Improve the resolution and parallel performance of the three-dimensional refine algorithm in RELION using CUDA and MPI

Jingrong Zhang, Zihao Wang, Zhiyong Liu, Fa Zhang*

Abstract—In cryo-electron microscopy, RELION is a powerful tool for high-resolution reconstruction. Due to the complicated imaging procedure and the heterogeneity of particles, some of the selected particle images offer more disturbing information than others. However, in the current RELION, all these particle images are treated equally. In our work, we extend RELION's model with one scalar parameter to score the contribution of a particle depending on the error between the experimental particle and the corresponding reprojection. This scores down weight potentially poor particles, hence accelerating the convergence. Besides, by now there is no sophisticated memory management system for RELION, fragmentation on GPU will increase with iterations, eventually crashing the program. In our work, we designed the stack-based memory management system to guarantee the stability of RELION and to optimize the memory usage condition. Also, to reduce memory usage, we developed a customized compressed data structure for the memory-demanding weight array. In addition, to speed up the GPU version of RELION, we proposed two highly efficient parallel algorithms for weight calculation algorithm and weight selection algorithm. Experiments show that compared with RELION, the optimized three-dimensional refine algorithm can speed up the converge procedure, the memory system can avoid memory fragmentation, and a better speed-up ratio can be obtained.

Index Terms—cryoEM, RELION, CUDA, Statistical Method, Top-k Selection.

1 Introduction

CRYO-ELECTRON microscopy (cryoEM) has become a mainstream technology for studying the architecture of biological macromolecules [1]. Single-particle analysis (SPA) is an important sub-discipline of cryoEM [2]. Here, "single particles" refers to structurally identical macromolecular complexes (like proteins and viruses). By acquiring thousands and even millions of two-dimensional (2D) projections of randomly oriented particles, SPA uses reconstruction software (like EMAN [3] and RELION [4]) to recover the three-dimensional (3D) structure [2]. Among reconstruction software packages, RELION significantly increases the attainable resolution by adopting a Bayesian approach [4], [5]. Many structures have achieved near-atomic resolution using RELION [1], [6]–[11].

However, the imaging procedure in cryoEM is quite complicated, with abundant randomness. In general, the noise, artifacts of particles in the same micrograph may differ greatly. The general quality of even selected particles will differ [12]. Particles with severe artifacts and noise ("bad particles") may negatively influence the reconstruction procedure, a problem that has not received enough attention. In our work, inspired by the training procedure of neural networks, we extended the model parameters of RELION with one scalar parameter and updated it based on the idea of back-propagation. In our method, the scalar of "bad particles" will decrease with iterations, while that of "good particles" will increase. This modification speeds up the converge procedure and boosts the final resolution.

The expectation–maximization (EM) algorithm is an iterative method to find the best hypothesis for the probability distribution of hidden parameters in statistical models. However, to estimate the probability distribution of the orientation parameters for each particle (in the expectation step), RELION needs to compare each particle image with every calculated projection [13], leading to a huge computation load [14] and large memory requirement. Taking the dataset used in the tutorial of RELION 2.0 [15] containing 6,496 β-galactosidase 128*128 particles as an example, the 3D refinement stage requires more than 2GB memory per process and 40 CPU hours even though this dataset is quite small.

In addition, RELION adopts a "modified version of the expectation-maximization algorithm". In its "Expectation step", RELION performs a two times finer procedure. At the end of the first pass, RELION selects a sub-domain of all orientations for each particle image. The selection rule reserves the orientations with high probability whose sum reaches 99.9% of the sum total of the probability value of all orientations. One naive solution for this selection step is to first sort all probability values and then sum them sequentially from largest to smaller until their sum reaches 99.9% of the sum total. It is obvious that this method is a sequential algorithm and has high data dependency. The processing times of sorting and summation will increase dramatically as the number of orientations increases. To reduce the time cost of this step, we drew inspiration from the parallel top-k selection and stream compaction algorithms and designed

- *Jingrong Zhang and Zihao Wang are with University of Chinese Academy of Sciences Beijing, China.*
- *Jingrong Zhang, Zihao Wang, Zhiyong Liu and Fa zhang are with High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
  E-mail: [zhangjingrong,zihaowang,zyliu,zhangfa]@ict.ac.cn

an efficient parallel top selection algorithm.

To reduce the running time, many cryoEM software programs [16]–[20] have been parallelized on GPUs. The reviews [21]–[23] have introduced the advances of the parallelizaiton softwares. For example, CryoSparc utilizes the GPU card to speed up the reconstruction and 3D classification. And it A GPU parallel version of RELION has also been proposed [24]. Also, the software THUNDER [19] accelerates the image processing and calculation of reprojections with GPUs. However, most parallel software in cryoEM overlooks memory-related operational stability and robustness, which heavily relies on GPU hardware characteristics and the memory accessing pattern. For memory-intensive applications like RELION, memory-related code will dramatically influence its stability, robustness and performance. One important aspect is memory fragments. Frequently randomly allocating and de-allocating memory on the GPU may divide the successive memory into several pieces of discontinuous memory. We call those small pieces of memory “fragments”. When the program tries to allocate a large piece of memory, even though the total available memory is sufficient, the allocation can still fall and report an “out of memory” error because of the fragments. However, no sophisticated memory management system can prevent the fragmentation problem. In addition, no stable memory management strategy has been introduced to RELION. For example, when processing the dataset EMPIAR-10028 with a particle size of 360*360 on GPU NVIDIA K20c, an “out of memory” error occurs during later iterations. To improve the stability and robustness of RELION, we developed a stabilizing memory management system, especially for large datasets. In this system, we utilize the characteristics of iterative methods to avoid fragmentation. This memory manage system can also be applied to other iterative methods.

Moreover, even with a fragment-free memory management system, when the total memory consumption of RELION exceeds the GPU memory, an “out of memory” error will still be reported. If we can reduce the memory consumption, this issue would be eased fundamentally. As we mentioned, the weight array is the data structure that demands the most memory. To reduce its memory consumption, we analyzed the data structure of the weight array and determined that we can reduce its memory requirements by carefully redesigning its structure.

In summary, in this work, to speed up the convergence procedure and boost the final resolution, we propose an extended 3D refine algorithm by extending the model parameters with one scalar parameter and updating it based on the idea of back-propagation. In addition, we designed a memory-efficient and memory-stabilizing management system to guarantee the robustness of our program and the efficiency of GPU memory usage. Then, to reduce the memory usage, we developed a novel data structure. Finally, we developed a weight calculation parallel algorithm and a top selection algorithm to speed up the calculation.

Our paper is organized as follows: in Section 2, we introduce related work; in Section 3, we show the extended 3D refine algorithm; in Section 4, we introduce our stabilizing management system and memory-efficient data structure; in Section 5, we present the weight calculation parallel algorithm and the top selection algorithm. The results of the experiments are presented in Section 6. We then conclude the paper.

2 Related Work

In this section, we introduce the related work from three aspects: statistical methods used in cryoEM, the basic workflow of the refine algorithm of RELION and efforts to accelerate it, and parallel top-k selection algorithms.

2.1 Statistical Methods

As mentioned, the structure determination method used in RELION is a Bayesian-based method and is therefore a statistical method. In general, the method of RELION is less sensitive to the starting model and permits recovery of the underlying signal from noisy data compared to traditional cross-correlation-based methods [25]–[27]. In general, statistical methods try to find the best hypothesis for the distribution of parameters in a statistical model given the observations. There have been numerous contributions beyond RELION applying this method based on different kinds of image formation models in cryoEM [25] [28] [29].

For example, in a previous work [28], the data model was $X_i = P_{i\phi}A_{ki} + \sigma G_i$ (where A_{ki} indicates the underlying image, $P_{i\phi}$ is the transformation with the ith image, σ is the standard deviation coefficient, and G_i indicates the independent noise). The model aims to find the most likely set of parameters (Θ) to construct a model describing structural data through optimization of the following log-likelihood function:

$$L(\Theta) = \sum_{i=1}^{I} ln \sum_{k=1}^{K} \int_{\phi} P(X_i|k,\phi,\Theta)P(k,\phi|\Theta)d\phi \qquad (2.1)$$

The reconstruction algorithm in RELION is also a typical statistical method. It is solved by a modified version of the expectation maximization algorithm based on the work of [13]. It formulates the image formation model in Fourier space as follows:

$$X_{ij} = CTF_{ij} \sum_{l=1}^{L} P_{jl}^{\phi} V_l + N_{ij} \qquad (2.2)$$

where X_i is the projection image obtained by experiments, $CTFi$ is its contrast transfer function for the ith image, and V is the underlying structure in the dataset. The operation $\sum_{l=1}^{L} P_{jl}^{\phi} V_l$ extracts a slice from V, which is equivalent to the real-space projection. In RELION, the parameter set Θ is V_l, σ, τ, where σ^2 and τ^2 are the variance of the noise and the signal component.

In this image formation model, the orientation parameters for projection images are unknown, and we use $\Gamma_{i\phi}^{(n)}$ to indicate the posterior probability of orientation assignment ϕ for the i-th image in the n-th iteration ($i \in [0, I]$). Here, the orientations $\phi \in [0, \Phi]$ ($\Phi = (rRot, rTilt, rPsi, tX, tY)$) comprise 3 rotation ($rRot, rTilt, rPsi$) and 2 translation ($tX, tY$) parameters.

2.2 Efforts to Accelerate RELION

GeRelion [30] is a parallel GPU version 1.4. To port the program to the GPU, GeRelion proposes a four-level schedule model. In addition, the weight array is assumed to be sparse and stored in compressed sparse row (CSR) matrix mode: one continuous vector stores the value, and an aux vector records the indexes of the weight array. Because of the usage of the aux vector, the memory consumption of this structure may exceed that of the original structure when the weight array is not sparse. Based on this consideration, we developed a new data structure that can reduce its memory cost.

RELION [24] also proposed a parallel GPU version for its expectation-maximization algorithm. In contrast to its CPU version, the parallel version runs in single precision mode by default to reduce memory consumption. In addition, during the expectation step, accessing the re-projection images consumes a lot of time. RELION stores re-projection images in texture memory to make use of its on-chip cache, which can reduce the memory accessing time. However, this strategy is not always effective. As the size of the re-projection image increases, it may exceed the size of the texture memory cache. In this case, there would be frequent data swapping between the cache and texture memory. Consequently, the performance of the cache will be heavily diminished. Based on this observation, we designed a different data accessing strategy to speed up the memory accessing.

Both GeRelion [30] and RELION [24] provided detailed analyses of the performance of each major step. According to their analyses, the expectation step is the most time-consuming step. The most memory-demanding data structure is the weight array Γ.

2.3 Parallel Top-k Selection Algorithms

In our work, we developed a parallel top selection algorithm for RELION based on the idea of parallel top-k selection algorithm. Here we introduce its current research condition. Top-k problem requests to find the top K elements from an un-ordered list [31]. One general kind of algorithm is selecting by Sorting. The basic idea of these algorithms is to sort the entire list and then query the top K elements. Although the top-k selection problem is a different from the top selection problem in RELION, the basic idea of current implementation of RELION is the same.

Another kind of algorithm is to select based on partitions. Many parallel selection algorithms are based on partitions. One advantage of such algorithms is that they can avoid the sorting operation. The basic steps of "Bucket Select" are as follows [32]. First, "Bucket Select" determines the size of buckets, assigns each element to a bucket and then identifies the bucket containing the k-th order statistic. And it projects the entries in that bucket onto a new set of buckets. After a number of iterations, when the bucket width equals 1, we find the k-th element.

Figure 1 shows an example of "Bucket Select". It uses the linear projection function Formula 2.3 to define the boundary of buckets.

$$bucket[i] = \lfloor \frac{B}{max - min}(vec[i] - min) \rfloor \tag{2.3}$$

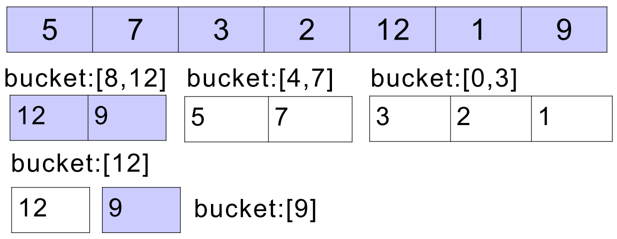


Fig. 1: Selecting the 2nd biggest element.

"Bucket Select" owns two phases. In Phase I, it reduces workload by creating a new vector containing the candidates. In Phase II, the workload is reduced by redefining the max and min.

"Bucket Select" can avoid the sorting operation. However, first, its efficiency reduces with the bucket width. Then this method can only find the kth element. There is still one step away from getting the top K elements.

So in our method, we first added an small dataset sorting to speedup the boundary converge. Then we implemented one more kernel to get the qualified data out based on the work [33].

3 Extend the Refine Algorithm to Estimate the Reliability of Particle Image

In general, we use statistical method to find the most likely perameter set Θ for the model constructed with the observation data and prior information. In work [28], they proposed to extent the model parameter set Θ with a multiplicative and an addictive factor for the signal in each experimental image to normalize the noise. In RELION 2.0, the variance of signal and noise has been involved in its statistical model. In Relion 3.0, per-particle CTF correcition and beam-tilt correction is introduced. These parameters can provide more detailed estimation for each particle so as to improve the reconstruction resolution. However, for each particle, the data can be used is so limited to achieve accurate statistical estimation. Besides, in the beginning iterations, the estimation about many particles' orientaions may be far from accurate. So the corresponding estimation about the signal, noise and CTF parameters are inaccurate. These parameters may be very heplful for high resolution refinement. But, when thay are inaccurate, they will slow down the convergence. Seeing these shortages, we try to estimate each particle in a more general way.

Due to the heterogeneity of particle, random noise and some other influence factors, the quality of particle images differs a lot. Some "bad" particles may introduce interference to the reconstruction result. If we can use a more generalized scaler parameter to estimate the realibility of each particle image and its related parameters, it can be used to restrain the influence from "bad particles" and enhance the contribution from "good" ones. And this scaler parameter would help accelarate the convergence of the expectation-maximization algorthm, especially for the begining iterations.

We desire to design this scaler parameter to reflect this realibilty of the particle images and their related parameters based on the understanding of the iteration procedure of expectation-maximazation algorithm. As we know, during the iterations, the estimation of relating parameter should be gradually getting close to the actual condition with the increase of the resolution. And the re-projections generated from the reconstruction result and these estimated parameters should be becomming more and more similar to the actual particle images. While as for the particles with low realibilty, this phenomenon doesn't hold. Because these particle didn't contribute positively to the reconstruction result and gradually the reconstruction may become more and more different to these particles. And these will shown in the error between re-projection and the particle image.

So based on this obervation, we desiged the concrete formula for this scalar parameter. For each particle, we store its best orientation and its corresponding error value $|X_i - CTF_i \sum_{l=1}^{L} P^{\phi'} V|^2$ and compare it with the re-projection. After one iteration, the error value of one particle will change. And if the error value gets smaller, it indicates that this particle made positive contributions and vice versa. By now we get the Formula 3.1:

$$\begin{aligned} V_l^{(n+1)} &= \frac{\sum_{i=1}^{N} \int_\phi \Gamma_{i\phi}^{(n)} \sum_{j=1}^{J} P^{\phi^T_{lj}} \frac{CTF_{ij} s_i X_{ij}}{\sigma_{ij}^{2(n)}}}{\sum_{i=1}^{N} \int_\phi \Gamma_{i\phi}^{(n)} \sum_{j=1}^{J} P^{\phi^T_{lj}} \frac{CTF_{ij}}{\sigma_{ij}^{2(n)}} + \frac{1}{\tau^{2(n)}}} \\ \sigma_{ij}^{2(n+1)} &= \frac{1}{2} \int_\phi \Gamma_{i\phi}(n) |X_{ij} - CTF_{ij} \sum_{l=1} LP_{jl}^{\phi} V_l^{(n)}|^2 d\phi \\ \tau_{ij}^{2(n+1)} &= \frac{1}{2} |V_l^{(n+1)}|^2 \\ s_i = \frac{3}{4} &+ \frac{e^{err_i^{(n)}}}{2*(e^{err_i^{(n)}}+1)} \end{aligned} \tag{3.1}$$

The second and third formulas are the same as RELION's formula written in work [4]. But, the first formula introduced the scalar parameter s_i to estimate the realibility of particle images. And the forth formula describes the concrete formula of this scalar parameter, within and $err_i^{(n)}$ is calculated by the following formula:

$$\begin{aligned} diff_i^n &= |\sum_{j=0}^{J} X_{ij} - CTF_{ij} \sum_{l=1} LP_{jl}^{\phi} V_l^{(n)}| \\ err_i^n &= \frac{diff_i^n - diff_i^{n+1}}{diff_i^{n+1}} \end{aligned} \tag{3.2}$$

Where $diff_i^n$ indicates the difference between the re-projection and particle image in the nth iteration. When $diff_i^n = diff_i^{n+1}$, s_i would equal to 1, which means the same as the original version. And $diff_i^n > diff_i^{n+1}$ indicates the particle has made positive contribution. The corresponding scalar parameter s_i is larger than 1. As scalar parameter s_i is a relaxation factor for each particle, to normalize the value of s_i, we used sigmoid function [34]. In our function, the range of s_i is [0.75, 1.25].

The general work flow of the modified algorithm is shown in Figure 2. In the expectation step (shown in the upper left corner of Figure 2), computer-generated projections (labeled as "re-projection") of the structure are compared with the experimental images (label as "real particle image") to acquire the probability distribution over orientations of the images.

In the maximization step, the 3D structure is updated combing the prior information (lower left corner of Figure 2). And according to the updated 3D structure V, we can update the variance of signal τ. Then, comparing the new re-projections and particle images we can update the variane of noise σ_{ij}. Also, comparing real-images with its re-porjection at best orientations of iteration n and $n+1$, we can update the scaler parameter s_i.

4 Memory Optimization Strategies

To optimize the memory usage of this program, we first designed a new data structure to reduce the memory consumption of the weight array. Then, to ensure the stability of RELION, we proposed a stack-based memory scheduling algorithm that is in charge of the memory operations between the CPU and GPU.

4.1 Customized Compressed Data Structure

As we mentioned previously, as the number of sampling points increases, the memory consumption of the weight array grows dramatically. Consequently, reducing the memory consumption of the weight array is of great importance.

As RELION adopts a modified expectation-maximization algorithm to iteratively optimize the results, the weight array would be used twice per iteration. In the first pass of one iteration, RELION calculates the difference between each particle and each sampled re-projection, which will fill the weight array densely. However, in the second pass of one iteration, only part of the sampling points will be considered, making the weight array a relatively sparse array.

One simple method to reduce the memory consumption of the weight array is to use the classical compressed data structure Compressed Sparse Row (CSR). Instead of storing every each one of the elements in the weight array, CSR stores only the non-zero elements and their corresponding subscripts in each row. In general, when the number of non-zero elements (n) is less than half of the total number of elements (N), this array is considered sparse ($n < 0.5 * N$). However, as for RELION, the weight array cannot always meet this criterion. For example, in the first several iterations, due to the low resolution of the estimated parameters, the weight array tends to be denser than later iterations. Under this condition, the memory consumption of the CSR structure may exceed the original memory consumption, which would damage the performance.

To cope with this issue, we designed a customized compressed data structure. The sparse weight array is generated in the second pass, during which only sampling points with high probability are considered with fine sampling steps, and one sampling point would generate 32 fine sampling points in the second pass. The weight of these 32 sampling points is stored successively. Thus, in contrast to the general sparse arrays, as shown in the top half of Figure 3, the non-zero elements of the sparse weight array are partially continuous. Making use of this trait, we can reduce the memory cost for the subscript. In the bottom half of Figure 3, instead of recording the column subscript of each non-zero element, we only store the index of the first non-zero elements in the index array.

In Figure 3, we list the details of this data structure. This data structure consists of three parts. The first part is the "Value Array", which stores all of the non-zero elements of the weight array. Then, the "Indices Array" stores the

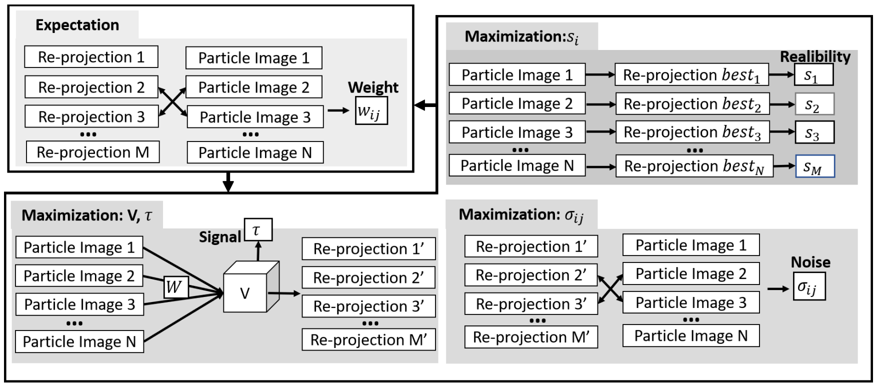
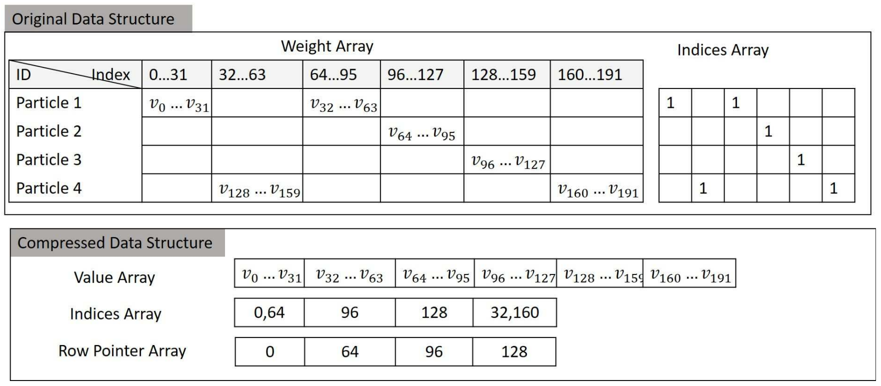


Fig. 2: The work flow of the modified algorithm



Original Data Structure

Weight Array

ID \ Index	0…31	32…63	64…95	96…127	128…159	160…191
Particle 1	$v_0 \dots v_{31}$		$v_{32} \dots v_{63}$			
Particle 2				$v_{64} \dots v_{95}$		
Particle 3					$v_{96} \dots v_{127}$	
Particle 4		$v_{128} \dots v_{159}$				$v_{160} \dots v_{191}$

Indices Array

1		1			
			1		
				1	
	1				1

Compressed Data Structure

Value Array	$v_0 \dots v_{31}$	$v_{32} \dots v_{63}$	$v_{64} \dots v_{95}$	$v_{96} \dots v_{127}$	$v_{128} \dots v_{159}$	$v_{160} \dots v_{191}$
Indices Array	0,64	96	128	32,160		
Row Pointer Array	0	64	96	128		

Fig. 3: Customized compressed data structure.

subscript of the first subscript of the 32 consecutive non-zero elements. Finally, we use "Row Point Array" to indicate the start position of each particle's non-zero elements in "Value Array".

For each particle, we can compare the compression ratio between the CSR and our structure. Let the total number of elements be N and the non-zero elements in the matrix be n. For each particle, $size\_original$, $size\_compressed$ and $size\_new$ refer to the memory cost in bytes for the original version of RELION, CSR and our method. Formula 4.1 shows their concrete memory cost. (Here $size\_value$ and $size\_subscript$ indicate the number of bytes used for one non-zero element and the subscript.) Based on these formula, we can conclude that when $n > 0.67N$, the memory consumption of CSR will exceed the original version. However, the memory consumption of our method will not exceed the original version unless $n > 0.98N$.

$$\begin{aligned}
&size\_value = sizeof(double) = 8\\
&size\_subscript = sizeof(int) = 4\\
&size\_original = N * (32 * size_{value})\\
&size\_compressed = n * (32 * (size_{value} + size_{subscript}))\\
&size\_new = n * (32 * size_{value} + size_{subscript})
\end{aligned} \tag{4.1}$$

4.2 Stack-based Stabilizing Management System

RELION adopts a modified version of the expectation-maximization method to perform reconstruction. During this procedure, numerous parameters (such as the particle images and the current resolution) need to be taken into account. These parameters become different variables used on the CPU and GPU. In addition to the large number of variables, the memory requirements of some of these variables vary during the iterations according to the sampling rate and resolution, among other factors. Consequently, the memory management system needs to handle not only the large number of variables but also the variety of memory requirements of these variables.

However, there is no fully automatic memory management system for the GPU. The native memory allocation on the GPU is generally random. Repeated allocation and de-allocation will introduce memory fragmentation, which will split the memory space into many small pieces. In the later stage of the iterations, these fragments will introduce memory allocation failure and cause the program to crash.

To manage these memory operations, RELION uses a linked list to manage its GPU global memory. In the linked list, each node indicates one piece of the consecutive memory, and the adjacent memory will be merged into one node. Thus, the linked list can reduce fragments to a degree. However, this method cannot prevent the generation of fragments. When available memory is separated by used memories, the issue of fragmentation still exists.

Importantly, the GPU memory allocation operation and the kernel running on the GPU are asynchronous. This property leaves us more freedom to manage the GPU memory. One important property of the expectation-maximization algorithm used by RELION is that it is an iterative method. In contrast to direct methods, iterative methods update the estimated parameters based on the estimation generated from previous iterations. Consequently, the algorithm is processed in one round after another. Many of the memory operations within the iteration are predictable and repeatable. For example, the allocation of projection images is only needed at the beginning of each iteration, and de-allocation is performed at the end of each iteration.

In addition, within one iteration, hundreds of thousands of particles are processed, but the basic processing procedure of each particle is similar. For example, each of the particle images needs to be transformed from real space to Fourier space first. Therefore, at the beginning of processing one particle, we need to allocate the memories for the particles in Fourier space and free them at the end of processing one particle.

Based on the introduction above, we can conclude that the memory operation can be managed in groups based on their life cycle: global scope, iteration scope, and per particle scope. One example is shown in Figure 4. At the beginning, variables needed by the whole program will be allocated. Then, at the beginning of each iteration, variables in the iteration scope will be allocated. Variables such as the particle images in Fourier space will then be allocated for each particle. Finally, at the end of their life cycle, the particles will be freed.

Another consideration is that, under some conditions, the memory requirement exceeds the total available memory on the GPU. In some cases, we can reduce the memory requirement by reducing the number of particles processed at one time. In our program, we also consider the extreme condition. Within the processing of one particle, the particle images need to be compared with many re-projection images from different orientations. When the size of one re-projection grows, the total memory requirement grows rapidly. Therefore, we add one more group for the orientations. When the memory of the GPU is limited, we can process all the re-projections in several batches, and the corresponding variables will be allocated and de-allocated at the beginning and end of the batch. This group is shown in Figure 4. Based on these rules, we can build a stack-based memory management system thoroughly to avoid memory fragments. This memory management system can also be used in other iterative methods.

5 Parallel Optimization Methods

In this section, we try to speed up the GPU parallel algorithm of RELION by proposing two highly efficient parallel algorithms: the in-kernel top-sum parallel algorithm and weight calculation parallel algorithm.

5.1 In-kernel Top-sum Selection Parallel Algorithm

As introduced in Section 2.3, in the field of computing technology, there is a traditional problem called "top-k", which involves selecting the top K (number K is constant) large or small elements from an unordered list. Considerable efforts have focused on the classic "top-k" problem in parallel.

In RELION, one similar but more complicated problem occurs. Because RELION implements a modified version of the adaptive expectation maximization algorithm, it first adopts a coarse grid to estimate the contribution of orientations. Then, a sub-domain of orientations ϕ whose total calculated probabilities exceed a fraction (99.9%) of the sum of all probabilities is retained. At the end of the first pass, we need to find the stopping point at which the sum of the values larger than the stopping point barely exceeds 99.9% of the total probability.

Instead of selecting the top k largest elements, we need to select the largest elements whose sum reaches 99.9% of the sum total. We define this problem as "Top-sum" selection. To the best of our knowledge, no effective parallel solution has been achieved.

To acquire the correct stopping point, the sequential version of RELION first allocates a block of memory to store the weight array of one particle and then eliminates zero elements from the array. RELION then sorts the array. Finally, the elements are summed in descending order one by one until the sum exceeds 99.9% of the sum of the total probability. The basic idea of the parallel version is similar. The data are sorted, and the elements are then selected sequentially. This approach has two weaknesses. First, this method requires a lot of memory to store a complete copy of the weight array on the GPU. As we discussed above, the size of the weight array may increase significantly in some iterations, while the memory of the GPU is quite limited. Second, the sorting operation is quite time-consuming and unnecessary. Sorting on the GPU costs many synchronization and write operations, which is very time consuming. We only need to know the stopping point, so sorting is unnecessary.

Based on the consideration above, we propose our own parallel method based on the following considerations: 1. We should not sort all of the data. 2. To obtain the partial 99.9% of the total probability, the data must be partially sequential. 3. During the process of summation, the information about partial sums will be lost.

With these considerations, we propose a parallel top-sum algorithm. In general, this algorithm has four basic steps (1. Bucket select, 2. Locate bucket, 3. Select data, 4. Use bucket sort to obtain stop point). We show the basic work-flow of our method in Figure 5. We summarize our in-kernel top-sum algorithm with pseudo code in Algorithm 1. In Algorithm 1, Lines 5 to 10 shows Step 1, and Lines 11 to 13 show Step 2. Then Step 4 is shown in line 18 to 19. We also use Algorithm 2 to show the details of Step 3.

First, in the "Bucket select" step, we choose some values as the pivots $(P_1, P_2, P_3 ... P_m + 1)$ to split the data into different buckets $(B_1, B_2 ... B_m)$. For example, if w_i is larger than pivot P_3 and smaller than pivot P_4, w_i belongs to bucket B_3. In this method, we select these pivots uniformly $P_j = \frac{j*(max(w_i) - min(w_i))}{m} + min(w_i)$. In practice, we use a thread to process w_i in parallel.

As shown in the left part of Figure 5. First, we locate the bucket that w_i belongs to. Then, we record the sum $(S[m])$ and number $(C[m])$ of each bucket. Finally, we convert the sum of each bucket to the prefix sum $(Ps[m])$.

With the prefix sum array $(Ps[m])$, we can determine which bucket $(B[t])$ contains the stopping point. If the number of elements in the corresponding bucket is less than the total number of threads $tNum$, we can enter step 3. Otherwise, we will repeat step 1 and step 2.

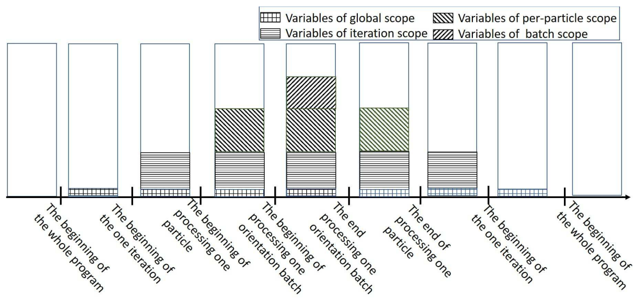


Fig. 4: Illustration of the memory operation on the GPU.

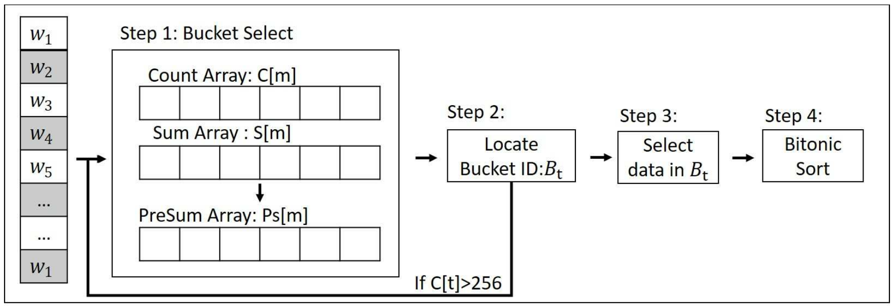


Fig. 5: The work-flow of the in-kernel top-sum solution.

In step 3, we select the candidates (w_is) in bucket $B[t]$. In contrast to the traditional parallel algorithm, because the number of candidates is less than $tNum$, the shared memory array S can store all candidates. We use a hash algorithm to load these candidates to shared memory. In step 4, we sort these candidates using a Bitonic sort algorithm so that we can locate the precise stopping point.

Algorithm 1 In-kernel Top-sum Algorithm

```
Require: Unsorted weight Array $W$
Require: Length of array $n$ and Number of buckets $m$
Require: Sum of weight array $tSum$
 1: $tNum \leftarrow totalNumberOfThreads()$
 2: $nCandidates \leftarrow 1000$
 3: $tId \leftarrow threadId()$
 4: while $nCandidates < tNum$ do
 5:   $p \leftarrow pivots()$
 6:   for $(i = 0; i < n; i = i + tNum)$ do
 7:     $bId \leftarrow LocateBucket(W[i], P)$
 8:     $C[bId] \leftarrow C[bId] + 1$
 9:     $S[bId] \leftarrow S[bId] + W[i]$
10:     $pSum \leftarrow convertSumtoPreSum(S)$
11:   end for
12:   $stopBucketId \leftarrow locateStoppingPoint(pSum, tSum)$
13:   $nCandidates \leftarrow C[stopId]$
14: end while
15: synchronize()
16: $sharedArr \leftarrow selectCandidate(W, tId, stopBucketId)$
17: synchronize()
18: $sortedArr \leftarrow BitonicSort(sharedArr)$
19: $stoppingPoint \leftarrow getStoppingPoint(sortedArr, tSum)$
```

Algorithm 2 Select Candidates within One Bucket

```
Require: Unsorted weight Array $W$
Require: The bucket index containing stopping point
    $stopBucketId$
 1: $tNum \leftarrow 256$
 2: $tId \leftarrow threadId()$
 3: $C[tId] \leftarrow 0$
 4: $S[bId] \leftarrow 0$
 5: $p \leftarrow pivots()$
 6: for $(i = 0; i < n; i = i + tNum)$ do
 7:   $bId \leftarrow LocateBucket(W[i], P)$
 8:   if $bId == stopBucketId$ then
 9:     if $C[bId] == 0$ then
10:       $S[bId] \leftarrow W[i]$
11:     else
12:       $nId \leftarrow hashToNewLoc(C)$
13:       $S[nId] \leftarrow W[i]$
14:     end if
15:   end if
16: end for
```

5.2 Weight Calculation Parallel Algorithm

As we mentioned, RELION compares each particle image with the re-projection images and then estimates the particle orientations based on the differences. One important function used to compare the particle image with re-projection images is "$getAllSquaredDifferences()$". In this function, we can calculate the squared difference of every component of the particle image and re-projection image and generate the weight value to indicate the difference. Figure 6(a) illustrates this comparison, where $w_{i\phi}$ indicate that the ith particle image is compared with the re-projection at orientation ϕ. We can use Equation 5.1 to describe this procedure:

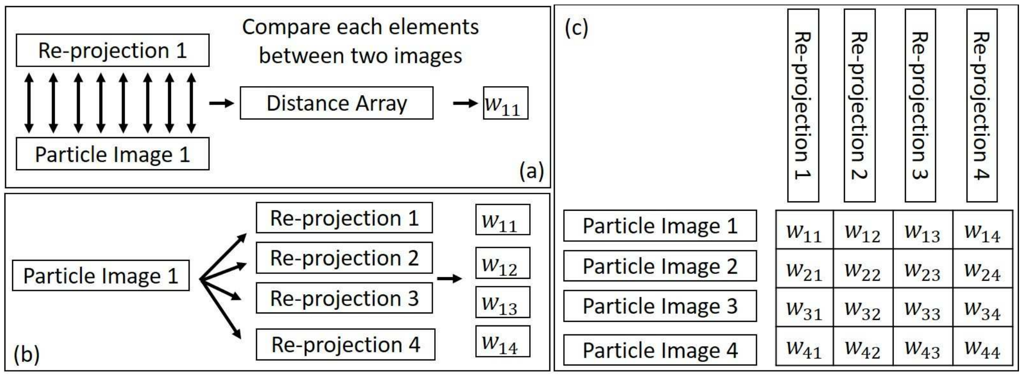


Fig. 6: Compare each component of the particle image and re-projection image

$$w_{i\phi}^{(n)} = \frac{1}{2}\sum_{k=1}^{N}((Re(p_{\Phi k})-Re(X_{ik}))^2+(Im(p_{\Phi k})-Im(X_{ik}))^2) \tag{5.1}$$

where X_{ik} is the k-th component for the ith particle image and $p_{\phi k}$ is the k-th component of the re-projection at orientation ϕ. One detail to explain is that the calculation of the first iteration differs from Equation5.1 by calculating the difference with normalized cross-correlation. However, both cross-correlations and squared differences contain two parts: comparing each component and reducing to one weight result.

This procedure is shown from one particle's perspective. As illustrated in Figure 6(b), we need to calculate the weight for each re-projection. In this whole procedure, as shown in Figure 6(c), each re-projection needs to be compared with each particle image.

To calculate one weight value $w_{i\phi}$ in parallel on the GPU, the most straightforward method is loading one whole particle image and one re-projection in one thread. If we use $size_{image}$ to indicate the size of the image, calculating the weight value requires $size_{image} * size_{image}$ times of global memory accessing, which makes the performance of this algorithm rely heavily on the memory accessing speed. As for the GPU, global memory accessing is 100 times slower than shared memory [35]. Thus, if we can replace this global memory accessing with shared memory, the running time can be greatly reduced.

However, one shortage of shared memory is that its size is quite limited. Based on these observations, we designed a new weight calculation parallel algorithm. We use Figure 7 to show the details of this method. First, we can load only a part of several particle and re-projection images. In the upper left corner of Figure 7, the white part shows the part loaded from global memory to shared memory. Using the data marked in white color, we can only calculate part of $w_{i\phi}$. However, one benefit is that we can calculate many different $w_{i\phi}$s. For the example shown in Figure 7, the first part of weights value $S_1 = \{w_{11-1}, w_{12-1}...w_{21-1}, w_{22-1}...w_{31-1}, w_{32-1}...w_{44-1}\}$. After loading the second (in light gray color) part of the data to shared memory, we can acquire the second part $S_2 = \{w_{11-2}...w_{44-2}\}$. Similarly, the third (in dark gray color) part is loaded to shared memory to calculate $S_3 = \{w_{11-3}...w_{44-3}\}$. After that, we can acquire the complete weight value ($w_{11} = w_{11-1} + w_{11-2} + w_{11-3}$), and at this point, several weight values $w_{11}, w_{12}...w_{44}$ can be acquired.

The most important advantage of this method is that the time spent accessing global memory is reduced. Take the example shown in Figure 7 as one example. In total, it can calculate $4 * 4 = 16$ different weight values. Furthermore, it only requires $4*2*size_{image}$ global memory. Consequently, for each weight value $w_{i\phi}$, the required duration of global memory access is only $\frac{4*2*size_{image}}{16} = \frac{size_{image}}{2}$. Compared with the original method, it reduced the time spent accessing global memory by $size_{image}*(size_{image}-2)$. As shared memory is available for all threads of one block, we can reuse the data and reduce the data accessing time. Taking a 32*32 block as an example, the global memory accessing time can be reduced by 32 times. For the step "Get Squared Differences (GSD)", the majority of the time is spent in the expectation step, and reducing global memory accessing can efficiently reduce the processing time of this step.

6 Experiments

6.1 Environment and Test Dataset

In our experiments, we adopted three different datasets from the EMPIAR [36]. The first dataset is 6,496 β-galactosidase (EMPIAR-10017 [15]). The size of each particle is $128 * 128$. Th second dataset is "*Plasmodium falciparum* 80S ribosome bound to the anti-protozoan drug emetine", which contains 100,000 particles with particle size $360*360$ (EMPIAR-10028 [9] [1]). The third dataset is also β-galactosidase (EMPIAR-10204) with 5000 particle with size $100 * 100$. The experiments are carried out on a machine running the Ubuntu operating system 14.04, 64-bit with an Intel(R) Xeon(R) CPU E5-2680 v3 at 2.50 GHz. The GPU card is NVIDIA GeForce GTX Titan X, with 3072 stream processers and 12288 MB global memory.

6.2 Performance of the extended refine algorithm of RELION

In this section, we compared our algorithm with the original version of RELION. To illustrate the performance, we recorded the resolution after each iteration for the three datasets mentioned above. The corresponding results are shown in Figure 8.

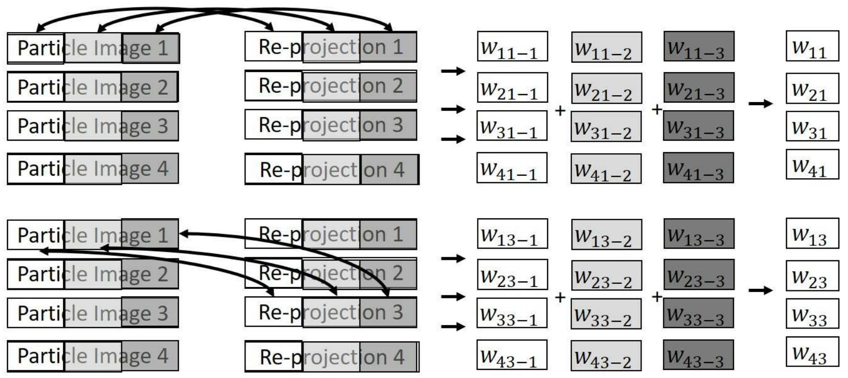


Fig. 7: Using shared memory to compare the particle image and re-projection image

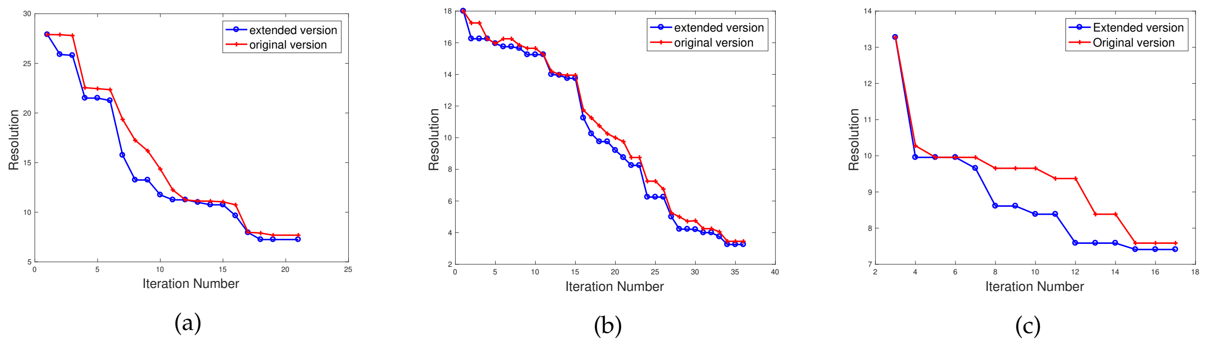


Fig. 8: The resolution after each iteration for both our extended algorithm and original RELION algorithm. (a)β-galactosidase (EMPIAR-10017) (b)*plasmodium falciparum* 80S ribosome (EMPIAR-10028) (c)β-galactosidase (EMPIAR-10204)

From these three images, in general, we can find that our method converge faster than the original version during the iterations. Especially in the middle iterations, our method can reach better resolution in less iterations. Take the dataset EMPIAR-10204 as example, in the Figure 8(c), we can find that from the seventh iteration, our method starts to get better resolution compared with the original verison. In the middle iterations, these advantages become more and more obvious. This is caused by the scalar parameter we introduced. During the first iterations, the projection error of all the particles are quite similar, the corresponding scalar parameter value is also quite close to each other. However, the estimation grows more and more accurate with the iterations. The difference between particles become evident. Especially, in the middle iterations, the value of scalar parameters for different particles differs with each other. And this is the stage where these scalar parameters make more significant contribution to the refinement procedure.

And we can see around the fifteenth iterations, the differences between the resolution of our method and the original method becomes quite small. The resolution generally converge to the final resolution. As for the final resolution, the dataset EMPIAR-10017 can acquire 7.25 Åand 7.69 Åfor our method and the original version. For the *Plasmodium falciparum* 80S ribosome, the final resolution is 3.25 Å(our method) and 3.45 Å(the original version). And for dataset EMPIAR-10204, its final resolution is 3.80 Å(our method) and 3.93 Å(the original version), respectively. Based on the final resolution, we can see that our method can increase the resolution in a certain degree.

6.3 Performance of Stack-based Memory management System

We proposed the stack-based memory management system for RELION, here we use the dataset EMPIAR-10028 to show the running procedure of the memory management system. Different from the original version of RELION who allocates the GPU memory randomly, our system allocates the memories without fragment. So we can use the top pointer of the "stack" to indicate the used GPU global memory. The memory before the "top pointer" is used memory. And memory with the larger address is continuous free memory. In Figure 9, the horizontal axis indicates each allocation operation during the running of RELION. And the vertical axis indicates the memory address. As we can see, the memory consumption of RELION shows the general cyclical pattern due to its iterative method. Also, our method can help group these memory usages with our stack-based memory management system.

6.4 Memory Consumption Optimization

We modified the data structure of weight array, which saves a substantial amount of memory. To show the performance of our customized data strucuture, we use the data set EMPIAR-10017 as benchmark. In Figure 10, we record the max memory consumption of the weight array of one particle of one iterations. As shown in Figure 10, the

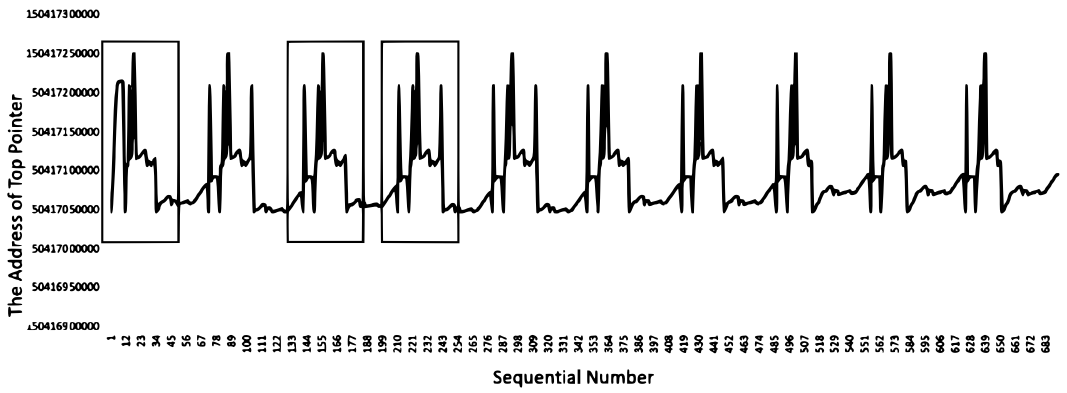


Fig. 9: The top pointer of used memory

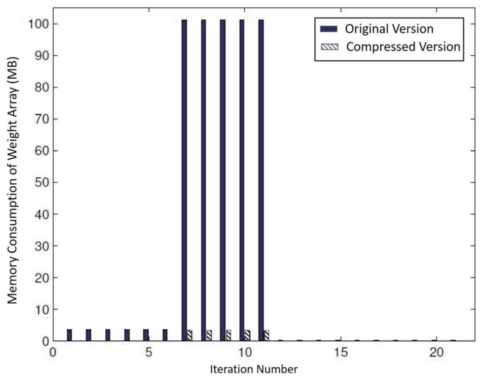


Fig. 10: Maximum memory consumption of the weight array for one particle for β-galactosidase in double mode.

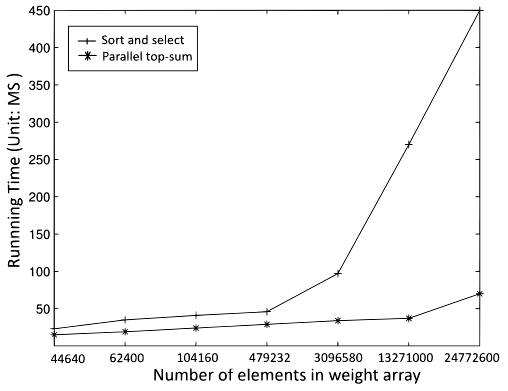


Fig. 11: The average processing time of different weight arrays.

most memory-demanding array appears in the middle of the iterations. In these iterations, our structure significantly reduces the memory requirements. The upper bound of the size of weight array is mainly the sampling number in the first pass in both float mode and double mode. The memory consumption of weight array is generally $1/32$ of the original version.

6.5 The Speedup of Top-sum Parallel Solution

To test the performance of our parallel top-sum algorithm, in this part, we compared our method with the original "sort and select" method. To show the performance, we record the average running time for different size of unsorted weight arrays. From Figure 11, we can see that for the weight array with a small number of elements ($N = 44640, 62400, 104160, 479232$), the running time of both methods is quite similar. However, for weight array with a larger number of elements, the running time of the naive "sort and select" method increases significantly. While the running time of the parallel time of our method didn't, which means our method can acquire better performance when processing larger weight array. Besides, as our method can process multiple particles at the same time, the overall performance would be much better comparing to processing only one particle at one time.

6.6 The Speedup Ratio of Weight Calculation Parallel Algorithm

In this section, we test the performance of our weight calculation parallel algorithm. In contrast to the texture memory, the performance of our method is not restricted by the total size of the re-projection. We compared our method with original GPU version of RELION. Figure 12 shows the speedup ratio of our method compared with original RELION. We tested these two methods on dataset EMPIAR-10028. In Figure 12, the horizontal axis indicates the iteration number and the vertical axis shows the speedup ratio.

As the number of sampling points for orientations and traslations may vary, the size of weight array changes accordingly. In the first few iterations, the sampling points number is quite small. Around the fifteenth iteration, the number of sampling points increases significantly. The calculation amount of the function also raises, correspondingly. So that we can make better use of computing resources of GPUs at these iterations, which would lead to better speedup ratio. As shown in Figure 12, in the first few iteratiosn, the performance of the two methods is similar. However, in iterations of larger datasets (around the fifteenth iteration), the speedup ratio increases and the advantage of reusing shared memory becomes apparent.

6.7 The General Speedup Ratio

We summarize the total speedup ratio for each and record the results in Figure 13. As our program is based on the

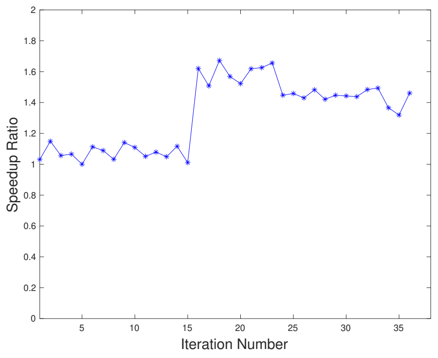


Fig. 12: Speedup ratio of the weight calculation parallel algorithm.

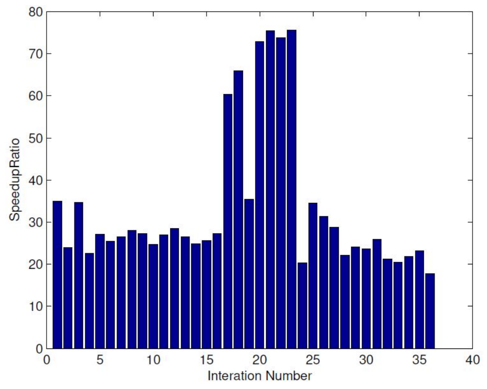


Fig. 13: The speedup ratio for each iteration.

MPI version of RELION, we don't modify the master-slave MPI parallel model. In this experiment, we compare the CUDA+MPI version with MPI version using five processes and one thread for each process. Different from the MPI version, each slave process utilize one GPU to calculate it.

For the first sixteenth iterations, the general speedup ratio is around 25x. And between iteration 20 and 23, its speed up ratio can acquire about 75x speedup ratio. This speedup ratio is caused by the larger number of sampling points.

To compare the performance of RELION's parallel version and our method, we recorded the general speedup ratio using dataset EMPIAR-10017 and EMPIAR-10204. In Table 1, we show the speedup ratio with different number of processes (in the second column) and particles per job (in the first column) in column "EMPIAR-10204 Speedup Ratio" and "EMPIAR-10017 Speedup Ratio". In general, we find that the speedup ratio is quite stable with the variation of number of processes. This is caused by the fact that we actually didn't change the architecture of the MPI processes. The speedup ratio is mainly due to the optimization of the weight calculation and selection step. As for dataset EMPIAR-10017, it speedup ratio is about 1.19 and EMPIAR-10017 owns about 1.17 speedup ratio. And, comparing the first three rows with the rest, we can find the speedup ratio are larger with more particles in each job. This is caused by the fact that more particles lead to more calculation amount to be processed in parallel.

Number of Particles	Number of Process	EMPIAR-10017 Speedup Ratio	EMPIAR-10204 Speedup Ratio
8	3	1.17	1.15
8	5	1.19	1.15
8	7	1.18	1.16
32	3	1.21	1.18
32	5	1.22	1.19
32	7	1.21	1.19

TABLE 1: The speedup ratio with different nodes and particle per process.

7 Conclusion

In this work, we optimized the 3D refine algorithm from three aspects: the reconstruction algorithm, memory usage, and time performance. 1. We optimized the 3D refine algorithm by extending the model parameter and updated it based on the idea of back-propagation. 2. To optimize the memory usage condition, we designed a memory-efficient and memory-stabilizing management system to guarantee the robustness of our program and the efficiency of GPU memory usage. Then, we developed a novel data structure for RELION to reduce the memory consumption. 3. To speed up the GPU parallel algorithm of RELION, we proposed two highly efficient parallel algorithms: an in-kernel top-sum parallel algorithm and a weight calculation parallel algorithm. Experiments showed that the optimized 3D refine algorithm can speed up the converge procedure and increase the final resolution and that the memory system can avoid memory fragments. We achieved a better speedup ratio compared to RELION.

8 Acknowledgments

This research was supported by the National Key Research and Development Program of China (2017YFA0504702); NSFC projects Grant Nos. U1611263, U1611261, 61232001, 61472397, 61502455, and 61672493; and the Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase).

References

[1] X.-c. Bai, G. McMullan, and S. H. Scheres, "How cryo-em is revolutionizing structural biology," *Trends in biochemical sciences*, vol. 40, no. 1, pp. 49–57, 2015.

[2] X. Li, N. Grigorieff, and Y. Cheng, "Gpu-enabled frealign: accelerating single particle 3d reconstruction and refinement in fourier space on graphics processors," *Journal of structural biology*, vol. 172, no. 3, pp. 407–412, 2010.

[3] G. Tang, L. Peng, P. R. Baldwin, D. S. Mann, W. Jiang, I. Rees, and S. J. Ludtke, "Eman2: An extensible image processing suite for electron microscopy," *Journal of Structural Biology*, vol. 157, no. 1, pp. 38–46, 2007.

[4] S. H. Scheres, "A bayesian view on cryo-em structure determination," *Journal of molecular biology*, vol. 415, no. 2, pp. 406–418, 2012.

[5] ——, "Relion: implementation of a bayesian approach to cryo-em structure determination," *Journal of structural biology*, vol. 180, no. 3, pp. 519–530, 2012.

[6] Y. Z. Tan, S. Aiyer, M. Mietzsch, J. A. Hull, R. McKenna, J. Grieger, R. J. Samulski, T. S. Baker, M. Agbandje-McKenna, and D. Lyumkis, "Sub-2 å ewald curvature corrected structure of an aav2 capsid variant," *Nature communications*, vol. 9, 2018.

[7] R. Zhou, G. Yang, X. Guo, Q. Zhou, J. Lei, and Y. Shi, "Recognition of the amyloid precursor protein by human γ-secretase," *Science*, p. eaaw0930, 2019.

[8] M. M. Shaik, H. Peng, J. Lu, S. Rits-Volloch, C. Xu, M. Liao, and B. Chen, "Structural basis of coreceptor recognition by hiv-1 envelope spike," *Nature*, vol. 565, no. 7739, p. 318, 2019.

[9] W. Wong, X.-c. Bai, A. Brown, I. S. Fernandez, E. Hanssen, M. Condron, Y. H. Tan, J. Baum, and S. H. Scheres, "Cryo-em structure of the plasmodium falciparum 80s ribosome bound to the anti-protozoan drug emetine," *Elife*, vol. 3, p. e03080, 2014.

[10] A. Amunts, A. Brown, X.-c. Bai, J. L. Llácer, T. Hussain, P. Emsley, F. Long, G. Murshudov, S. H. Scheres, and V. Ramakrishnan, "Structure of the yeast mitochondrial large ribosomal subunit," *Science*, vol. 343, no. 6178, pp. 1485–1489, 2014.

[11] M. Liao, E. Cao, D. Julius, and Y. Cheng, "Structure of the trpv1 ion channel determined by electron cryo-microscopy," *Nature*, vol. 504, no. 7478, pp. 107–112, 2013.

[12] Y. Chen, F. Ren, X. Wan, X. Wang, and F. Zhang, *An Improved Correlation Method Based on Rotation Invariant Feature for Automatic Particle Selection*. Springer International Publishing, 2014.

[13] H. D. Tagare, A. Barthel, and F. J. Sigworth, "An adaptive expectation–maximization algorithm with gpu implementation for electron cryomicroscopy," *Journal of structural biology*, vol. 171, no. 3, pp. 256–265, 2010.

[14] F. J. Sigworth, P. C. Doerschuk, J. Carazo, and S. H. W. Scheres, "Maximum-likelihood methods in cryo-em. part i: theoretical basis and overview of existing approaches," *Methods in Enzymology*, vol. 482, p. 263, 2010.

[15] S. H. Scheres, "Single-particle processing in relion-1.3," ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion30_tutorial.pdf, p. 5, 2014.

[16] Y. Chen, Z. Wang, J. Zhang, L. Li, X. Wan, F. Sun, and F. Zhang, "Accelerating electron tomography reconstruction algorithm icon with gpu." *Biophysics Reports*, vol. 3, no. 1, pp. 36–42, 2017.

[17] F. Zhang, J. Zhang, A. Lawrence, F. Ren, X. Wang, Z. Liu, and X. Wan, "Bsirt: a block-iterative sirt parallel algorithm using curvilinear projection model," *IEEE Trans Nanobioscience*, vol. 14, no. 2, pp. 229–236, 2015.

[18] Z. Wang, Y. Chen, J. Zhang, L. Li, X. Wan, Z. Liu, F. Sun, and F. Zhang, "Accelerating electron tomography reconstruction algorithm icon using the intel xeon phi coprocessor on tianhe-2 supercomputer," in *International Symposium on Bioinformatics Research and Applications*, 2017, pp. 258–269.

[19] M. Hu, H. Yu, K. Gu, Z. Wang, H. Ruan, K. Wang, S. Ren, B. Li, L. Gan, S. Xu *et al.*, "A particle-filter framework for robust cryo-em 3d reconstruction," *Nature methods*, vol. 15, no. 12, p. 1083, 2018.

[20] A. Punjani, J. L. Rubinstein, D. J. Fleet, and M. A. Brubaker, "cryosparc: algorithms for rapid unsupervised cryo-em structure determination," *Nature methods*, vol. 14, no. 3, p. 290, 2017.

[21] J. J. Fernandez, "High performance computing in structural determination by electron cryomicroscopy," *Journal of structural biology*, vol. 164, no. 1, pp. 1–6, 2008.

[22] M. Schmeisser, B. C. Heisen, M. Luettich, B. Busche, F. Hauer, T. Koske, K.-H. Knauber, and H. Stark, "Parallel, distributed and gpu computing technologies in single-particle electron microscopy," *Acta Crystallographica Section D: Biological Crystallography*, vol. 65, no. 7, pp. 659–671, 2009.

[23] P. Cossio and G. Hummer, "Likelihood-based structural analysis of electron microscopy images," *Current opinion in structural biology*, vol. 49, pp. 162–168, 2018.

[24] D. Kimanius, B. O. Forsberg, S. H. Scheres, and E. Lindahl, "Accelerated cryo-em structure determination with parallelisation using gpus in relion-2," *eLife*, vol. 5, p. e18722, 2016.

[25] F. J. Sigworth, P. C. Doerschuk, J. M. Carazo, and S. H. W. Scheres, "An introduction to maximum-likelihood methods in cryo-em." *Methods in Enzymology*, vol. 482, pp. 263–294, 2010.

[26] N. C. Dvornek, F. J. Sigworth, and H. D. Tagare, "Subspaceem: A fast maximum-a-posteriori algorithm for cryo-em single particle reconstruction," *Journal of Structural Biology*, vol. 190, no. 2, pp. 200–214, 2015.

[27] G. H. Shan, J. Liu, and X. B. Chi, "Local maximum absorption method for structure fitting in cryoem," *Key Engineering Materials*, vol. 460-461, pp. 77–82, 2011.

[28] S. H. W. Scheres, H. Gao, M. Valle, G. T. Herman, P. P. B. Eggermont, and et.al., "Disentangling conformational states of macromolecules in 3d-em through likelihood optimization." *Nature Methods*, vol. 4, no. 1, pp. 27–29, Jan 2007.

[29] S. H. W. Scheres, M. Valle, P. Grob, E. Nogales, and J. M. Carazo, "Maximum likelihood refinement of electron microscopy data with normalization errors," *Journal of Structural Biology*, vol. 166, no. 2, pp. 234–240, 2009.

[30] H. Su, W. Wen, X. Du, X. Lu, M. Liao, and D. Li, "Gerelion: Gpu-enhanced parallel implementation of single particle cryo-em image processing," *bioRxiv*, p. 075887, 2016.

[31] G. Das, "Top-k algorithms and applications," in *Database Systems for Advanced Applications*. Springer, 2009, pp. 789–792.

[32] T. Alabi, J. D. Blanchard, B. Gordon, and R. Steinbach, "Fast k-selection algorithms for graphics processing units," *Journal of Experimental Algorithmics (JEA)*, vol. 17, pp. 4–2, 2012.

[33] U. A. Markus Billeter, Ola Olsson, "Efficient stream compaction on wide simd many-core architectures," vol. 2009, pp. 159–166, 2009.

[34] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning." vol. 930, pp. 195–201, 06 1995.

[35] N. Corporation, "Cuda in c best practices guide," https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf, 2016.

[36] A. Iudin, P. K. Korir, J. Salavert-Torres, G. J. Kleywegt, and A. Patwardhan, "Empiar: a public archive for raw electron microscopy image data," *Nature methods*, vol. 13, no. 5, p. 387, 2016.

Jingrong Zhang received a BS degree in computer science from Yanshan University, China. She is a PhD candidate at the Institute of Computing Technology (ICT) and University of the Chinese Academy of Sciences (UCAS). Her current research interests include bioinformatics, biomedical image processing, and high-performance computing.

Zihao Wang received a BS degree in software engineering from Xiamen University, China. He is a third-year PhD student at the Institute of Computing Technology and Chinese Academy of Sciences, Beijing, China. His current research interests include bioinformatics, image processing and high-performance computing.

Zhiyong Liu received a PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). He is the Chair professor of ICT, CAS. His current research interests include high-performance algorithms and architecture, parallel processing and bioinformatics.

Fa Zhang received a PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). He is a professor at ICT, CAS. His current research interests include bioinformatics, biomedical image processing, and high-performance computing.