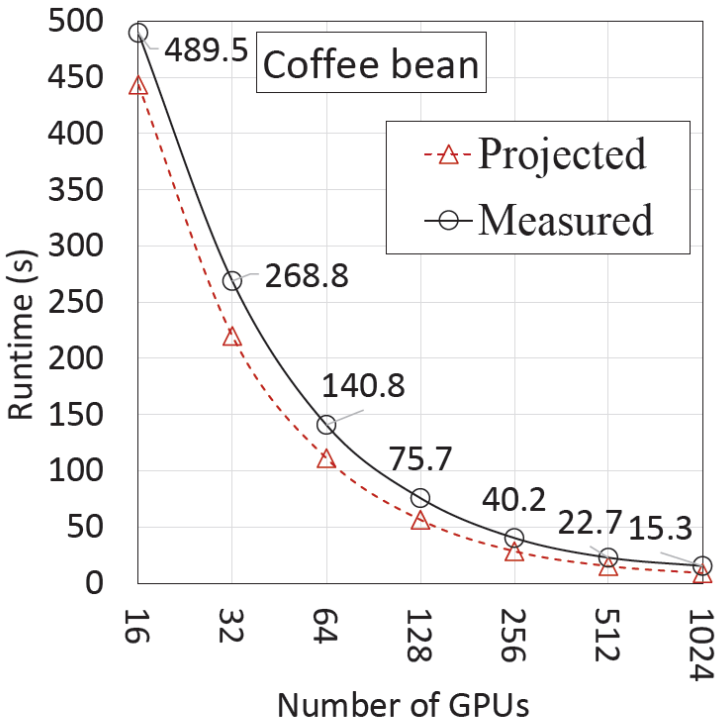
Reconstruction Toolkit (RTK): https://www.openrtk.org

Back-projection Roofline Analysis on a V100 GPU

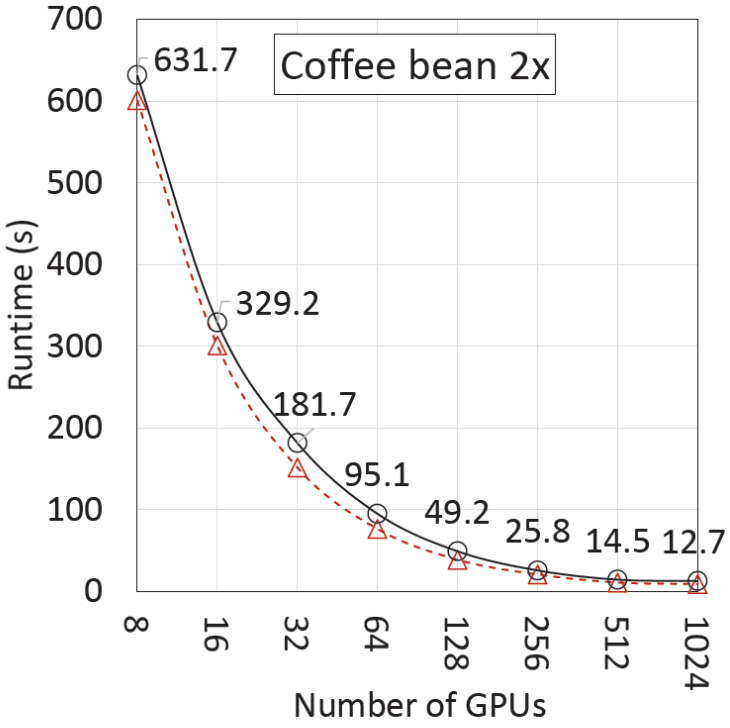
- The roofline result is generated by NVIDIA Nsight Compute profiler
- We use tomo_00030 dataset
- We achieve out-of-core computing capability without sacrificing the back-projection performance



Strong Scaling Evaluation (1/2)



(a) $3928 \times 1998 \times 6401 \Rightarrow 4096^3$.



(b) $1864 \times 999 \times 6401 \Rightarrow 4096^3$.

- Evaluation on Coffee bean datasets
- The Projected performance is predicted by our performance model
- The Measure performance is our runtime
- We achieve outstanding strong scalability, scales up to 1024 GPUs
- Our performance is bounded by the storage IO
- We achieved 78% of the peak performance in average

Fig. 1: Strong scaling. Coffee bean 2x is a rebinning of the original dataset (i.e., double the pixel size to reduce the input size to 1/4)

Strong Scaling Evaluation (2/2)

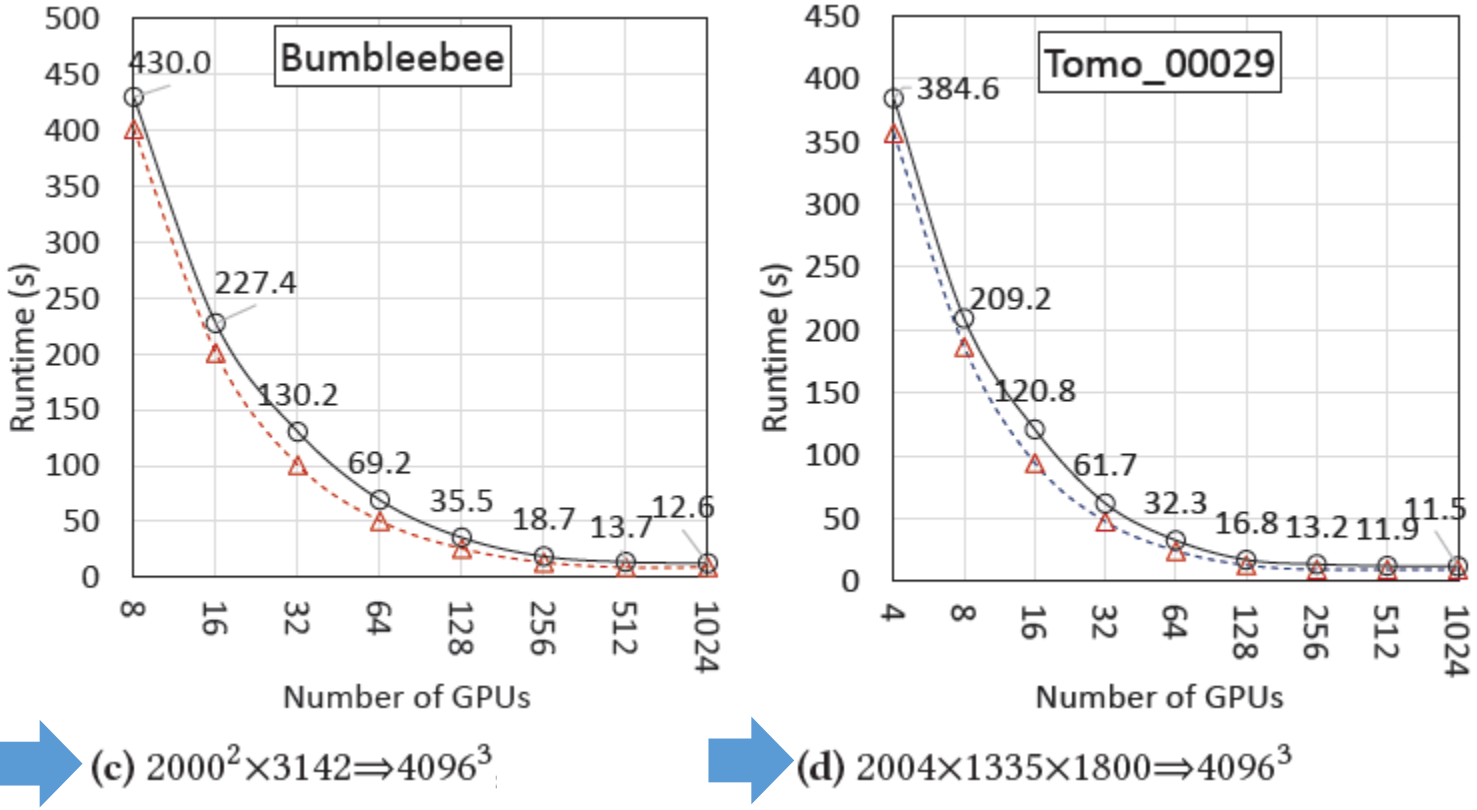
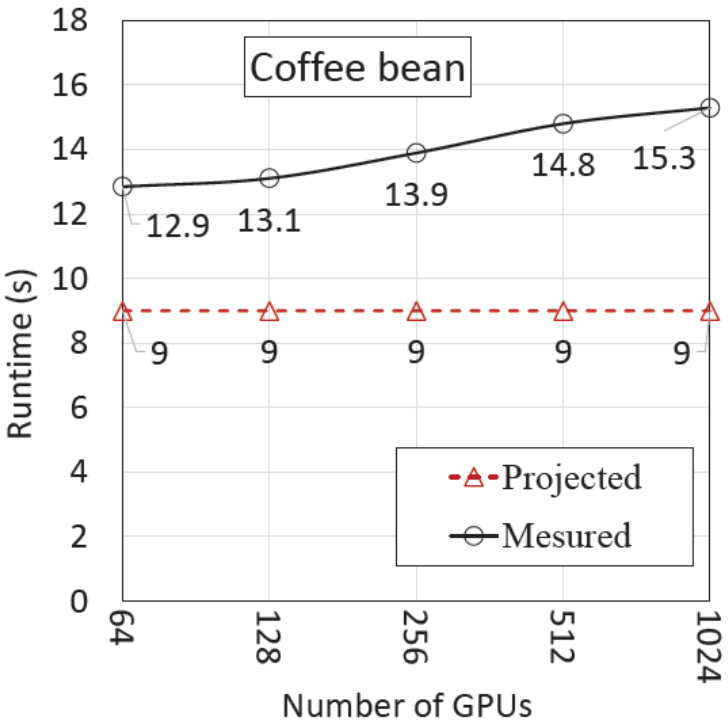
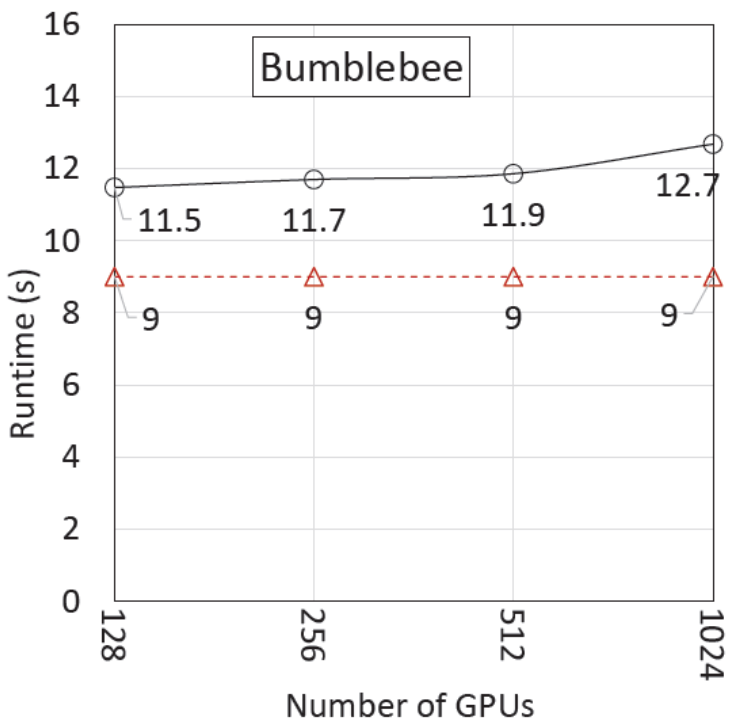


Fig. 2: Strong scaling.

- Evaluation on Bumbleebee and Tomo_00029 datasets
- The Projected performance is predicted by our performance model
- The Measure performance is our runtime
- We achieve outstanding strong scalability, scales up to 1024 GPUs
- Our performance is bounded by the storage IO
- We achieved 78% of the peak performance in average

Weak scaling





(a) $3928 \times 1998 \times \frac{6401 \cdot Ngpus}{1024} \Rightarrow 4096^3$.

(b) $2000 \times 2000 \times \frac{3142 \cdot Ngpus}{1024} \Rightarrow 4096^3$.

Fig. 1: Weak scaling.

- The Projected performance is predicted by our performance model
- The Measure performance is
- We achieve outstanding weak scalability
- Our performance is bounded by the storage IO
- We achieved 78% of the peak performance in average

Computational Performance

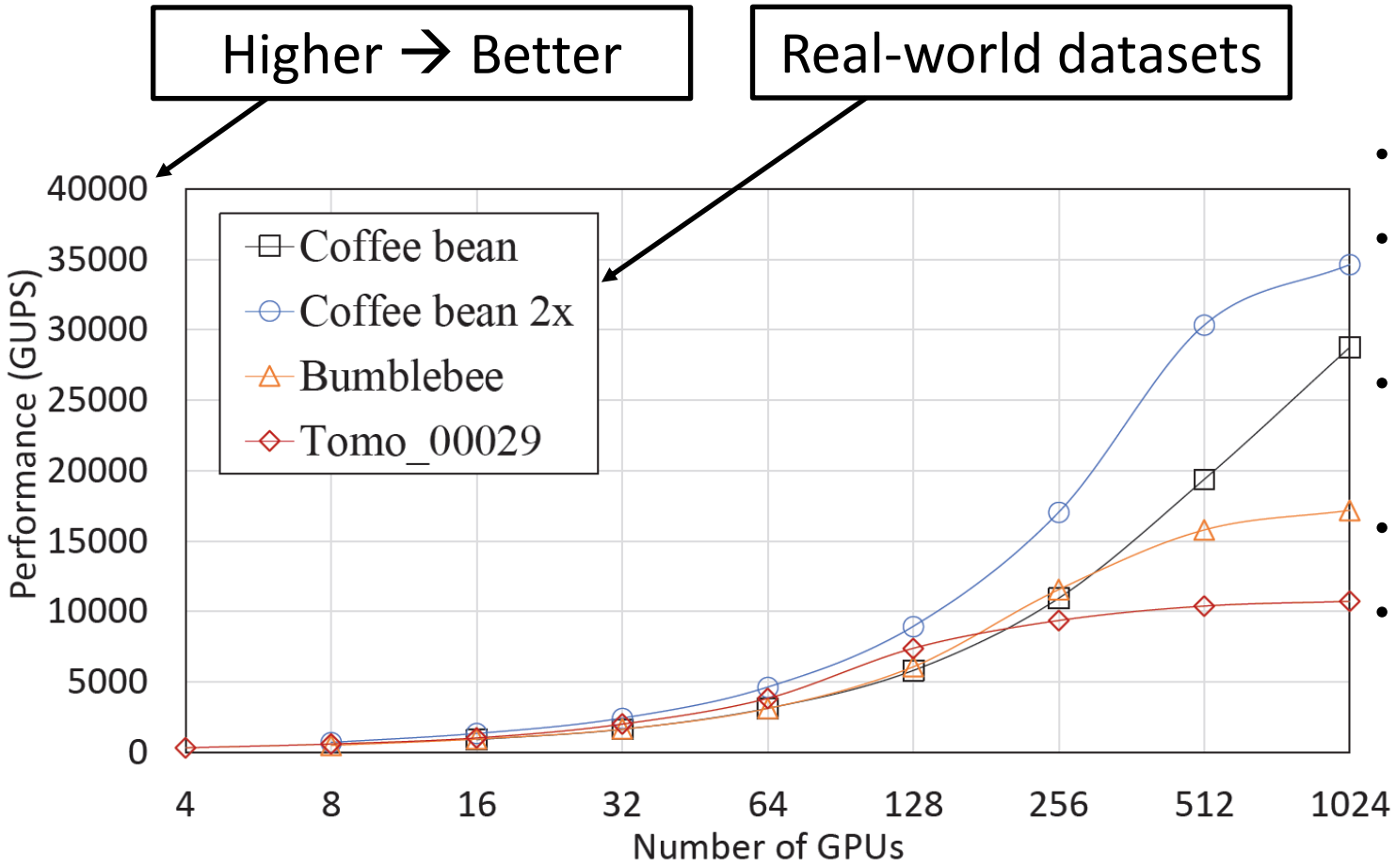


Fig. 1: Performance (GUPS) when generating 4096^3 volumes

- Extremely high performance (GUPS)
- Over **two order of magnitude** faster than a single Tesla V100 GPU (Perf. < 100 GUPS)
- Higher computational intensity, better scalability
- Bottleneck becomes the IO on storage
- **Using large scale system, we can solve any FBP problems immediately (~10s)**

Examples of Achieved Overlapping

- We show two evaluated examples
  - Tomo_00029: 2004×1335×1800 (17.9GB)
  - Bumblebee: 2000×2000×3142 (46.8GB)
- H2D means moving data from host to device
- D2H means moving data from device to host

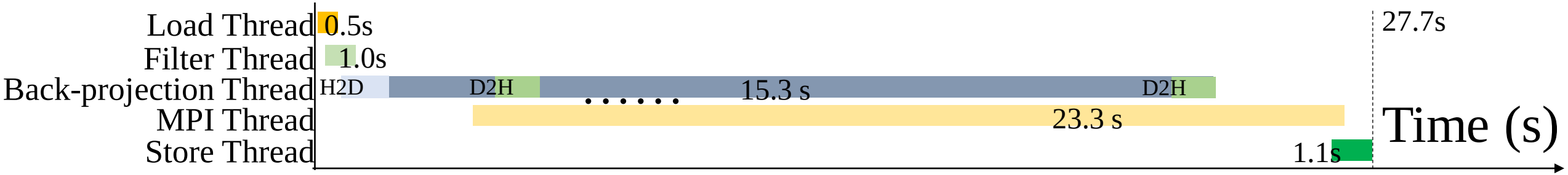


Fig. 1: Reconstructing 2048^3 volume of Tomo_00029. N_{gpus}=1.

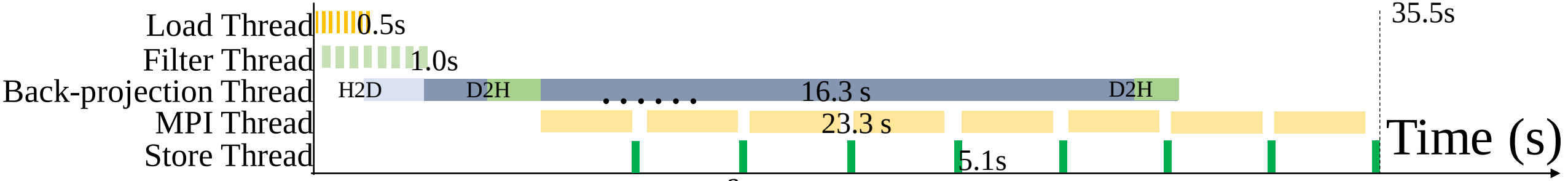


Fig. 2: Reconstructing 4096^3 volume of Bumblebee. N_{gpus}=128.

An Example of MPI_Reduce Operation

- MPI_Reduce is a highly-optimized primitive for inter-processes communication
- MPI_Reduce can take advantage of the high-bandwidth connectors to perform reduce operations in supercomputers

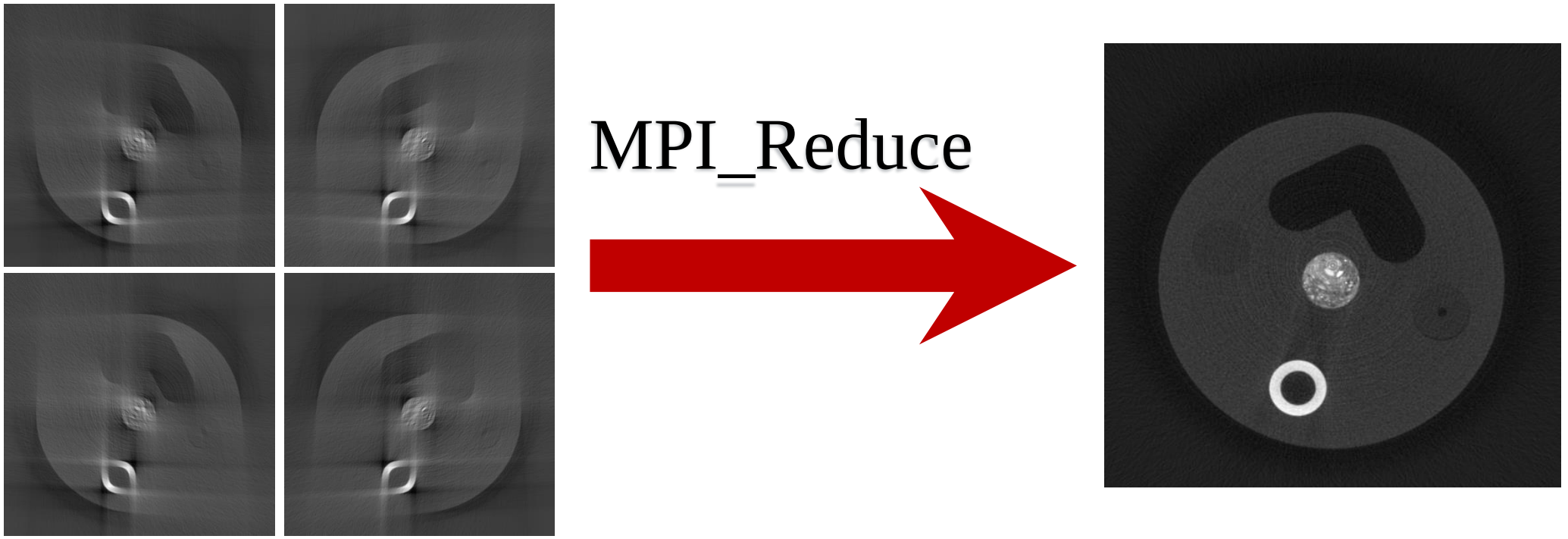
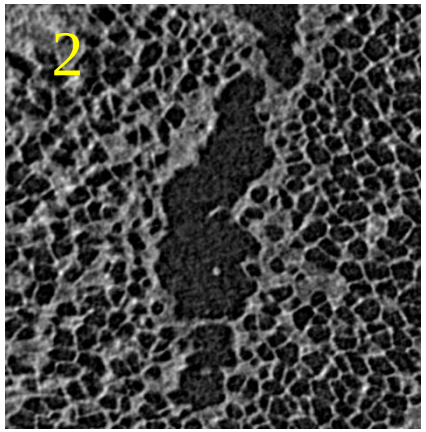
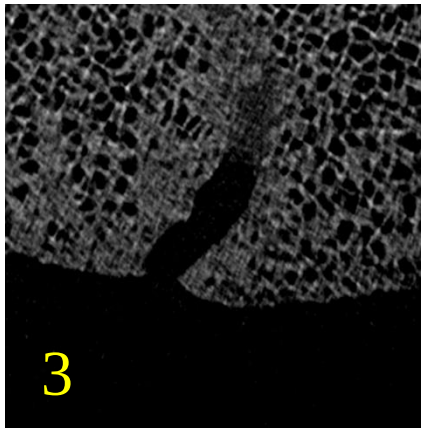
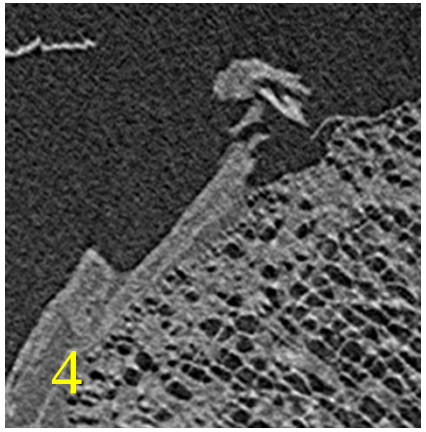


Fig. 1: MPI_Reduce on a slice (512 × 512) of tomo_00030 dataset.

An example of High-resolution Volume data (coffee bean)



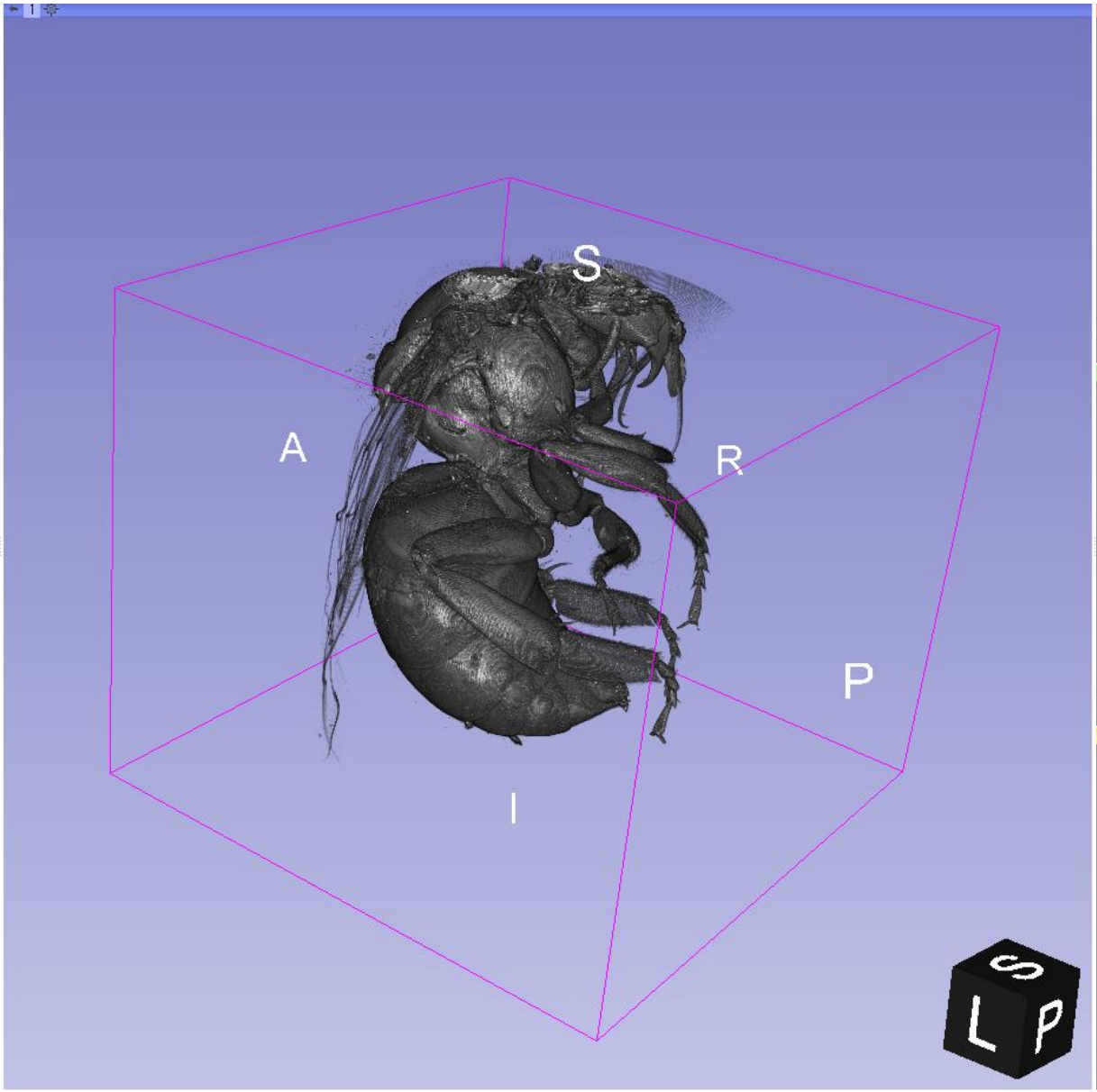


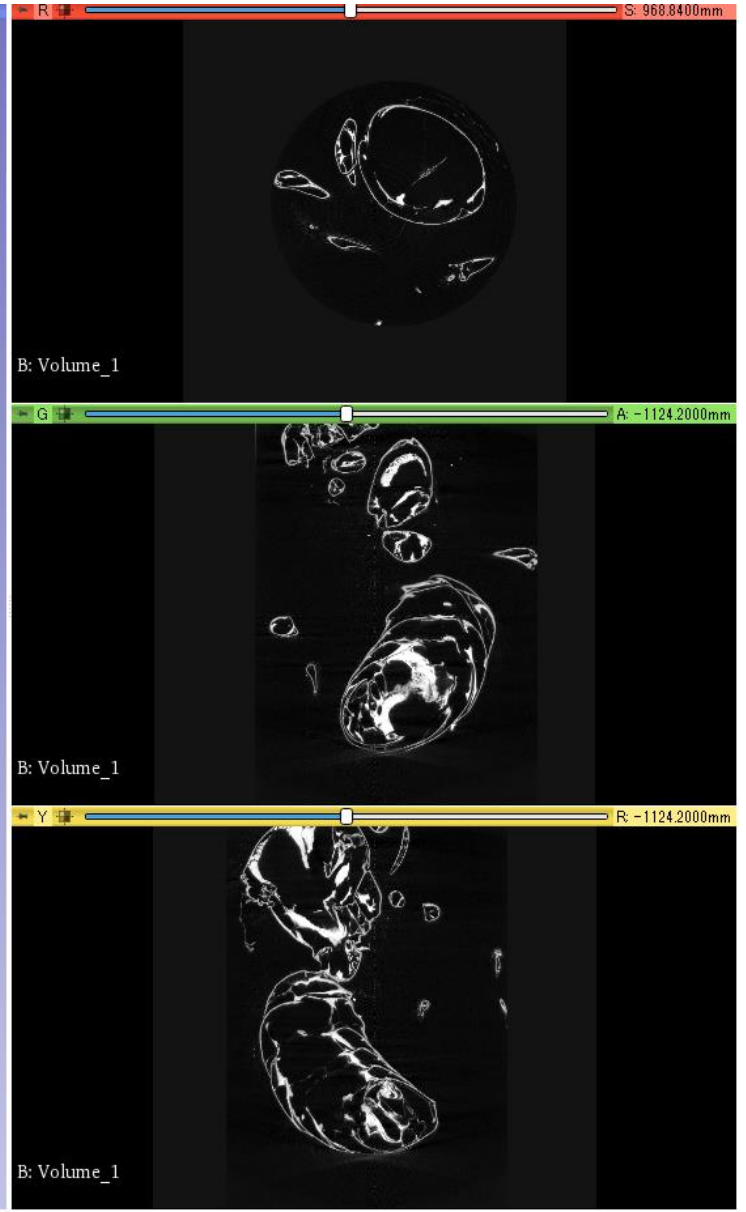


(a) Volume rendering of coffee bean. (b) A slice of size 4096×4096. (c) Sub-images of Figure b.

Fig 1: A generated volume data of **Coffee Bean** on ABCI. The size of volume data is 4096^3.

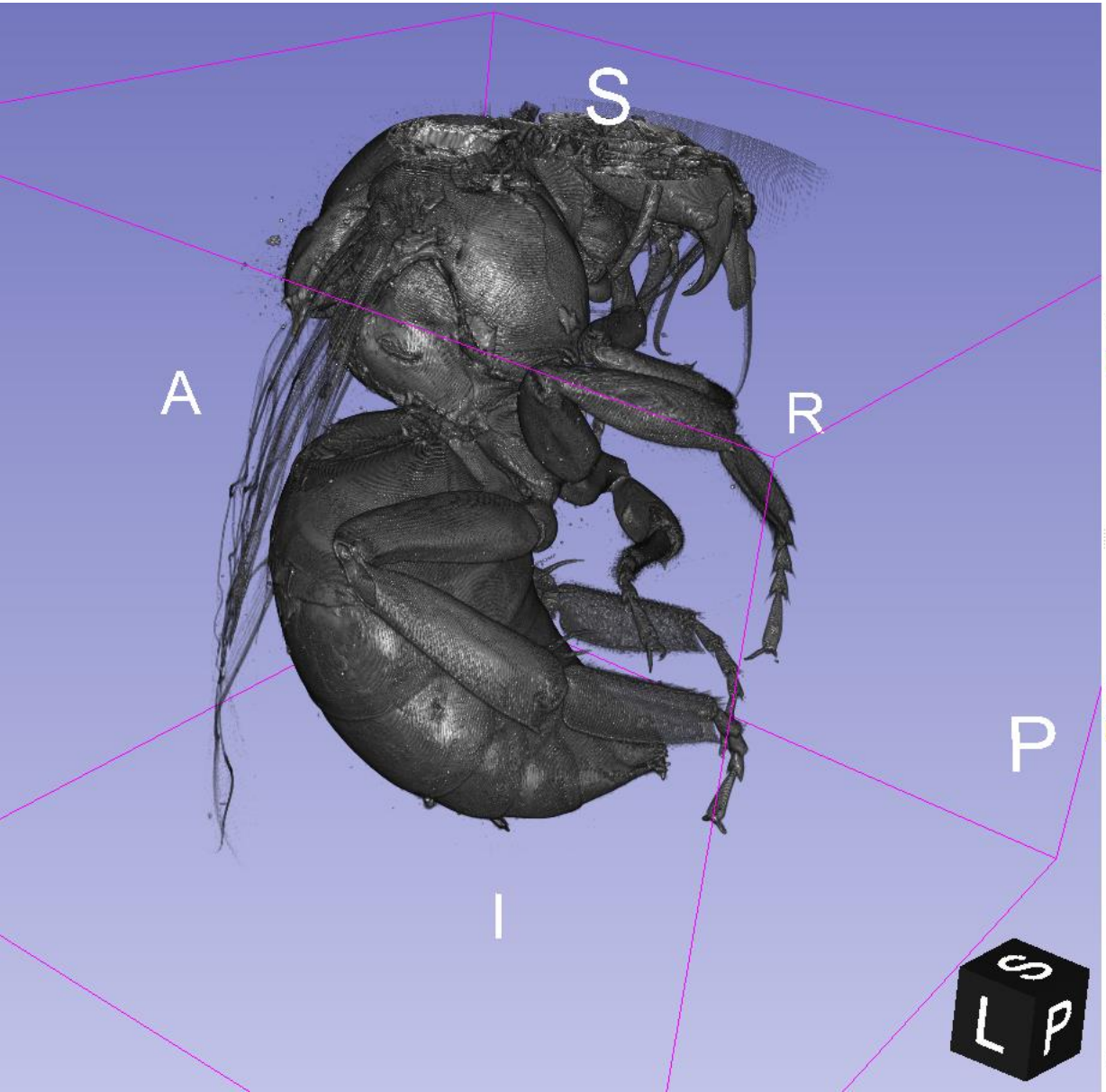
An example of High-resolution Volume data (Bumblebee)

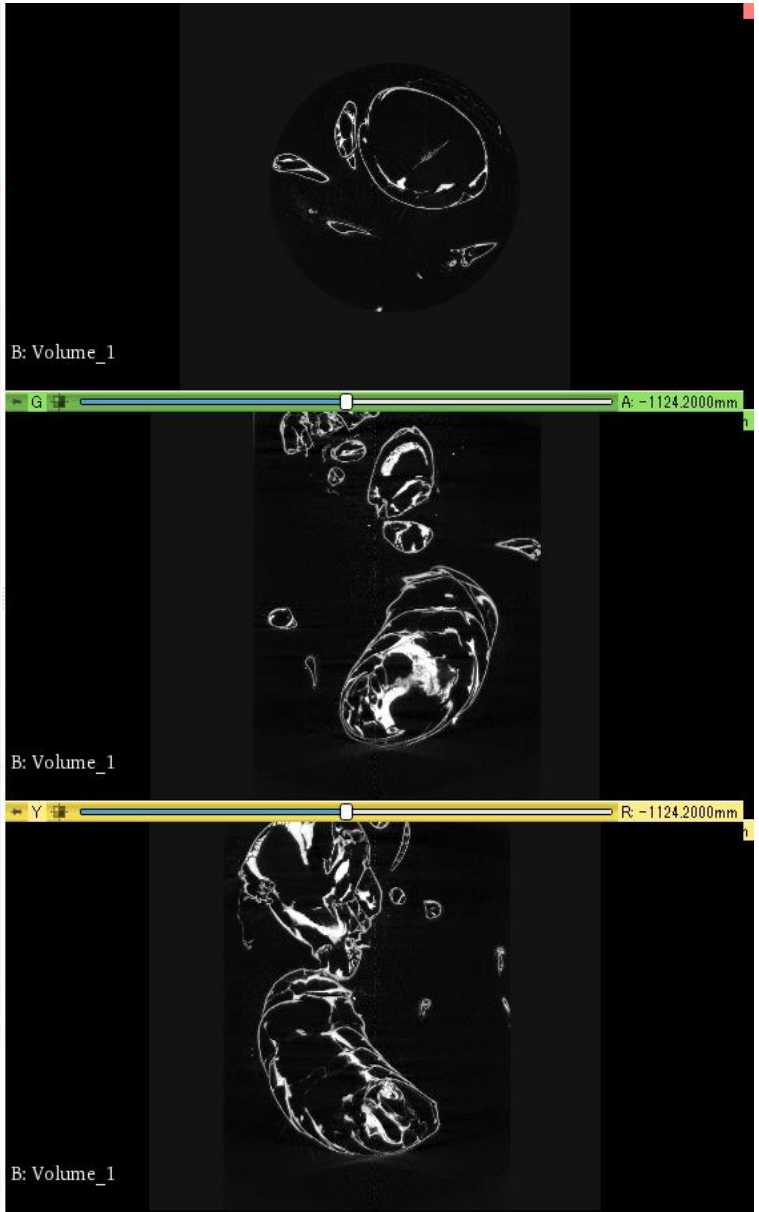




- Visualized by 3D slicer viewer

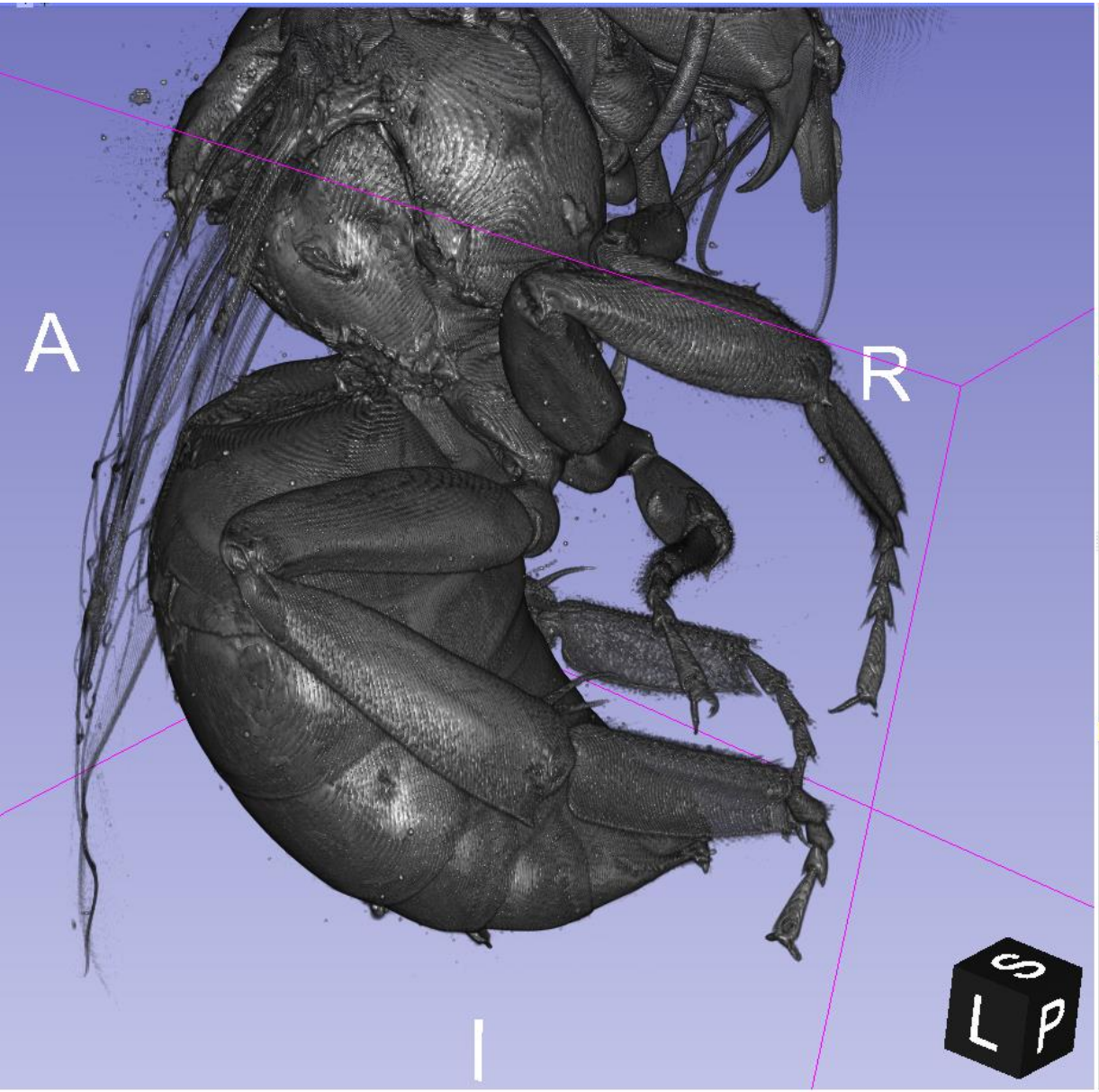
An example of High-resolution Volume data (Bumblebee)

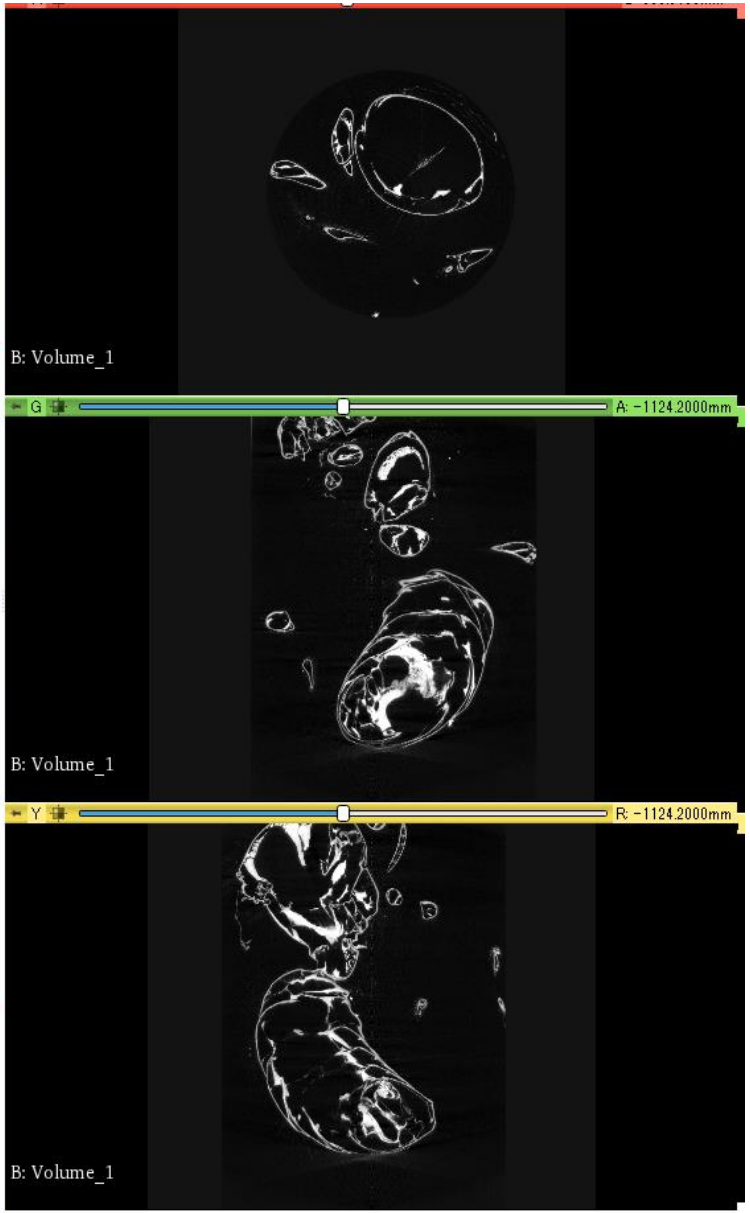




- Visualized by 3D slicer viewer

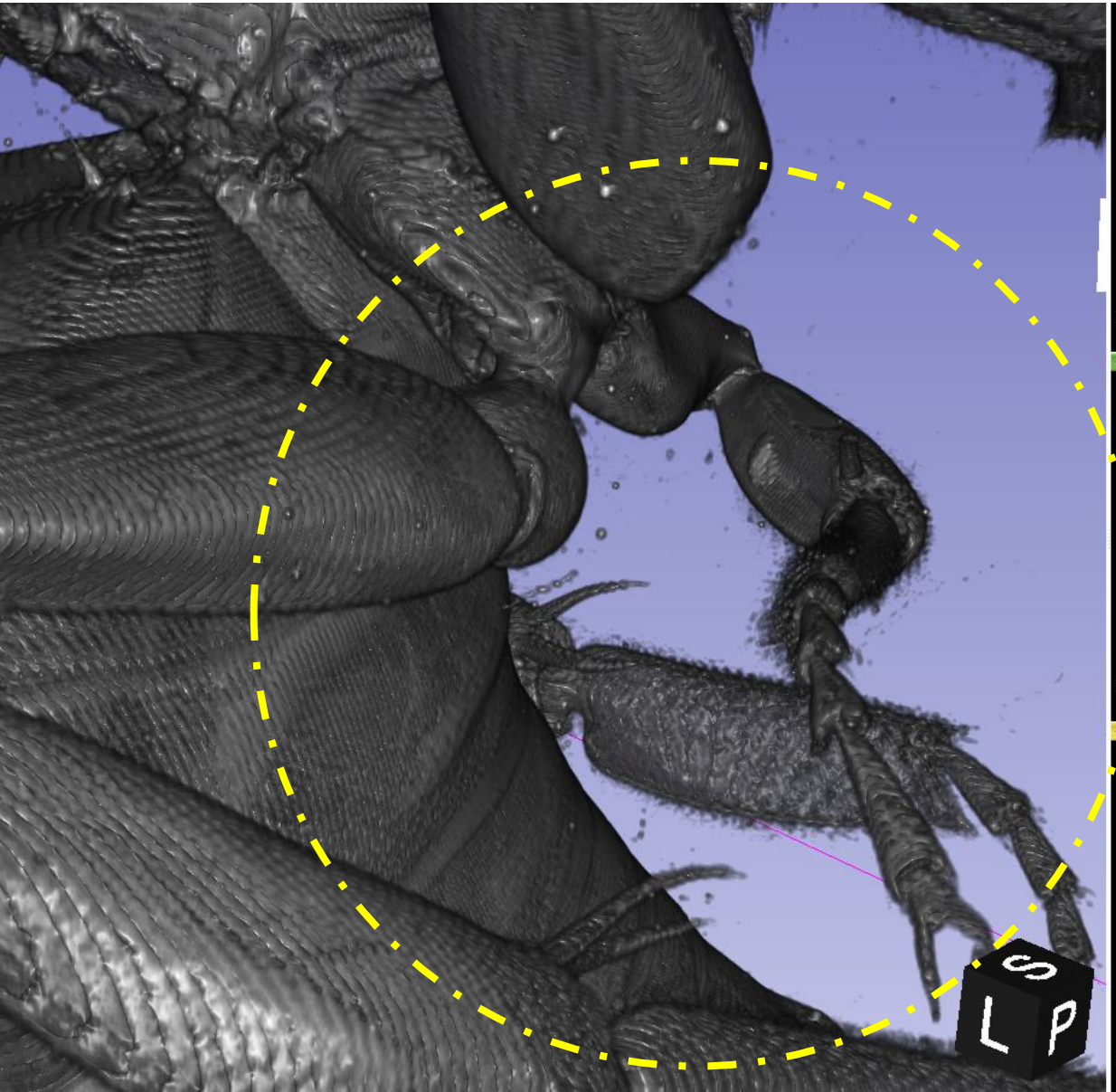
An example of High-resolution Volume data (Bumblebee)

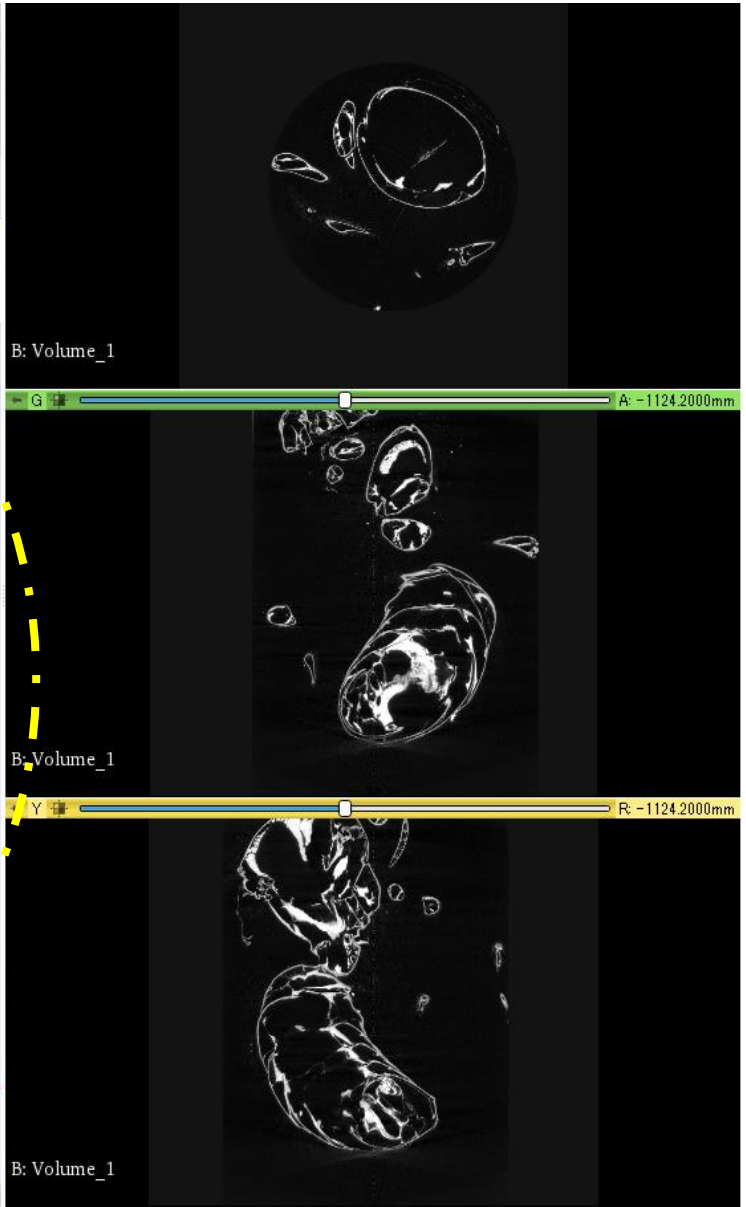




- Visualized by 3D slicer viewer

An example of High-resolution Volume data (Bumblebee)





- Visualized by 3D slicer viewer

Conclusion

1. We proposed a novel problem decomposition algorithm for FBP computation
2. We implemented an efficient CUDA kernel enabling out-of-core back-projection
3. We proposed a framework to generate high-resolution image
   - Two characteristics:
     - Pipeline processing
     - Parallel computation
   - Take advantage of the heterogeneity of GPU-accelerated supercomputer
     - Use CPU for filtering computation
     - Use GPU for back-projection
   - Ideal strong and weak scaling
4. Using up to 1,024 GPUs, we can generate 2048^3 and 4096^3 volume data in 3 seconds and 16 seconds (including I/O), respectively.

Future work

- Optimize the iterative reconstruction algorithms for CBCT on supercomputer
- Research on rendering High-resolution image in HPC
- Research on compressing the High-resolution images
- Provide an image reconstruction service via cloud

Thank You Very Much!