Parallelizing Cryo-EM 3D Reconstruction on GPU Cluster with A Partitioned and Streamed Model

Kunpeng Wang
wkp16@mails.tsinghua.edu.cn
Tsinghua University
National Supercomputing Center in Wuxi, China

Shizhen Xu*
xsz12@mails.tsinghua.edu.cn
Tsinghua University

Haohuan Fu†
haohuan@tsinghua.edu.cn
Tsinghua University
National Supercomputing Center in Wuxi, China

Hongkun Yu
yhk15@mails.tsinghua.edu.cn
Tsinghua University
National Supercomputing Center in Wuxi, China

Wenlai Zhao
cryinlaugh@gmail.com
Tsinghua University
National Supercomputing Center in Wuxi, China

Guangwen Yang†
ygw@tsinghua.edu.cn
Tsinghua University
National Supercomputing Center in Wuxi, China

ABSTRACT

As a vital approach to determine the structure of biomacromolecules, high-resolution cryo-electron microscopy (cryo-EM) 3D reconstruction is extremely compute-intensive, and has gradually migrated to GPU accelerators in recent years. With certain kernels already achieving high speedup and efficiency on GPUs, the reconstruction part, which inherently requires accesses of a large 3D model in different orientations, brings tough challenges to GPU architectures and has no effective GPU-based options. To fill the above gap, in this paper, we propose Stream3D, a novel GPU-based parallel design for cryo-EM 3D reconstruction. Our major idea is to reorganize the related problem space as streams of key-value pairs, so that we can achieve both the flexibility and efficiency to compute and accumulate the contribution to the final 3D model from all different 2D image inputs. In addition, we design a hybrid communication mechanism to reduce intra-node communications and enable the solving process on a larger scale. With the addition of our GPU-based reconstruction design, we are able to improve the performance of the reconstruction part itself by 9.50 times, and the performance of the entire processing part (the reconstruction part and the other parts with mature GPU options) by 2.83 times. Moreover, Stream3D enables using the approach at a large scale, with 65.32-fold speedup when using up to 80 GPUs.

CCS CONCEPTS

• **Applied computing → Computational proteomics**.

*Shizhen Xu conducted this work while he was a Ph.D. student at Tsinghua University.
†Corresponding author. (Haohuan Fu and Guangwen Yang)

ICS '19, June 26–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6079-1/19/06…$15.00
https://doi.org/10.1145/3330345.3330347

KEYWORDS

Cryo-EM 3D reconstruction, GPU computing

ACM Reference Format:
Kunpeng Wang, Shizhen Xu, Haohuan Fu, Hongkun Yu, Wenlai Zhao, and Guangwen Yang. 2019. Parallelizing Cryo-EM 3D Reconstruction on GPU Cluster with A Partitioned and Streamed Model. In *2019 International Conference on Supercomputing (ICS '19), June 26–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3330345.3330347

1 INTRODUCTION

Cryo-electron microscopy (cryo-EM) techniques, which were awarded the 2017 Nobel Prize in chemistry, have been regarded as the most promising techniques to acquire the structure of biomacromolecules from cryo-EM images at near-atomic resolutions [5, 12, 16, 17]. At the core of these techniques lies the three-dimensional reconstruction (**3D reconstruction**), which plays a central role in completing the final 3D determination. However, the prominent successes achieved in recent years [2, 32] are largely driven by the complex algorithm and huge image dataset (~10^5 images), which in return bring great computational challenges. It usually takes tens to hundreds of hours to produce a medium size model on a lab cluster with thousands of CPU cores [6, 35]. The computational requirements are currently becoming the main constraints for the application of more complex methods, such as the statistic-based Bayesian approaches [24], and high-resolution large structure determination [25].

Heterogeneous many-core systems, such as GPU clusters that feature tremendous computing power and favorable cost effectiveness, have become an attractive choice to address the increasing computational demand in cryo-EM field. Prior research [15, 26, 37] have put enduring efforts on GPU-accelerated cryo-EM structure determination and have shown promising speedups. They mostly focused on the image correlation, contrast, alignment and etc., which are largely dominated by element-wise operations. In contrast, the 3D reconstruction part, which inherently requires accesses of a large 3D model in different orientations, bring significantly tougher challenges for the GPU architectures, and still lack effective GPU-based solutions. As a result, with the other parts accelerated by GPUs, the 3D reconstruction part is occupying larger and larger

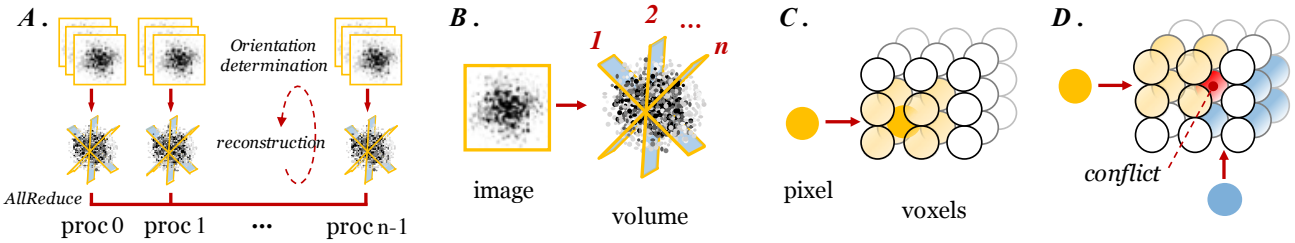

Figure 1: The 3D reconstruction in cryo-EM and its computational challenges. *A.* The 3D reconstruction workflow and its straightforward data parallel design which is normally bounded by the GPU memory capacity constraints and suffers from high communication overhead when dealing with large models. *B.* The demand to process a number of different image orientations imposes a great challenge for effective partition of the problem space. *C.* The discontinuous memory access when inserting a pixel into volume. The value of a pixel in image contributes to voxels in a cubic neighborhood area in volume. *D.* Data conflicts when simultaneously inserting two pixels which are close enough to each other.

portions (more than 70% according to profiling results of [9]) of the total run time.

The challenge of the 3D reconstruction part mainly comes from two factors. The first one is the requirement to access different 2D images with different orientations in the 3D model volume, as shown in Figure 1. In such a processing pattern, the traditional spatial partition or blocking strategies are ineffective. A straightforward approach is to store the entire 3D model in each single node or device, which leads to expensive memory and communication cost. The second factor is the discontinuous memory accesses. Again, when mapping the contribution of the final 3D result from numerous 2D images with different orientations, i.e., computing voxels from pixels, we face complicated and largely distributed memory access patterns.

As a result, we see few efforts that try to achieve an efficient design of the 3D reconstruction part. Among the most widely used software packages in cryo-EM labs such as Relion-2 [11], the code is still at the stage of accomplishing functions, rather than considering efficiency. While they already provide GPU accelerations for most of the modules, the reconstruction part is still pure-CPU code.

Aiming at a comprehensive solution, we intend to tackle the challenging problem from an *element-centric view*, which concentrates on the *working* partitions of the model, instead of the *model-centric view* which holds the entire model on device. We further extend this basic idea based on two key observations. First, each image only updates a 2D plane of the 3D model at a specific angle, which means only a subset of the referenced voxels is required. Second, by treating the voxel subset as the main problem space, the irregularity and contention could be better regularized and controlled by designing a dedicated memory layout. These features inspire us to reorganize the model as a large set of voxels, and design a novel parallel method with a partitioned model.

In this paper, we propose Stream3D, a novel parallelizing design with a partitioned and streamed model abstraction for cryo-EM 3D reconstruction. Our major idea is to reorganize the related problem space as streams of key-value pairs, so that we can achieve both the flexibility and efficiency to compute and accumulate the contribution to the final 3D model from all different 2D image inputs. By reshaping and compressing the voxel streams, it further improves the memory access efficiency and greatly reduces the memory consumption. In addition, we design a hybrid communication mechanism to reduce intra-node communications and enable the solving process on a larger scale.

Experiments show that, with the addition of our GPU-based reconstruction design, we are able to improve the performance of the reconstruction part itself by 9.50 times, and the performance of the entire processing part (the reconstruction part and the other parts with mature GPU options) by 2.83 times. Moreover, Stream3D enables using the approach at a large scale, with 65.32-fold speedup when using up to 80 GPUs.

2 BACKGROUND AND MOTIVATION

2.1 3D Reconstruction Algorithm in cryo-EM

Cryo-EM images are projections of an macromolecule, e.g. protein, from different project directions. The relationship between an object and its projection is referred to as **central slice theorem** [18, 23]: the Fourier transformation of the projection of a 3D object is the central 2D plane cross-section (*i.e., passing though the origin of Fourier space as illustrated in Figure 1-A/B*) of the object's 3D transform and is perpendicular to the projection direction. A Fourier transformation on these images generates a number of cross-section slices. Therefore, when the number of slices is enough to retrieve information in 3D Fourier space, an inverse Fourier transformation can be applied to recover the original 3D model. Most computations occur at the steps of merging the slices, i.e. the transformed images, into 3D Fourier space by rotation, translation and interpolation.

The above idea forms the basics of *direct Fourier inversion*, a transform algorithm currently considered the most accurate reconstruction method [19]. The direct Fourier inversion method directly and exclusively operates in Fourier space by casting the problem as one of Fourier space interpolation between polar system of coordinates of the projection and Cartesian system of coordinates of the object [1, 27]. The algorithm comprises four steps: (1) Use reverse gridding method to resample 2D input projection images into 2D polar coordinates. (2) Calculate the convolution for the Fourier transform of each projection by $\sum_i \mathcal{F}[w] * (c\mathcal{F}[g_i])$,

Algorithm 1 The General Reconstruction Procedure

Input: Image set $\mathcal{I} = \{image_i\}$, rotation matrix $\mathcal{R} = \{r_i\}$
Output: Model volume F, weight volume C

```
1:  F ← 0, C ← 0
2:  for image_i ∈ I do
3:      for pixel_j ∈ image_i do
4:          voxel = CoordinateTransform(pixel_j, r_i)
5:          weight[ ] = Interpolation(voxel)
6:          for voxel_k ∈ neighbor(voxel) do
7:              F[voxel_k.index] += voxel × weight[k]
8:              C[voxel_k.index] += weight[k]
9:          end for
10:     end for
11: end for
```

where c are gridding weights and $\mathcal{F}[w]$ is an appropriately chosen convolution kernel, g is 2D projections. (3) Use 3D inverse FFT to compute $wd = \mathcal{F}^{-1}[\mathcal{F}[w] * \mathcal{F}[d]]$ on a Cartesian grid. (4) Remove the weights using real-space division $d = (wd)/w$. The major computational complication lies at step 2 which is essentially a convolution. In practice, a modified trilinear interpolation scheme is always used as simplified replacement of the convolution. To highlight the reconstruct logic, we show the actual calculation kernel briefly in Algorithm 1, with element-wise operation and (inverse) FFT omitted.

2.2 Challenges of 3D Reconstruction on GPUs

Huge models. Potentially, the most critical impediment to apply GPU in 3D reconstruction is the limited on-device memory. Table 1 shows common model configurations with memory consumption ranging from 0.52 GB to 71.99 GB. Prior practices [7], implied in Algorithm 1 Line 7-8, which hold the entire model on GPU are not applicable when dealing with the medium and large cases [1].

Table 1: Common Model Configuration.

Model Size	360	680	1000	1220	1540	1860
Memory Consume(GB)	0.52	3.52	11.20	20.33	40.87	71.99

The 3D model size is represented by the number of voxels (n.o.v) of one dimension (the other two dimensions have the same n.o.v.), we use this as a convention in following. Model is measured in double precision.

However, finding an effective model partition method is nontrivial. Because the orientation parameters of images are undetermined and updated during every iteration, each single image can be potentially interacted with any parts of the volume. Traditional 2D or 3D partition schemes may split the volume into several regular parts, and then distribute the different parts among different processes. However, the nondeterministic orientations of images could potentially require each process to reference all the other images. The communication overhead will increase linearly with the number of computing nodes, while the image data utilization will decrease linearly, leading to a non-scalable design. Besides, the centralized distribution of voxels in volume also presents a challenge for this design to achieve load balance. In fact, the huge memory consumptions of large models influence the parallelization design. *Data-parallel*, in which each GPU preserves a model replica and communicates the entire model every iteration, suffers from huge communication overhead.

Discontinuous and irregular memory accesses. The memory access behavior of merging images into volume exhibits severe discontinuity. In the interpolation procedure, the value of one image pixel needs to be added to tens of voxels in a cubic area of model, which are not continuous in a normal memory layout. Under the current parallelization policy [9], each GPU thread is responsible for the computation of one (or several) pixel interpolation. As shown in Figure 1, these threads as a whole update a 2D plane of the 3D model at a nondeterministic angle, which makes it impossible to identify a suitable thread mapping that achieve coalesced memory access. A common practice is to assign continuous thread with the task of inserting continuous pixels to coalesce the pixel read access. However, considering that the image can be inserted by an arbitrary orientation, the coalescing of voxel access will be significantly reduced. At the same time, the 2D plane is only accessed once during insertion of one image, so that caching it in shared memory can hardly benefit.

Low computational efficiency is another potential problem resulted from the excessive data contention inherited in merging images into volume. Multiple GPU threads might update the same model voxel during the interpolation. This is a common scenario when adjacent pixels are projected to voxels near the model center. Atomic operations are introduced to guarantee the correctness, while their overhead is non-negligible. Although a proposed method [35] alleviates such contentions by increasing the distance of pixels that are concurrently inserted into the volume, it actually reduces the parallelism and has negative effects on performance.

2.3 Motivation

To solve the problem, we first review the 3D reconstruction process. Two key observations throw lights on our design. First, when merging an image into the volume, only a subset of the voxels near the corresponding slice plane will be updated. Thus *there is no need to store the whole model on the device* if we insert a certain batch of images each time. Second, without storing the whole model, the irregular memory access to update the voxels in a cube of volume can be eliminated. Because *no-whole-model design provides us with a flexibility to reshape the memory layout* for storing the generated voxels in a regular and continuous way. Inspired by these observations, we rethink the reconstruction calculation from an element-centric view, in which pixels and voxels are treated as first class objects to organize the computation.

3 STREAM3D: REORGANIZING THE PROBLEM AS KEY-VALUE STREAMS

3.1 Key-Value Expression for Reconstruction

The first design process that we perform is to identify an effective model representation that can help both the expression of the algorithm and the efficiency of the computation. In our proposed key-value abstraction, a key represents the coordinate index of a

[1] Current GPUs usually own 6 GB to 16 GB on board memory.

Algorithm 2 Identical Thread-mapping

Input: Image set $\mathcal{I} = \{image_i\}$, rotation matrix $\mathcal{R} = \{r_i\}$
Output: $voxel\_stream = [kvparis_i]$

```
1:  F ← 0, C ← 0
2:  for image_i ∈ I do                                   ▷ parallelized
3:      for pixel_j ∈ image_i do                         ▷ parallelized
4:          voxel = CoordinateTransform(pixel_j, r_i)
5:          weight[ ] = Interpolation(voxel)
6:          for voxel_k ∈ neighbor(voxel) do
7:              kv = make_pair(Δvoxel_k, weight[k])
8:              voxel_stream.write(kv)
9:          end for
10:     end for
11: end for
```

Algorithm 3 Adaptive Thread-mapping

Input: Image set $\mathcal{I} = \{image_i\}$, rotation matrix $\mathcal{R} = \{r_i\}$
Output: $voxel\_stream = [kvparis_i]$

```
1:  F ← 0, C ← 0
2:  define < k, v >=< index, voxel{values, weight} >
3:  shared memory : s_vxl[ ], s_wet[ ]
4:  for image_i ∈ I do                                   ▷ parallelized
5:      for pixel_j ∈ image_i do                         ▷ parallelized
6:          s_vxl[i, j] = CoordinateTransform(pixel_j, r_i)
7:          s_wet[i, j] = Interpolation(s_vxl[i, j])
8:      end for
9:      for k ∈ neighbor({< [i, j], _ >}) do             ▷ parallelized
10:         kv = make_pair(Δs_vxl[i, j], s_wet[i, j], k)
11:         voxel_stream.write(kv)
12:     end for
13: end for
```

voxel, and the value is a structure storing the voxel's numeric information, including a complex value in Fourier Space and a weight value. Treating the model voxels as streams of key-value pairs, the reconstruction process can be expressed as accumulating the values with identical keys into the model. By storing the voxels in key-value pairs, we can express the reconstruction calculation in a more memory-efficient style, as illustrated in Algorithm 2 Line 7-8. Compared with Algorithm 1, first, there is no need to reference the whole model; and second, the memory access behaviors can be reorganized by reordering the voxels.

In fact, the key-value pair representation is a kind of deconstruction to the original structured model representation in that the calculations are reformed to generate voxel streams without referencing the whole model. Besides, two attractive properties of key-value representation, associativity and commutativity, make it naturally straightforward to express parallelism.

3.2 Leverage the Key-Value Abstraction for GPU Computing

We present Stream3D's design of mapping the exposed parallelism to GPU architecture, leveraging key-value expression.

Eliminate the un-coalesced memory access. The key-value representation provides us with an extra opportunity to redesign the global memory layout and regularize the access pattern. Stream3D exploits this flexibility by an adaptive thread mapping in a two-phase design as illustrated in Algorithm 3 Line 5–12. The first phase is a pixel centric calculation during which a pixel is normalized, transformed to 3D volume space and evaluated by interpolation. All of these computations are performed independently on each pixel. So it's an element-wise operation and free to be mapped in a low-level scheme that each single thread processes a single pixel. In the second phase, our focus changes to voxel centric calculation. The task of this phase is to complete the interpolation, i.e. for each voxel we calculate all the contributions of the pixel and save in key value pairs. This update operation is again a voxel independent calculation and is assigned to a single thread. Both pixel-thread mapping and voxel-thread mapping ensure the coalesced memory access in a way that adjacent threads read adjacent pixels in the first phase and write to adjacent voxels in the second phase.

Leverage shared memory that being hardly to benefit from in prior design. According to our two-phase design, the intermediate results are well suited to be stored in shared memory. By design, the two-phase arrangement does require a stage memory for phase transfer as two different thread mapping schemes are used respectively. The fast on-chip shared memory serves this purpose directly. Another attractive feature is that shared memory is much less sensitive to un-coalesced memory access, so the impact of less regular writing operations to store intermediate results in the first phase is reduced. Moreover, the use of shared memory greatly reduces the number of registers consumed by the streaming kernel and thus increases the SM occupancy, which is beneficial to achieving high computing throughput.

Conflict-free by ordered key. One major challenge for 3D reconstruction is the voxel area's overlapping nature, which inevitably leads to write conflict. In Stream3D, this problem is effectively solved by the sort-by-key operation, which is originally designed for implementing reduce-by-key. The attribute that the voxel stream offloaded from the device is sorted by the coordinate index makes the final model update operation on host conflict-free.

4 PARTITION AND REDUCTION STRATEGIES

Different from the model-centric view which holds the entire model on device, Stream3D employs an element-centric view which concentrates on the working voxels of the model, thereby is more efficient for both computation and memory, as illustrated in Figure 2. Leveraging the key-value abstraction, Stream3D successfully breaks up the memory bottleneck and de-structures the model into a large set of voxels. However, this is achieved at the cost of migrating memory access from on-chip interconnection, i.e. memory bus, to off-chip interconnection, i.e. PCIe. Considering the bandwidth and latency gap, a straightforward stream offloading implementation will degrade the performance seriously.

To avoid this degradation, Stream3D exploits an important feature derived from the central slice theorem, as referred in section 2.1, that all images are central cross-sections of the object's 3D transform which pass through the origin. It means for every two images, and their corresponding generated voxel sets, there is an overlapping area. Therefore, this implies a huge *reduction* space which can be leveraged to minimize the overhead of voxel stream offloading.

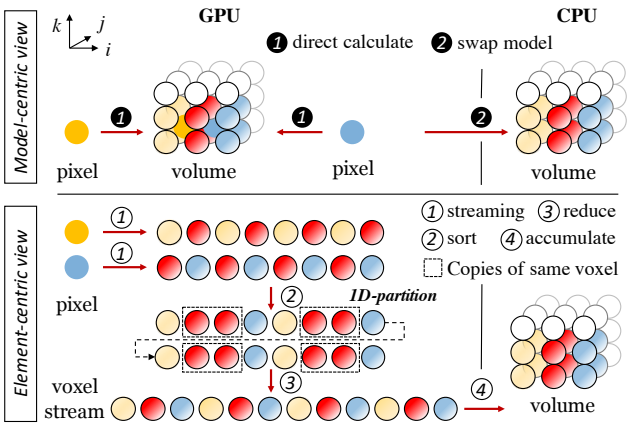

Figure 2: Comparison of Stream3D's element-centric computation view with model-centric view. The *yellow* and *blue* pixels generate the corresponding voxels in the same color, and the *red* ones indicate the reduction opportunity.

Stream3D is designed with a *partition scheme* on the voxel set to facilitate reduction. The total set of voxels is divided into several subsets according to the images which they are generated from. Each voxel subset, represented in key-value pairs and linearly ordered, forms an **1D-partition** (one dimensional partition), as illustrated in Figure 2. Since they are from the same image, voxels in the same partition share the same weight and orientation parameters during insertion. Every two 1D-partitions have an overlapping area near the center of the model; and a global reduction on all the 1D-partitions produces the complete 3D volume. This partition scheme enables the *batching* and *reduction* mechanism of Stream3D, which are critical for maximizing fine-grained parallelism, reducing transfer overhead and sustaining a high computing throughput.

4.1 Batching

Two phases of reconstruction process employ batching strategy in Stream3D. One is *batching* the images to be uploaded to devices, and another is to perform the reduction operation on *batched* 1D-partitions. A major consideration on batching uploaded images is to expose maximal parallelism at pixel level while better accord with GPU architecture's mechanism of hiding latency. The batch size, i.e. the number of images in a batch, has a direct relation to the configuration of CUDA thread blocks when launching kernel, and further has a severe impact on the occupancy of the Streaming Multiprocessors (SMs). A relative high occupancy is critical to SM's execution efficiency since there are more warps to switch to hide latency when one warp is paused or stalled on an SM. The batching 1D voxel partitions is mainly designed to amortize the data transfer overhead and more importantly, provides a granularity control for reduction, which is presented below.

4.2 Multi-Reduction

The overlapping feature of 1D voxel partitions provides us with a huge space for compression. In view of this, Stream3D is designed with a reduction phase to reduce the redundant data transfer by aggregating all the generated voxels in all potential 1D-partitions.

The reduction phase is performed after the streaming stage by means of sort-based *reduce-by-key* operation, and is organized by a three-level design:

- The first level is inside a 1D-partition to reduce the overlapping voxel areas generated by adjacent pixels.
- The second level is in a batch which makes use of the intersection of different 1D-partitions in a batch, a feature derived from the central slice theorem.
- The third level is an across-batch reduction which further aggregates the redundant voxels from different batches.

Each level concentrates on a distinct aspect of the reduction space. And a thorough three-level reduction on the whole 1D-partition set will produce the complete model.

The reduction operation in fact trades computations for memory transfer which offloads more computations on GPU. This will make influences both on CPU-GPU cooperation and computation-transfer overlapping on devices. Therefore, to achieve seamless overlapping and cooperation, an adaptive reduction strategy is used in Stream3D by dynamically rebalancing the reduction ratio with the CUDA stream overlapping outcome. During the reduction phase, the Stream3D will control the level of reduction to be performed by tracing the reduction ratio of each batch. Besides, Stream3D also stages the reduced 1D-partition batches on device whenever the global memory is available. Based on the statistics of processed batches, it estimates the up coming batch's reduction ratio and then makes a decision on whether offloading the staged ones or performing another reduction. Experiments show that more than 70% of voxels, with modest model size and batch size, can be reduced, thereby the memory copy overhead from device to host is greatly decreased.

4.3 Discussion on Reduction Ratio

Theoretically, the larger the batch is, the more the the reduced voxels are. We explain this by a simple quantitative analysis. Let p denote the total number of the original voxels before reduction in a batch, q to be the number of the remaining voxels after reduction. Then we define the *reduction ratio* as $\eta = 1 - \frac{q}{p}$. Now we increase the batch size to be 2 times of the original and do a reduction on this new batch. The reduction ratio becomes $\eta' = 1 - \frac{q \oplus q}{p+p}$, where $\oplus$ represents a reduction sum. Since all the voxels reside

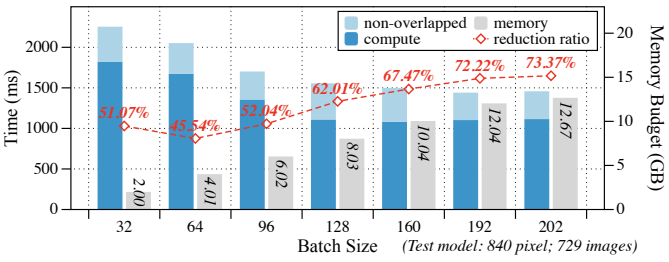

Figure 3: The influence of different batch size on reduction ratio, memory consumption and execution time breakdown.

on a slice plane which through the centre of the model, there are always redundant voxels generated by two images with arbitrary orientation. Thus, $q \oplus q < q + q = 2q$, which leads to

$$\eta^{'} = 1 - \frac{q \oplus q}{p + p} > 1 - \frac{2q}{2p} = 1 - \frac{q}{p} = \eta$$

. This demonstrates that the large batch size contributes to high reduction ratio in general, when orientations of images follow a uniform distribution.

However, the large batch has impact on CPU, since it's more sensitive to the increment of the task than GPU. So large batch has potential negative effect on dragging the whole process. Moreover, large batch also consumes more memory. Therefore, finding the best batch size for different models is a trade-off, as illustrated in Figure 3.

5 STREAM SCHEDULING

The streaming model abstraction has transformed the 3D reconstruction process into a pipelined process which comprises of four stages, *streaming*, *sorting*, *reduction* and *accumulation*, as illustrated in Figure 2. Based on this decomposition, Stream3D employs a top-down approach to design a three level (high-level, mid-level and low-level) scheduling controls, shown in Figure 4. Each level concentrates a distinct concurrency aspect of CPU-GPU system.

5.1 Coordination between CPU and GPU

The high-level control strategy focuses on stage distribution and cooperation between CPU and GPU. Based on the computational features and resource requirements, the four stages are distributed on different architectures. The streaming, sorting and reduction, which feature computation-intensive calculation and possess a high degree of data parallelism, are assigned to GPUs. On the other hand, the memory-intensive accumulation stage, which is limited by the entire model storage requirement, is assigned to CPU.

The scheduling of the coordination between host and device revolves around a basic objective - to reduce the idle time of device and sustain a high and stable compute throughput. The mechanism is facilitated by the following components:

- **Dedicated Thread** Two kinds of worker threads, updater and accumulator, run in background serving for the image batch buffer update and model accumulate request.
- **Blocking Task Queue** Update and accumulate task descriptor are encapsulated into task objects, which are scheduled through a blocking First-In-First-Out (FIFO) queue, achieving coordination between host threads and device CUDA streams.
- **Buffer Pool** A buffer memory maintained on host to cope with occasional request burst.

The dedicated threads combined with the blocking queue compose the asynchronous task scheduling mechanism, where the requests from devices are guaranteed to be scheduled immediately so as to not delay any following operations on devices. To ensure the completion of the requested task, an explicit synchronization is available to be invoked. By identifying and leveraging the opportunity for provoking the appending task in advance, immediately after the data dependency is dismissed, our scheduling design exploits

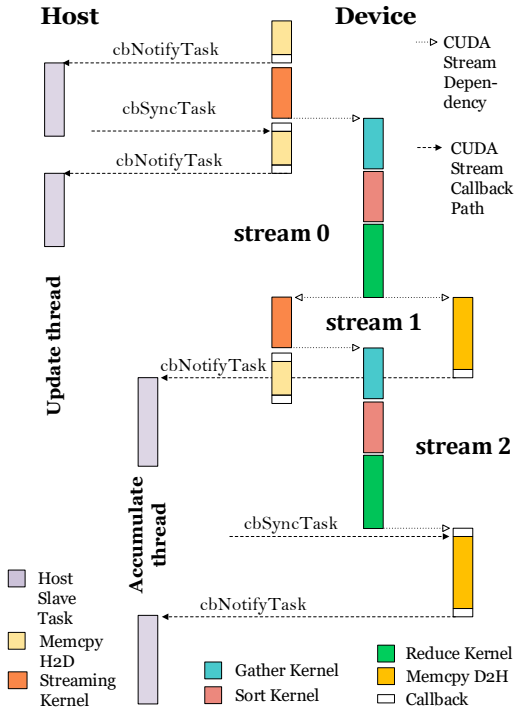

Figure 4: Three-level scheduling controls of Stream3D. The scheduling between host and device, among three device streams and inside a single stream corresponds to high-level, mid-level and low-level, respectively.

sufficient parallelism both of the pipeline and the heterogeneous architecture. To handle any request bursts which may result from the dynamic nature of the reduction stage, a page-locked memory buffer pool is deployed. This contributes to a stable sustained compute throughput.

5.2 Scheduling Multi-Streams on GPU

The concentration of the mid-level control is how to efficiently schedule multi-CUDA-streams on devices. Two strategies for multi-stream design are examined. One is employing *identical style*, each stream is enqueued with the identical operations but with different parts of the input data; while an alternative is to enqueue each stream with different operations and completes the whole calculation in all together, leading to a *cooperative style* multi-stream design.

The two are both capable of implementing multi-stream concurrency, but have different impact on the reduction ratio which is critical to the efficiency of data transferring. Based on the batched processing method, the identical style which holds several identical buffers actually exacerbates the fragmentation of the calculation, being contradictory to the batched idea. In view of this, the cooperative style outperforms the identical style, because it is capable of compressing the full batch data while the latter can only do

compression on a subset. The sub-batch compression not only directly decreases the absolute reduction amount but also decays the reduction ratio, as shown in Section 4.3.

5.3 Intra-Stream Scheduling

The low-level control is proposed to schedule operations in single stream, where major challenge presented here is the dynamic nature inherited from the reduction operation. To achieve the computation-communication overlapping, the reduction and data copy operations should be invoked asynchronously. But the non-deterministic length of the voxel stream remaining after reduction controls the up next offloading operation, which means a synchronization is required before calling the memory copy operation to get the actual data size after reduction in runtime. This kind of synchronization has a negative impact on maximizing the concurrency as it delays the execution of the following operations, which further impedes the operations overlapping.

We adopt a pre-issue scheduling strategy by carefully arranging the order of the operations in a stream to achieve the minimum delay on stream's asynchronous execution. The basic idea is to issue the operations on the next batch as early as possible. Considering that the reduction occupies the computation unit of GPU, we pre-issue the next batch's memory copy operations to increase overlapping.

5.4 Hybrid Communication Design

Stream3D presents a hybrid communication design. Each GPU within a node communicates by reading/writing model partitions from/to host memory, which contributes to intra-node model parallelism. Each node communicates the entire 3D model after each iteration, which contributes to inter-node data parallelism. Such a hybrid design comes from the trend that large models are usually reconstructed with big data. That is, each node is responsible for processing a large number of images during the 3D reconstruction. With regard to intra-node parallelism, a batch of images only update a small portion of the whole model, so partitioned model updates can greatly reduce the intra-node communication. However, for the inter-node case, the number of images on each node is large to update the whole model. A model parallel design can not reduce the amount of communication, but increase the frequency of communication. Therefore, when scaling to multi-nodes, we choose data parallel strategy that each node acts as a model server. The input images are evenly distributed among the nodes, and model is updated at each iteration through *allreduced* operation. The communication pattern of both intra-node and inter-node has been taken into consideration in such a hybrid design.

6 IMPLEMENTATION TECHNIQUES

6.1 Scheduling Mechanism

The scheduling of CPU-GPU cooperative parallel scheme is implemented through blocking task queues and CUDA stream callback mechanism. The blocking queue on host provides two operations: the `push` for enqueuing a task into the FIFO list and the `pop` for dequeuing a task from the FIFO list. The two operations are blocking in that if a `pop` operation is invoked on an empty queue, it will be blocked until a new task is pushed into the queue.

The `push` and the `pop` operations are then encapsulated into `notify` operation and `synchronize` operation respectively, which are invoked by CUDA stream callback functions. The `notify` operation is designed to be properly inserted into CUDA stream to ensure the task to be scheduled immediately after the data dependency is dismissed at runtime. And the `synchronize` operation is inserted into stream just before the data buffers are reused to guarantee that the last operation on this buffer is completed.

The worker threads on host undertake the actual tasks, such as updating the host buffer and accumulating the voxel stream to the final model, by scanning the appending queue in background. Whenever there is a new task in the queue, the worker will take this job out of it and add to the accomplished queue after completion.

6.2 Optimization Techniques

Reduce-by-key. The internal reduction process is a sort based reduce-by-key implementation. To make the calculations more memory friendly and efficient, the key-value pairs data stream is organized as structure of arrays with keys and values stored continuously. The actual reductions perform in two stages. The first is key reduction stage after which the boundary information will be stored. Then goes to the value reduction stage, where each part of the array structure is reduced based on the boundary info. We leverage the well optimized implementations of sort and segment-reduce kernel in THRUST [8] and MGPU [3] libraries.

Cached memory allocator. The memory allocation and deallocation on GPUs are very expensive operations, so an effective device memory management is required to obtain the optimum overall performance. In our implementation, a cached memory allocator is implemented to minimize the impact of allocations by reusing and sub-allocating GPU global memory.

NUMA-aware data placement. In order to improve the bandwidth of the memory intensive accumulation operations on CPU, NUMA-aware data placement, as well as vectorization optimizations, are leveraged.

7 EXPERIMENTS

In this section, we evaluate Stream3D on multiple datasets and get the following findings: 1) Stream3D outperforms baseline GPU implementation through key-value based streaming transformation. 2) Stream3D makes it possible to run large models on GPUs and the efficiency is guaranteed. 3) Stream3D enables a hybrid communication, which achieves 65.32-fold speedup on 20 nodes (80 GPUs).

7.1 Platform and Test Cases

System setup: Experiments are performed on two GPU platforms - System A and Cluster B. System A is a two node cluster, each of its node is equipped with two 14 cores (28 threads) Intel Xeon E5-2690 v4 CPUs and 4 Nvidia Tesla P100 GPUs. The memory of a node is configured with a 256GB DRAM and each P100 GPU has a 16GB on device memory. Cluster B has 20 CPU-GPU heterogeneous nodes. Each node comprises two 12 cores (24 threads) Intel Xeon E5-2643 v4 CPUs with 256GB DRAM and 4 Nvidia Tesla K40m GPUs, each of them has a device memory of 12GB.

Table 2: Single GPU Results with Different Model Size

Model Size	Batch Size	Reduc. Ratio	Entire Mem.	Stream Mem.	Rlt. Speedup GPU	CPU
200	320	88.52%	0.09	1.06	2.36	12.66
360	320	85.31%	0.52	3.40	2.26	11.74
520	300	82.78%	1.58	6.62	2.12	12.31
680	260	79.35%	3.52	9.80	1.91	11.41
840	196	72.72%	6.64	11.25	1.69	12.13
1000	156	66.01%	11.20	12.69	1.71	10.98
1060	112	56.22%	13.34	11.18	–	9.29
1220	96	50.25%	20.33	12.68	–	7.02
1380	38	47.51%	29.40	6.42	–	6.25
1540	48	44.21%	40.87	10.10	–	6.93
1700	32	49.31%	54.97	8.20	–	6.33
1860	32	49.25%	71.99	9.82	–	6.96

"GPU" represent the baseline implementation; "-" means model size exhausted device memory or are not supported; memory is measured in GB.

Test cases: The dataset used in our evaluation comprises 12 generated cases and 1 real-world case, see in section 7.5. The largest model size is set to 1860 pixel as the upper bound of the images that current electronic microscope can obtain is around 2000 pixel. Both the CPU baseline and GPU baseline are adapted from [9], the only one published software with full-fledged CPU/GPU 3D reconstruction implementation to the best of our knowledge. The evaluation of single GPU performance and intra-node scalability are conducted on System A. The inter-node scalability are evaluated on Cluster B. The baseline performances of the CPU-only implementations for single node and multi-nodes are tested on Cluster B.

7.2 Performance Results on Single GPU

Table 2 reports the single GPU performance of Stream3D. The results show that Stream3D outperforms the CPU implementation by an average relative speedup of 11.87, 7.13 and 9.50 on model-fitted cases, out-of-memory cases and total cases, respectively. For model-fitted cases, Stream3D also achieves 1.69 to 2.36 times speedup than the straightforward GPU implementation. Note that with the increment of the model size, there is a slight performance decay on Stream3D. This is due to the reduced batch size to fit the large model and decreased reduction ratio. Moreover, the large model will need a wider datatype to store the index, this also contributes to the performance loss observed between model size of 1000 and 1060. On the other hand, Stream3D greatly reduces the memory consumption. Specifically, it saves up to 7.33x device memory for the biggest model in our test case. While it still 6.96x faster than the CPU implementation.

We evaluate the computational efficiency of Stream3D by profiling the kernel performance and compare with the GPU baseline, as shown in Table-3. The results show that Stream3D achieves much higher global bandwidth at 178.04 GB/s, up to 15 times of the baseline. Benefiting from shared memory, the local memory access overhead is greatly reduced as more local variables can be accessed at a fast speed. This is further revealed in high warp execution efficiency, which represents the average percentage of active threads

Table 3: Kernel Performance Comparison

Mem.	Shared mem. usage (KB)	Global mem. bandwidth	Local mem. overhead
mdl-fit	0 / 48	11.77 GB/s	59.20%
stream	38.81 / 48	178.04 GB/s	10.50%
Comp.	**Instruction divergence**	**Achieved active warp occupancy**	**Warp exe. efficiency**
mdl-fit	48.19%	24.9%	53.0%
stream	33.10%	75%	96.8%

The metrics are collected by nvprof through 560/1000/1540 test cases on K40m with number averaged. "mdl-fit" refers to baseline implementation.

in each executed warp. Because the reduced memory access latency, the overall calculation throughput increases.

7.3 Benefit of Scheduling Strategies

We measure the individual benefit of pre-issue and multi-reduction for cooperative multi-stream implementation, and compare with the identical multi-stream one. To achieve this, we disable each technique and measure the impact on the overlapped percentage of the total time. Figure 5-A gives the results. We observe that for small-sized and medium-sized model, multi-reduction brings more significant improvement than pre-issue, while the large model benefits more from pre-issue than multi-reduction. This is because the large model adopts small batch size, which leaves less space to be reduced by reduction. But for the larger batch size used by small and medium models, it will impose more pressure for CPU and have negative impact on the overlapping, if not leverage multi-reduction. The results further confirm that cooperative multi-stream design outperforms the identical one. The effectiveness of buffer pool design is evaluated by measuring the longest waiting time on `synchronize` call. This metric reflects the stall time on CPU side in the pipeline. Figure 5-B gives the results normalized by one buffer configuration in each case. It shows that using three buffers, the longest waiting time can be reduced lower than 3% of computation time for all the three cases.

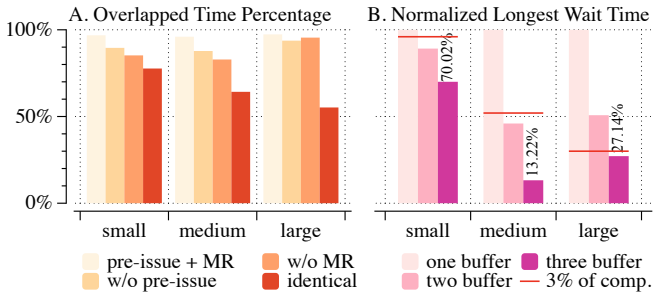

Figure 5: Benefit of scheduling optimizations in Stream3D. *A.* Stream scheduling strategies comparison on one device. (MR: multi-reduction) *B.* Effects of buffer pool size, using 4 devices. All tests are performed on System A.

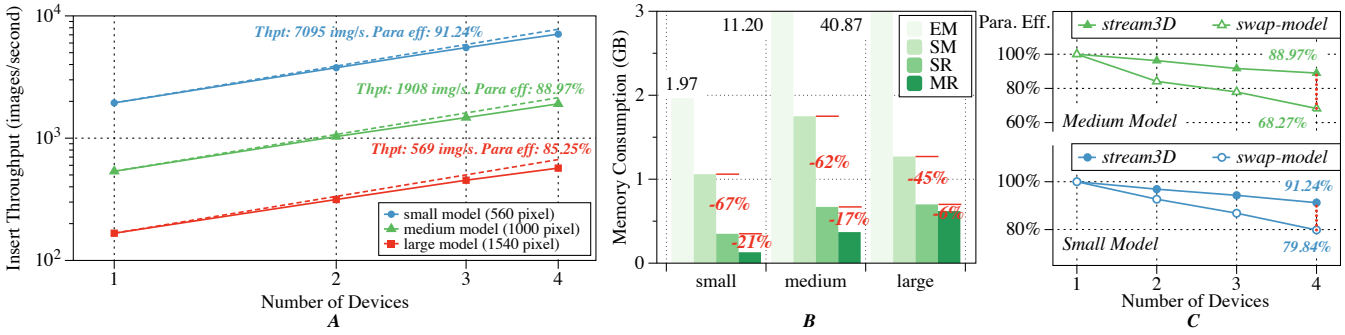

Figure 6: Intra-node weak scalability of Stream3D and analysis. *A.* Weak scaling result on a System A node. Each device is assigned with 58,320 images. *B.* Memory and communication reduction. (EM: entire memory; SM: streaming memory; SR: single reduction; MR: multi-reduction. SM, SR and MR is the average number measured of one batch.) *C.* Parallel efficiency comparison between Stream3D and swap-model implementation which can only handle small and medium-sized models.

7.4 Scalability

Intra-node. Figure 6-A reports the intra-node weak scalability of Stream3D. We perform the test by incrementally adding GPUs to a single node and keeping each GPU with fixed workload. It shows that, Stream3D can sustain a throughput of 569 images/s for the large model with four GPUs configured, while keep a parallel efficiency at 85.25%. The (multi-)reduction mechanism contributes a lot to this scalability, as shown in Figure 6-B, 50%∼80% of the redundant communication is avoided when single (and multi-)reduction was enabled on streaming design. Compared with the entire model consumption, the result in Figure 6-B further reveals the Stream3D's ability of eliminating memory capacity bottleneck. Moreover, for cases where models can be fitted in device memory, Stream3D also improves the intra-node scalability than swap-model implementation, as shown in Figure 6-C.

Inter-node. We examine the strong scalability of Stream3D when scaling to multi-nodes. The curves indicate that Stream3D retains a reasonable parallel efficiency (above ∼70% for four device configurations) when scaling up to 20 nodes (80 GPUs). It demonstrates that Stream3D is an effective solution to handle large model reconstruction problem.

7.5 A Real-World Case

We tested Stream3D on a real-world case for reconstructing proteasome [4] – a highly sophisticated and important protein complex. Results are reported in Figure 8, 9. Compared with the baseline (of which the 3D reconstruction is CPU code, other parts are GPU-based), Stream3D reduces the total structure determination time from 7h 12m to 2h 32m, speeds up the reconstruction process and the whole process by 7.58 and 2.83 times, respectively.

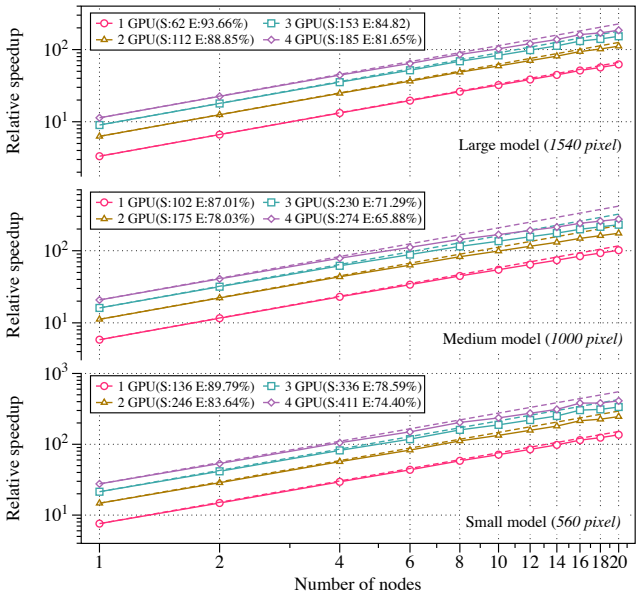

Figure 7: Inter-node strong scaling results of Stream3D on Cluster B. Tests are performed with three different model size and different device configuration (1,2,3,4 GPU(s) per node). The testset is of 583,200 images. The *S* and *E* represent relative speedup and parallel efficiency achieved on 20 GPU nodes with 80 GPUs, respectively.

8 RELATED WORK

3D Reconstruction. Due to the huge computational challenges, 3D reconstruction problem has earned much research effort, such as the 3D image reconstruction in Computed Tomography (CT) [22, 29]. Compared with CT the stacked 3D reconstruction which is decomposed into a series of 2D reconstructions from 1D projections with fixed orientation, direct Fourier Transform based reconstruction in cryo-EM is a *true* 3D reconstruction problem, which presents tougher challenges due to the non-deterministic orientations and huge memory requirements. Relion-2 [11] proposed two memory access patterns to implement 3D reconstruction for its CPU code, named Gather and Scatter. However, it is a pure complexity analysis only takes the total number of memory access into consideration, leaving out the discontinuity and irregularity. To mitigate the data

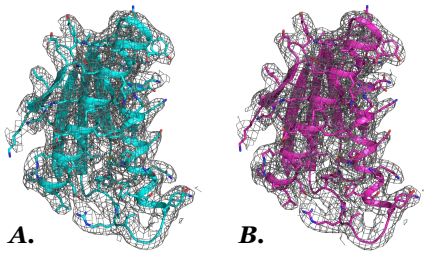

Figure 8: Result of the real-world case. Proteasome Chain K, a part of the whole structure of high resolution. *A, B* are produced by the CPU baseline and Stream3D respectively. Stream3D achieves the same resolution with baseline, comparison is performed in double precision. Test is conducted on Cluster B. (Dataset: EMPIAR-10025, PDB: 6BDF)

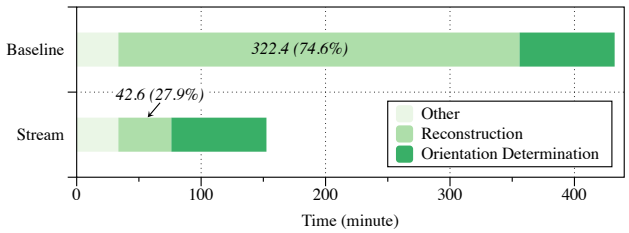

Figure 9: Performance results of the real-world case. The reconstruction process occupies 74.6% of baseline total time. Stream3D reduces the reconstruction time from 5h 22m to 42m and reduces the ration to 27.9%.

race condition intrinsic in 3D reconstruction, eLife3D [35] constructed an interleaved scheme which made use of the linearity nature of the coordinate transform by controlling the size of the pixel group among different threads to reduce data dependency. Although this method worked well for removing speedup hindrance resulting from write-collision, it was not designed for multi-GPU scaling and solving the memory capacity problem of large model. By comparison, our design exploits the parallel opportunity at pixel/voxel level, which is a more fine-grained parallelism than pixel-group level design. Focusing on increasing utilization of memory bandwidth and the floating-point performance, [26] proposed a percolation technique to optimize off-chip memory performance. The philosophy behind the percolation idea falls in the same way with our key-value abstraction in that both try to improve the calculation by decoupling computation with memory operations. However, that work is not proposed for 3D reconstruction process.

GPU Computing. Systematic strategies to the challenge of *insufficient GPU memory size* and *irregularity in GPU computing* have been the concentrations of many researches. To *eliminate the memory bottleneck*, [10, 36] focus on solutions relying on hardware extensions. Since they normally manipulate memory at page granularity and take programmability as a primary goal, the performance is less attractive for our case. Other software solutions [21, 28] leverage domain specific knowledge, such as deep learning (DL) filed where the large model is also a great concern. But neural networks in DL are layered models with static structures, which makes it natural to partition the model into layers fitted in GPU memory. However, finding an effective model partition method is non-trivial in cryo-EM as we discussed in section 2.2. For *irregular memory access*, [14, 30, 33, 34] made elaborate explorations on two types of the problem, the static irregularity and the dynamic irregularity, based on whether the access pattern can be determined at compilation time or run time. Data reposition and duplication [30], as well as thread reorganization [34] were proposed for irregularity removal either offline or on-the-fly. However, finding the optimum data layout in-place was proven to be NP-complete [30]. In practice, a relaxation of the space constraint, i.e. using extra memory, is leveraged to reduce the complexity. Unfortunately, the 3D reconstruction problem manifests itself as a constrained problem with extremely stringent memory budget as discussed above. Therefore, the nondeterministic image orientations make the static methods less attractive, while the limited memory capacity makes it can hardly directly benefit from the dynamic whole model transformation methods either in-place (e.g. using space-filling curve [31]) or out-of-place (e.g. through sort-padding-sharing proposed by [30]).

9 GENERALIZATION

While this work focuses on the GPU acceleration of Cryo-EM 3D reconstruction, the element-centric view and techniques proposed in this paper are also applicable to other domains, where the problem space exhibits extreme sparsity and nondeterminicacy. For example, machine learning applications, such as graph embedding [20] and word embedding [13], which exhibit sparse model update and suffer from huge models. Our key-value streamed model partition strategy can also benefit.

10 CONCLUSION

To enable compute-intensive high-resolution 3D cryo-EM reconstruction to benefit from powerful GPUs, we propose Stream3D, a novel parallel design with a key-value stream abstraction on GPU clusters. Benefiting from this abstraction, we transform the challenging problem into a more memory-friendly and compute-efficient calculation, improving the performance of the reconstruction part by an average of 9.50 times and the performance of the entire processing part by 2.83 times. Moreover, we design a hybrid communication mechanism to enable Stream3D at a large scale, achieving 65.32-fold speedup when using up to 80 GPUs.

ACKNOWLEDGEMENT

We thank all the anonymous reviewers for their detailed reviews and valuable feedback. We also thank Zhao Wang, Mingxu Hu and Xueming Li for their comments. This work is supported in part by National Key R&D Program of China (Grant No. 2016YFA0602200), National Natural Science Foundation of China (61672312 to G.Y.), Advanced Innovation Center for Structural Biology (to X.L., Y.S., and G.Y.) and China Postdoctoral Science Foundation (Grant No. 2018M641359). Author (H.F.) was also supported by Center for High Performance Computing and System Simulation, Pilot National Laboratory for Marine Science and Technology (Qingdao).

REFERENCES

[1] V. Abrishami, J.R. Bilbao-Castro, J. Vargas, R. Marabini, J.M. Carazo, and C.O.S. Sorzano. 2015. A fast iterative convolution weighting approach for gridding-based direct Fourier three-dimensional reconstruction with correction for the contrast transfer function. *Ultramicroscopy* 157 (2015), 79–87. https://doi.org/10.1016/j.ultramic.2015.05.018

[2] Soojay Banerjee, Alberto Bartesaghi, Alan Merk, Prashant Rao, Stacie L Bulfer, Yongzhao Yan, Neal Green, Barbara Mroczkowski, R Jeffrey Neitz, Peter Wipf, et al. 2016. 2.3 Å resolution cryo-EM structure of human p97 and mechanism of allosteric inhibition. *Science* 351, 6275 (2016), 871–875.

[3] Sean Baxter. 2016. moderngpu 2.0: Patterns and behaviors for GPU computing. http://moderngpu.github.io/moderngpu

[4] Melody G Campbell, David Veesler, Anchi Cheng, Clinton S Potter, and Bridget Carragher. 2015. 2.8 Å resolution reconstruction of the Thermoplasma acidophilum 20S proteasome using cryo-electron microscopy. *Elife* 4 (2015), e06380.

[5] Y Cheng, N Grigorieff, A Penczek, P, and et al. 2015. A primer to single-particle cryo-electron microscopy. *Cell* 161, 3 (2015), 438–449.

[6] J.J. FernÃąndez. 2008. High performance computing in structural determination by electron cryomicroscopy. *Journal of Structural Biology* 164, 1 (2008), 1–6. https://doi.org/10.1016/j.jsb.2008.07.005

[7] N Grigorieff. 2007. FREALIGN: high-resolution refinement of single particle structures. *Journal of Structural Biology* 157 (2007), 117–125.

[8] Jared Hoberock and Nathan Bell. 2015. Thrust: Parallel Algorithms Library. https://thrust.github.io

[9] Mingxu Hu, Hongkun Yu, Kai Gu, Zhao Wang, Huabin Ruan, Kunpeng Wang, Siyuan Ren, Bing Li, Lin Gan, Shizhen Xu, et al. 2018. A particle-filter framework for robust cryo-EM 3D reconstruction. *Nature methods* 15, 12 (2018), 1083.

[10] Y. Kim, J. Lee, J. Jo, and J. Kim. 2014. GPUdmm: A high-performance and memory-oblivious GPU architecture using dynamic memory management. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 546–557. https://doi.org/10.1109/HPCA.2014.6835963

[11] D. Kimanius, B.O. Forsberg, S.H.W. Scheres, and E. Lindahl. 2016. Accelerated cryo-EM structure determination with parallelisation using GPUs in RELION-2. *eLife* 5, NOVEMBER2016 (2016). https://doi.org/10.7554/eLife.18722.001

[12] W Kuhlbrandt. 2014. The Resolution Revolution. *Science* 343, 6178 (2014), 1443–1444.

[13] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.

[14] Seyong Lee, Seung-Jai Min, and Rudolf Eigenmann. 2009. OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '09)*. ACM, New York, NY, USA, 101–110. https://doi.org/10.1145/1504176.1504194

[15] L. Li, X. Li, G. Tan, M. Chen, and P. Zhang. 2011. Experience of parallelizing cryo-EM 3D reconstruction on a CPU-GPU heterogeneous system. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing* (2011), 195–204. https://doi.org/10.1145/1996130.1996157

[16] E. Nogales. 2015. The development of cryo-EM into a mainstream structural biology technique. *Nature Methods* 13, 1 (2015), 24–27. https://doi.org/10.1038/nmeth.3694

[17] Eva Nogales. 2016. The development of cryo-EM into a mainstream structural biology technique. *Nature methods* 13, 1 (2016), 24–27.

[18] N. Oliveira, A.M. Mota, N. Matela, L. Janeiro, and P. Almeida. 2016. Dynamic relaxation in algebraic reconstruction technique (ART) for breast tomosynthesis imaging. *Computer Methods and Programs in Biomedicine* 132 (2016), 189–196. https://doi.org/10.1016/j.cmpb.2016.05.001

[19] P. A. Penczek. 2010. Fundamentals of three-dimensional reconstruction from projections. *Methods in Enzymology* 482, 482 (2010), 1–33.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[21] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. 2016. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–13.

[22] 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. In *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '17)*. ACM, New York, NY, USA, 207–220. https://doi.org/10.1145/3018743.3018765

[23] S.H.W. Scheres, M. Valle, and J.-M. Carazo. 2005. Fast maximum-likelihood refinement of electron microscopy images. *Bioinformatics* 21, SUPPL. 2 (2005), ii243–ii244. https://doi.org/10.1093/bioinformatics/bti1140

[24] Sjors HW Scheres. 2012. RELION: implementation of a Bayesian approach to cryo-EM structure determination. *Journal of structural biology* 180, 3 (2012), 519–530.

[25] Devika Sirohi, Zhenguo Chen, Lei Sun, Thomas Klose, Theodore C Pierson, Michael G Rossmann, and Richard J Kuhn. 2016. The 3.8 Å resolution cryo-EM structure of Zika virus. *Science* 352, 6284 (2016), 467–470.

[26] G. Tan, Z. Guo, M. Chen, and D. Meng. 2009. Single-particle 3D reconstruction from cryo-electron microscopy images on GPU. *Proceedings of the International Conference on Supercomputing* (2009), 380–389. https://doi.org/10.1145/1542275.1542329

[27] A. Voropaev, A. Myagotin, L. Helfen, and T. Baumbach. 2016. Direct Fourier Inversion Reconstruction Algorithm for Computed Laminography. *IEEE Transactions on Image Processing* 25, 5 (2016), 2368–2378. https://doi.org/10.1109/TIP.2016.2546547

[28] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. 2018. Superneurons: Dynamic GPU Memory Management for Training Deep Neural Networks. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '18)*. ACM, New York, NY, USA, 41–53. https://doi.org/10.1145/3178487.3178491

[29] Xiao Wang, Amit Sabne, Sherman Kisner, Anand Raghunathan, Charles Bouman, and Samuel Midkiff. 2016. High Performance Model Based Image Reconstruction. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '16)*. ACM, New York, NY, USA, Article 2, 12 pages. https://doi.org/10.1145/2851141.2851163

[30] Bo Wu, Zhijia Zhao, Eddy Zheng Zhang, Yunlian Jiang, and Xipeng Shen. 2013. Complexity Analysis and Algorithm Design for Reorganizing Data to Minimize Non-coalesced Memory Accesses on GPU. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '13)*. ACM, New York, NY, USA, 57–68. https://doi.org/10.1145/2442516.2442523

[31] Pan Xu and Srikanta Tirthapura. 2012. On the Optimality of Clustering Properties of Space Filling Curves. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '12)*. ACM, New York, NY, USA, 215–224. https://doi.org/10.1145/2213556.2213587

[32] Chuangye Yan, Jing Hang, Ruixue Wan, Min Huang, Catherine CL Wong, and Yigong Shi. 2015. Structure of a yeast spliceosome at 3.6-angstrom resolution. *Science* 349, 6253 (2015), 1182–1191.

[33] Yi Yang, Ping Xiang, Jingfei Kong, and Huiyang Zhou. 2010. A GPGPU Compiler for Memory Optimization and Parallelism Management. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '10)*. ACM, New York, NY, USA, 86–97. https://doi.org/10.1145/1806596.1806606

[34] Eddy Z. Zhang, Yunlian Jiang, Ziyu Guo, Kai Tian, and Xipeng Shen. 2011. On-the-fly Elimination of Dynamic Irregularities for GPU Computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVI)*. ACM, New York, NY, USA, 369–380. https://doi.org/10.1145/1950365.1950408

[35] X. Zhang, X. Zhang, and Z. Hong Zhou. 2010. Low cost, high performance GPU computing solution for atomic resolution cryoEM single-particle reconstruction. *Journal of Structural Biology* 172, 3 (2010), 400–406. https://doi.org/10.1016/j.jsb.2010.05.006

[36] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler. 2016. Towards high performance paged memory for GPUs. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 345–357. https://doi.org/10.1109/HPCA.2016.7446077

[37] Y. Zheng and P. C. Doerschuk. 2008. A parallel software toolkit for statistical 3-D virus reconstructions from cryo electron microscopy images using computer clusters with multi-core shared-memory nodes. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–11. https://doi.org/10.1109/IPDPS.2008.4536242