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

Peng Chen$^{1,3}$, Mohamed Wahib$^{3}$, Shinichiro Takizawa$^{3}$, Ryousei Takano$^{2}$, Satoshi Matsuoka$^{1,4}$

1. Tokyo Institute of Technology, Dept. of Mathematical and Computing Science, Tokyo, Japan
2. National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
3. AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory, National Institute of Advanced Industrial Science and Technology
4. RIKEN Center for Computational Science, Hyogo, Japan
Background

• Computed Tomography (CT) is widely used
  • Medical diagnosis
  • Non-invasive inspection
  • Reverse engineering

• Possibility of obtaining high-resolution image
  • Rapid development in CT manufacturing
  • CMOS-based Flat Panel Detector (FPD, X-ray imaging sensor) become larger
    • $2048 \times 2048$, $4096 \times 4096$, 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

What happens if we start manipulating \((6k)^3\) and \((8k)^3\) volumes? [1]

Harry E. Martz, Clint M. Logan, Daniel J. Schneberk, and Peter J. Shull

• High-resolution CT image is important but not attainable
  1. Intensive computation
  2. Critical timing demanding for image reconstruction
  3. Huge memory capacity
     • \(2048^3: 32\text{GB}, 4096^3: 256\text{GB}, 8192^3: 2\text{TB}\)

• We use GPU-accelerated supercomputers to solve this problem

• Challenges
  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?

Contributions

1. We proposed a **novel** back-projection algorithm
2. We implemented an efficient CUDA kernel for back-projection
3. We take advantage of the heterogeneity of modern systems
   - Use CPU for filtering computation
   - Use GPU for back-projection
4. We proposed a framework to generate high-resolution images
   - High performance
   - High scalability
5. Using up to 2,048 V100 GPUs, the 4K and 8K problems can be solved within 30 seconds and 2 minutes, respectively (including I/O)

\[
\begin{align*}
2K \text{ problem} & : 2048 \times 2048 \times 4096 \rightarrow 2048^3 \\
4K \text{ problem} & : 2048 \times 2048 \times 4096 \rightarrow 4096^3 \\
8K \text{ problem} & : 2048 \times 2048 \times 4096 \rightarrow 8192^3
\end{align*}
\]
Introduction of Compute Tomography

- CT system can generate **3D image** from a set of **2D projections (or images)**
- Cone Beam Compute Tomography (CBCT)
- CBCT Geometry & Parameter

![Diagram of CBCT geometry and trajectory](image1)

![Diagram of 3D volume geometry](image2)

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_u)</td>
<td>the number of 2D projections</td>
</tr>
<tr>
<td>(N_v)</td>
<td>the width and height of a 2D projection, respectively</td>
</tr>
<tr>
<td>(N_p)</td>
<td>the number of voxels in X, Y, Z dimension, respectively</td>
</tr>
</tbody>
</table>

- Reconstruction Problem Definition:
  \[N_u \times N_v \times N_p \rightarrow N_x \times N_y \times N_z\]
- Performance metrics:
  \[GUPS = \frac{N_x \times N_y \times N_z \times N_p}{T}\]
  where \(T\) is execution time in a unit of second.
FDK algorithm

- Presented by Feldkamp, Davis, and Kress in 1984 (35 years ago)
- FDK is also known as the Filtered Back-projection (FBP) algorithm
- 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, $O(N^4)$
- FFT primitive is required in Filtering computation
  - Intel IPP, MKL, cuFFT, etc.

![Diagram of FDK algorithm]

Load projections \[\rightarrow\] Filtering computation \[\rightarrow\] Back-projection computation \[\rightarrow\] Store volume

Filtered Back-projection (FBP)
Overview of the Proposed iFDK Framework

Proposed Framework on Multi-nodes with Multi-GPUs
Prior GPU-based work

- Implement filtering processing on GPU
  - cuFFT
- Rely on CUDA 2D-layered texture
  - Texture cache for spatial locality
  - Hardware interpolator (8-bits accuracy)
- Parallelize FDK on GPU without improving the algorithm
  - Arithmetic computation
  - Data locality
Proposed back-projection

Algorithm 2: Back-projection stage.

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

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

Input: $P_t, Q_t, N_p, N_x, N_y, N_z, t \in [0, N_p)$
Output: $I$
1: $I \leftarrow 0$
2: for $s \in [0, N_p)$ do
3:     for $k \in [0, N_x)$ do
4:         for $j \in [0, N_y)$ do
5:             for $i \in [0, N_z)$ do
6:                 $l \leftarrow \text{interp2}(Q_t, u, v)$
7:                 $y \leftarrow (P_t[1], [i, j, k, 1])$
8:                 $W_{dis} \leftarrow f^2$
9:                 for $k \in [0, N_z/2)$ do
10:                     $l(k, j, i) \leftarrow \hat{l}(k, j, i) + W_{dis} \cdot \text{interp2}(\hat{Q}_s, v, u)$
11:                     $\hat{k} \leftarrow N_z - 1 - k$
12:                     $\hat{v} \leftarrow N_z - 1 - v$
13:                     $\hat{l}(k, j, i) \leftarrow \hat{l}(k, j, i) + W_{dis} \cdot \text{interp2}(\hat{Q}_s, \hat{v}, u)$
14:                 end for
15:             end for
16:         end for
17:     end for

$P_s$ is a matrix of size $3 \times 4$

Note: We provide a theoretical prove in the paper of the correctness of the proposed algorithm
Proposed back-projection kernel on GPU

• We re-organize the loops
• We do not rely on texture cache
  • Use L1/L2 cache directly due to the better data locality
  • The locality is improved by using the transposed projections and volume
• We do not use texture interpolator
  • Achieve high precision of float32
• We compute a batch of 32 projections
  • Benefit to in-register accumulation
  • Reduce the global memory access
• We perform thread communication by shuffle intrinsic
  • Simple and efficient
• Detailed CUDA kernel can be found in our paper
An example of Problem Decomposition Scheme

Use 128 GPUs (32 Nodes) to solve a 2K problem.
Input: 4k count of $2k^2$ image, Output: $4k^3$

Input: 2D Projections

Output: 3D volume

Input

```
\begin{array}{ccc}
R_0 & R_1 & R_{31} \\
C_0 & C_1 & C_2 \\
0 & 32 & 64 \\
31 & 63 & 95 \\
\end{array}
```

16 MB/img x4096

11 MB/vol x64

MPI_Allgather

MPI_Reduce
Orchestration and Overlapping in iFDK

- Each MPI rank launches two extra-threads by pthread library
- Filtering thread launches multiple OpenMP threads for filtering computation

(a) Processing pipeline by three threads

(b) Reduce and Store operations by Main Thread
An example of achieved overlapping

- An example of pipeline to solve $2048 \times 2048 \times 4096 \rightarrow 4096^3$ problem
- Use 128 V100 GPUs
- Filtering thread processes 32 projections
- Main thread gathers 1024 projections
- Back-projection thread processes 1024 projections

Filtering thread
Main thread
BP thread

- Exchange data
- Load & Filtering
- MPI-AllGather
- H2D copy
- Back-projection
- D2H copy
- MPI-Reduce
- Store to PFS
An example of MPI_Reduce operation

- Use 16 GPUs to solve $2048 \times 2048 \times 4096 \rightarrow 2048^3$ problem
- Each GPU process a sub-volume of size 8GB
Performance model

• Micro-benchmarking
  • To better understand the characteristics of our system
  • Measure the constant parameters of our system, e.g.
    • Bandwidth of Parallel File System (PFS)
    • Bandwidth of PCIe connector
    • Throughput of MPI primitives

• Building a performance model
  • We can predict the potential peak performance
  • We can justify the scalability of iFDK
    • iFDK scales with the number of GPUs ($N_{gpus}$) linearly

• Detailed equations can be found in our paper
Evaluation environment

• ABCI supercomputer
  • Constructed and operated by AIST
  • 1,088 computing nodes, 4,352 Tesla V100 GPUs

• Software
  • CentOS 7.4
  • CUDA 9.0
  • Intel library 2018.2.199 (MPI, IPP)
  • RTK (Reconstruction Toolkit) 1.4.0

• Evaluation dataset
  • Computation complexity is independent of the content of projections
  • Use Shepp-Logan phantom
  • Generate projections by RTK library
# ABCI Compute Node

**FUJITSU PRIMERGY Server (2 servers in 2U)**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Xeon Gold 6148 (27.5M Cache, 2.40 GHz, 20 Core) x2</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA Tesla V100 (SXM2) x4</td>
</tr>
<tr>
<td>Memory</td>
<td>384GiB DDR4 2666MHz RDIMM</td>
</tr>
<tr>
<td>Local Storage</td>
<td>1.6TB NVMe SSD (Intel SSD DC P4600 u2) x1</td>
</tr>
<tr>
<td>Interconnect</td>
<td>InfiniBand EDR x2</td>
</tr>
</tbody>
</table>

**Connectivity**

- **IB HCA (100Gbps)**
  - Xeon Gold 6148 to Xeon Gold 6148: 10.4GT/s x3 with UPI x3
  - Xeon Gold 6148 to DDR4-2666 32GB x 6: 128GB/s
  - Tesla V100 SXM2 to Tesla V100 SXM2: NVLink2 x2

**Networking**

- PCIe gen3 x16
  - Xeon Gold 6148 to x48 switch
  - Xeon Gold 6148 to x64 switch

**Storage**

- NVMe
  - Xeon Gold 6148 to NVMe

**Compute**

- Intel Xeon Gold 6148 (2 x 6 cores)
- NVIDIA Tesla V100 SXM2 (4 x 100Gbps)
- DDR4-2666 32GB x 6
- NVLink2 x2
Evaluation on back-projection kernels

- A single Tesla V100 GPUs
- **Single precision**
- Up to **1.6 times faster** than baseline

Performance summary
- Better data locality
- Advantaged use of L1 cache
- Efficient data communication

<table>
<thead>
<tr>
<th>FDK problem</th>
<th>Baseline (GUPS)</th>
<th>Ours (GUPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel→voxel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512²×1k→128³</td>
<td>65.3</td>
<td>118.0</td>
</tr>
<tr>
<td>512²×1k→256³</td>
<td>107.4</td>
<td>188.6</td>
</tr>
<tr>
<td>512²×1k→512³</td>
<td>115.1</td>
<td>206.0</td>
</tr>
<tr>
<td>512²×1k→(1k)³</td>
<td>118.1</td>
<td>211.4</td>
</tr>
<tr>
<td>512²×1k→(1k)²×2k</td>
<td>N/A</td>
<td>212.7</td>
</tr>
<tr>
<td>(1k)³→128³</td>
<td>41.9</td>
<td>27.2</td>
</tr>
<tr>
<td>(1k)³→256³</td>
<td>77.4</td>
<td>83.7</td>
</tr>
<tr>
<td>(1k)³→512³</td>
<td>115.7</td>
<td>190.3</td>
</tr>
<tr>
<td>(1k)³→(1k)³</td>
<td>117.9</td>
<td>205.7</td>
</tr>
<tr>
<td>(1k)³→(1k)²×2k</td>
<td>N/A</td>
<td>207.9</td>
</tr>
<tr>
<td>(2k)²×1k→128³</td>
<td>16.1</td>
<td>7.7</td>
</tr>
<tr>
<td>(2k)²×1k→256³</td>
<td>38.6</td>
<td>24.1</td>
</tr>
<tr>
<td>(2k)²×1k→512³</td>
<td>80.2</td>
<td>81.6</td>
</tr>
<tr>
<td>(2k)²×1k→(1k)³</td>
<td>116.9</td>
<td>186.9</td>
</tr>
<tr>
<td>(2k)²×1k→(1k)²×2k</td>
<td>N/A</td>
<td>198.7</td>
</tr>
</tbody>
</table>

The higher, the better
Weak scalability evaluation

- Using up to 2048 GPUs to evaluate a **4K problem**
- Peak performance is predicted by our performance model
- We achieve outstanding weak scaling

Weak scaling for $2048^2 \times N_p \rightarrow 4096^3 \cdot N_p = 16^* N_{gpu}$
Weak scalability evaluation

- Using up to 2048 GPUs to evaluate a **8K problem**
- Peak performance is predicted by our performance model
- We achieve outstanding weak scaling

**Weak scaling for** $2048^2 \times N_p \rightarrow 8192^3$. $N_p = 4 \times N_{gpus}$.
Strong scalability evaluation

- Using up to 2048 GPUs to solve a **4K problem**
- Peak performance is predicted by our performance model
- We can achieve about 76% of the peak performance
Strong scalability evaluation

- Using up to 2048 GPUs to solve a **8K problem**
- Peak performance is predicted by our performance model
- We can achieve about 76% of the peak performance

![Graph showing runtime vs. number of GPUs]

**Strong scaling for** $2048^2 \times 4096 \rightarrow 8192^3$. 
Performance

- Extremely high performance
- Higher computational intensity, better scalability
- Bottleneck becomes the data movement
- Over two order of magnitude faster than a single Tesla V100 GPU
- Solve any FDK problems instantaneously

The achieved performance of solving 2K, 4K, and 8K problems
Conclusion

1. We proposed a general FDK algorithm
2. We implemented an efficient CUDA kernel for back-projection
3. We proposed a framework (iFDK) to generate high-resolution image
   • Two characteristics:
     • Pipeline processing
     • Parallel computation
   • Take advantage of the heterogeneity of modern systems
     • Use CPU for filtering computation
     • Use GPU for back-projection
   • Almost ideal Strong and weak scaling
4. Using up to 2,048 V100 GPUs to solve a 4K and 8K problems within 30 seconds and 2 minutes, respectively (including I/O).
Future work

• Research on rendering High-resolution image in HPC
• Research on compressing the High-resolution images
• Provide an image reconstruction service via cloud