Vina-GPU 2.0：further accelerating AutoDock Vina and its derivatives with GPUs

Ji Ding [1,2], Shidi Tang [1,2],Lingyue Wang [3], Qinqin Huang [3], Haifeng Hu [3], Ming Ling[4], Jiansheng Wu[1,2,]*

(1.School of Geographic and Biological Information, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China; 2.Smart Health Big Data Analysis and Location Services Engineering Research Center of Jiangsu Province, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China; 3.National ASIC System Engineering Technology Research Center, Southeast University, Ltd., Nanjing, 210096, China; 4.School of Telecommunication and Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing,210023, China)

***Correspondence: jansen@njupt.edu.cn**

Abstract: Modern drug discovery typically faces large virtual screens from huge compound databases where multiple docking tools are involved for meeting various real scenes or improving the precision of virtual screens. Among these tools, AutoDock Vina and its numerous derivatives are the most popular and have become the standard pipeline for molecular docking in modern drug discovery. Our recent Vina-GPU method realized 14-fold acceleration against AutoDock Vina on a piece of NVIDIA RTX 3090 GPU in one virtual screening case. Further speedup of AutoDock Vina and its derivatives with GPUs is beneficial to systematically push their popularization in large-scale virtual screens due to their high benefit-cost ratio and easy operation for users. Thus, we proposed the Vina-GPU 2.0 method to further accelerate AutoDock Vina and the most common derivatives with new docking algorithms (QuickVina 2 and QuickVina-W) with GPUs. Caused by the discrepancy of their docking algorithms, our Vina-GPU 2.0 adopts different GPU acceleration strategies. In virtual screening for two hot protein kinase targets RIPK1 and RIPK3 from the DrugBank database, our Vina-GPU 2.0 reaches an average of 65.6-fold,1.4-fold and 3.6-fold docking acceleration against the original AutoDock Vina, QuickVina 2 and QuickVina-W while ensuring their comparable docking accuracy. In addition, we develop a friendly and installation-free graphical user interface (GUI) tool for their convenient usage. The codes and tools of Vina-GPU 2.0 are freely available at https://github.com/DeltaGroupNJUPT/Vina-GPU-2.0, coupled with explicit instructions and examples.

Keywords: Virtual screening; AutoDock Vina; Vina-GPU 2.0; GPU

1. Introduction

Previous virtual screening schemes were typically executed on only a scale of $10^6 \sim 10^7$ compounds. Such scale of compound sizes would seriously affect the capability and improve the failure risk ratio in modern drug discovery. As willing, the whole chemical space of small drug-likeness molecules was expected to reach more than 10^{60} compounds[1]. The compound scale of large virtual screens is of vital importance in modern drug discovery since the more compounds to be screened, the lower the risk of false positives and the higher the quality of the lead compounds[2]. Recent experiments have also shown that ultra-large-scale virtual screening could improve the success rate of true positives. In large virtual screens from huge compound databases, multiple docking tools are typically involved for meeting various real demands（such as primary screening and fine screening）or for improving the precision of virtual screens of hit candidates. Among these tools, the AutoDock Vina suite[3], which involved the original AutoDock Vina and its numerous derivatives, was the most popular and has become the gold standard for molecular docking in modern drug discovery.

Previous studies demonstrated that, in large-scale virtual screening scenes, primitive AutoDock Vina and its derivatives cannot meet the speed demand of the majority of users in modern drug discovery. Therefore, their acceleration has become a central problem in current virtual screens of drug hits from huge compound databases. Till now, there have been many attempts for the acceleration of the AutoDock Vina suite on large virtual screens [6–8]. For instance, VirtualFlow is an automated drug discovery platform that speeded up AutoDock Vina and its derivatives in virtual screens of an ultra-large compound database with 1.4 billion molecules by running over 160,000 CPUs[6]. Recently, DP Technology proposed the Uni-Dock, a GPU-accelerated docking method on AutoDock Vina 1.2, and implemented the ultra-large virtual screens from the Enamine Diverse REAL drug-like set with 38.2 million compounds targeting the KRAS protein by leveraging a GPU cluster which contains 100 NVIDIA V100 GPUs, and it reached more than 1000-fold speed-up when compared with the single-CPU-core version[9].

As all we know, GPU is an ideal means of acceleration for common users due to its low barrier, high cost-effectiveness, and ease of development. Our recently proposed Vina-GPU[10] method realized the acceleration of AutoDock Vina with GPUs, which solved the difficulty of implementing its parallel acceleration on GPUs caused by the seriality design of the AutoDock Vina algorithm. Our Vina-GPU method realized the 14-fold acceleration in one typical virtual screening case, and it achieved an average of 21-fold and a maximum of 50-fold docking acceleration in a benchmark dataset with 140 complexes against the original AutoDock Vina while ensuring their comparable docking accuracy. Due to its value and excellent performance, our Vina-GPU method had rapidly gained a lot of attention, whose GitHub codes have earned 36 stars and 17 forks, and whose citations had received thousands of full-paper downloads. The derivatives of AutoDock Vina can be classified into two categories. The first category has the optimization of their docking algorithm, such as QuickVina[11], QuickVina 2, QuickVina-W, Vina-Carb[12], AutoDock VinaXB[13] and Vinardo[14], etc. The second category has no change in its docking algorithm but add some new functions, such as AutoDock Vina 1.2.0[15] and Smina[16], etc. For instance, AutoDock Vina 1.2.0 supports the modeling of some specific features such as macrocycles or the explicit water molecules and involves the AutoDock4.2 scoring function, and also provides the simultaneous docking of multiple ligands or a batch docking of large size of ligands [15]. Smina is an enhancement of AutoDock Vina that especially supports more custom functions for virtual screening [16]. Our previous Vina-GPU method realized the improvements in the docking algorithm of AutoDock Vina which is suitable to run on the GPU. Thus, the same acceleration scheme can also work on the GPU speedup of the second category of AutoDock Vina derivatives. For example, DP Technology accelerated the AutoDock Vina 1.2 with 100 GPUs by using our Vina-GPU acceleration algorithm [9]. Therefore, we just focus on the first category of AutoDock Vina derivatives and optimize their docking algorithms to make them suitable for the efficient parallel acceleration on GPUs in this paper. Among these methods, QuickVina 2 and QuickVina-W are very popular because of their fast docking speed and their code framework similar to that of AutoDock Vina, so they are chosen as the representatives for implementing the GPUs acceleration in this paper. QuickVina 2 was to promote the docking speed by relying on the novel first-order-consistency-check heuristics which can move some unnecessary local searches and keep the precision of the original AutoDock Vina [4]. QuickVina-W could improve the accuracy and speed of molecular docking by adding inter-process communication, and QuickVina-W supports blind docking which can eliminate the demand of running the docking tool several times.

To systematically push the popularization of the AutoDock Vina suite in modern drug discovery, we propose a novel method Vina-GPU 2.0 to realize the acceleration of representative AutoDock Vina derivatives (QuickVina 2, QuickVina-W) and the further speedup of AutoDock Vina with GPUs. Caused by the discrepancy of their docking algorithms, our Vina-GPU 2.0 adopts different GPU acceleration strategies. For Vina-GPU+, it implemented the further acceleration of the grid cache of Vina-GPU, which is valuable to facilitate the molecular docking of a large number of ligands with a single protein receptor in real virtual screening scenarios. This implementation ensures that Vina-GPU+ can utilize thousands of compute cores on the GPU to compute the grid cache. For QuickVina 2-GPU, it speeds up the Monte-Carlo based simulated annealing as well as the BFGS process in QuickVina 2 [4]with GPUs. QuickVina 2-GPU applies large-scale parallelism on the Monte-Carlo based iterated docking threads and then significantly reduces the search depth in each thread. This implementation ensures that QuickVina 2-GPU can leverage thousands of computational cores on GPU and achieve large-scale parallelization and acceleration. For QuickVina-W-GPU, we accelerate the simulated annealing and BFGS process of QuickVina-W [5]with GPUs. QuickVina-W-GPU realizes large-scale docking threads for parallel running the Monte-Carlo based iterated process and significantly decreases the search depth in each thread. Also, a heterogeneous OpenCL implementation was efficiently executed on QuickVina-W-GPU by converting the octree structure into a global buffer whose history points are stored. These implementations can ensure that QuickVina-W-GPU leverages thousands of computational cores on GPU and achieve a large-scale parallelization and acceleration, and realizes the thread communication on GPUs core.

Apoptosis and necroptosis are two kinds of different mechanisms of cell death. Apoptosis is mediated by caspases, whereas in the absence of apoptotic conditions, the RIPK1 (Receptor-interacting protein kinase 1)[17] and its downstream RIPK3 (Receptor-interacting protein kinase 3)[18] and MLKL[19] will activate the programmed necrosis pathway. Numerous studies have confirmed that RIPK1 and RIPK3 are key regulators of apoptosis, necrosis, and inflammatory pathways[20]. RIPK1 and RIPK3 have emerged as effective targets for the treatment of various diseases such as neurodegenerative diseases, autoimmune diseases, and inflammation. Currently, studying small molecule inhibitors targeting RIPK1 and RIPK3 has become a hot topic.

In real virtual screening on RIPK1 from DrugBank, our Vina-GPU 2.0 reaches the 70.67-fold,1.46-fold and 3.23-fold docking acceleration on one NVIDIA RTX 3090 GPU against the original AutoDock Vina, QuickVina 2 and QuickVina-W while ensuring their comparable docking accuracy. For RIPK3, Vina-GPU 2.0 improves the original AutoDock Vina, QuickVina 2 and QuickVina-W by 60.54-fold,1.41-fold and 3.97-fold docking acceleration on one NVIDIA RTX 3090 GPU while maintaining their equivalent docking accuracy. For benchmark tests on 140 complexes, Vina-GPU 2.0 achieves an average docking acceleration of 37.90-fold,1.74-fold and 5.52-fold when compares to the original AutoDock Vina, QuickVina 2 and QuickVina-W based on comparable docking accuracy. In addition, we develop a friendly and installation-free graphical user interface (GUI) tool for their convenient usage. The codes and tools of Vina-GPU 2.0 are freely available at https://github.com/DeltaGroupNJUPT/Vina-GPU-2.0, coupled with explicit instructions and examples.

2. Methodology

We systematically analyzed the algorithms of AutoDock Vina, QuickVina 2 and QuickVina-W, and rewrote their code to implement the heterogeneous OpenCL architecture with GPU cards. Vina-

GPU 2.0 employs different GPU acceleration strategies as a result of the differences in their docking algorithms. Vina-GPU+ is an upgraded version of Vina-GPU that achieves further GPU acceleration for AutoDock Vina, QuickVina 2-GPU is the method that implements GPU acceleration for QuickVina 2, and QuickVina-W-GPU is the algorithm that provides GPU acceleration for QuickVina-W.

2.1 Vina-GPU+

The heterogeneous OpenCL implementation of Vina-GPU+ is shown in Figure 1, which consists of three blocks. The conformations preparation is the primary responsibility of block I, which is implemented on the CPU. Block II is executed on kernel 1 of the GPU core and prepares the grid cache for calculating the conformation energy. The interaction forces between probe atoms and atoms in the docking box are obtained by running parallel calculations of GPU's kernel 1. Block III is in charge of molecular docking of a receptor with all ligands on GPU's kernel 2.

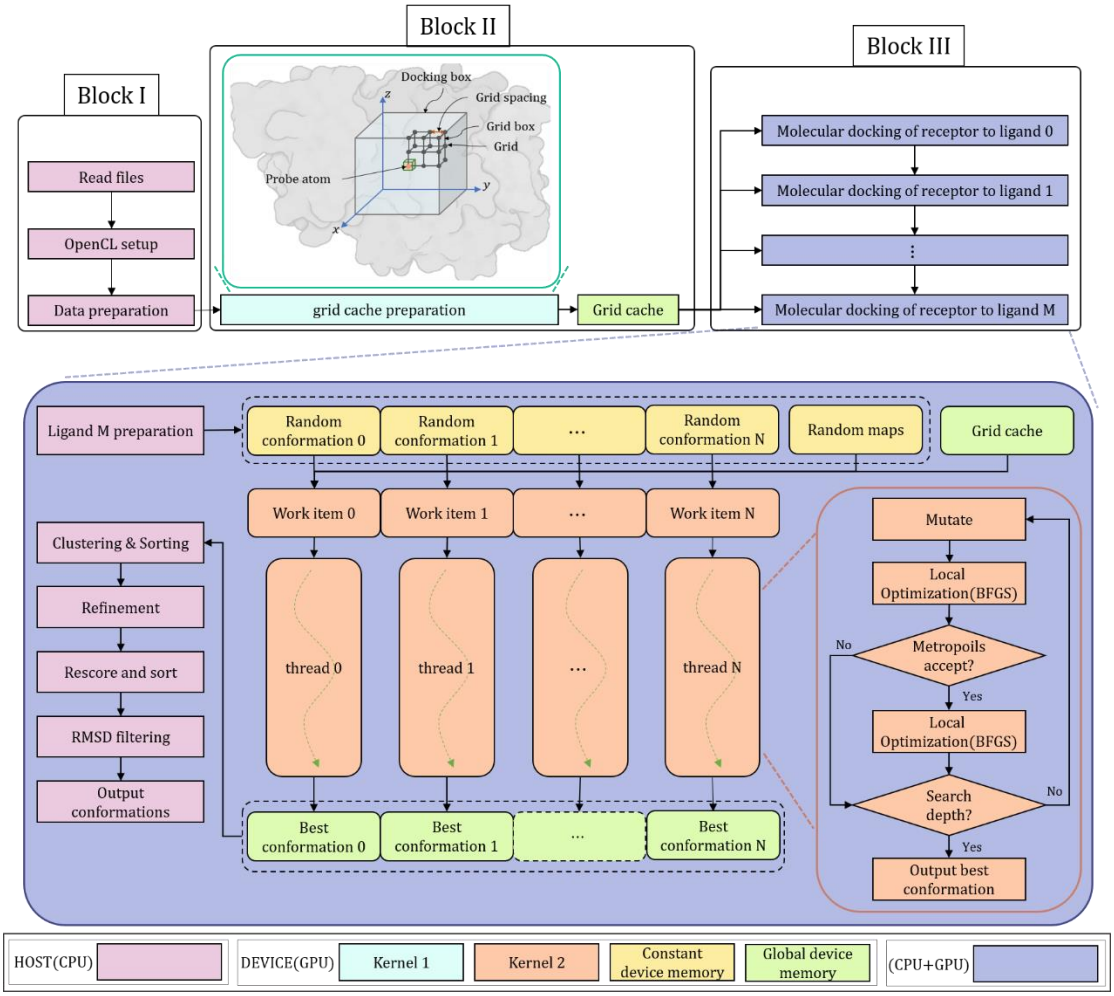

Figure 1. Vina-GPU+ implementation by using the OpenCL architecture. It consists of three blocks. Block I is mainly in charge of data preparation and OpenCL setup. The grid cache preparation for computing the energy of a conformation is implemented in block II. Block III focuses on the molecular docking of a receptor and all ligands.

2.1.1 Data Preparation

Three operations that make up Data Preparation are the read files, the OpenCL setup and the data preparation (Block I in Figure 1). These operations are implemented on the CPU core. In detail, the 'Read files' operation reads ligand and protein files in .pdbqt format, the center of the docking box (indicated by $center_x, center_y, center_z$) and the volume of the docking box (denoted

by $size_x, size_y, size_z$). Configuring the OpenCL environment is done through the 'OpenCL setup' operation (platform, device, context, queue, program and kernels). Random maps are created for producing probability random numbers in the 'Data preparation' operation. Afterward, all data is rearranged to load in the GPU memory following the way of access (read-only or read-write). The random maps are allotted in the constant memory.

2.1.2 Grid Cache Preparation

When facing the molecular docking of a receptor with many ligands in virtual screens, our previous Vina-GPU needs to recalculate the grid cache for each ligand and computes as follows: first, the docking box is divided into multiple grid boxes whose volume is represented by $gridsize_x, gridsize_y, gridsize_z$ and whose quantized coordinates are obtained by calculating. Then, Vina-GPU traverses these grid points to ascertain atomic types and calculates the interaction forces between each grid point and the receptor, and finally stores the results in the grid cache. However, we discovered that recalculating the grid cache for different ligands is time-consuming behavior. Vina-GPU+ will accelerate docking by calculating the intermolecular energies of the 17 different atom types at the grid points in advance. Since it is an independent process to calculate the interaction forces between each grid point and the receptor, Vina-GPU+ computes the grid cache in parallel by using the kernel1 of the GPU platform. The number of threads in kernel1 is determined by the size of grid points. For instance, black points are the grid points in figure 1, each grid point is assigned to a work item of kernel 1, and each work item calculates the intermolecular energy of the grid point which currently has 17 types. The grid caches are stored in the global memory and are shared by all GPU threads. Vina-GPU+ speeds up molecular docking and reduces the communication times of CPU and GPU by performing batch docking and only needs to calculate the grid cache once.

2.1.3 molecular docking

Block III is implemented on both CPU and GPU which focuses on the molecular docking of a receptor and ligands (highlighted in purple). The CPU is the main responsibility for the preparation and post-refinement of conformations (highlighted in pink). The allocated constant memory (highlighted in yellow) is used for the initialization and computation throughout the reduced-step Monte-Carlo iterated local search processes in the GPU kernel1 (highlighted in orange), and global memory stores the final best conformations (highlighted in green). The ligand preparation which is above the generation of random initial conformations is performed on the CPU. These initial conformations were then rearranged and stored into the constant memory. The kernel 2 of GPU is capable of running thousands of reduced-steps iterated local search processes simultaneously.

Each process is represented by a docking thread. For each thread, an OpenCL work item is offered a randomly initialized conformation $\boldsymbol{C}$, which is described by its position, orientation and torsion (POT):

$$\boldsymbol{C} = \{x, y, z, a, b, c, d, \psi_1, \psi_2, \ldots, \psi_{N_{rot}}\} \quad (1)$$

where x, y, z indicate the conformation position in a pre-determined searching space; a, b, c, d represent the orientation as a rigid body in the form of a quaternion; $\psi_1, \psi_2, \ldots, \psi_{N_{rot}}$ stand for the torsions of N_{rot} rotatable bonds. Moreover, a new conformation $\boldsymbol{C}'$ is generated by randomly mutating one POT element of conformation $\boldsymbol{C}$ with the uniform distribution:

$$\boldsymbol{C}' = R(\boldsymbol{C}) \quad (2)$$

where $R(\cdot)$ is a random jitter function to perturb the conformation. The scoring function which quantifies the potential energy of the binding pose continually assesses the conformation.

Usually, the potential energy e is computed by adding intermolecular energy and intramolecular energy:

$$e = e_{intra} + e_{inter} \tag{3}$$

The interaction energy of the paired atoms inside the ligand is defined by the e_{intra}, while the interaction energy between the ligand and the receptor is denoted by the e_{inter}. Trilinear interpolation is used to calculate the e_{inter} by looking up the grid cache.

Since both e_{inter} and e_{intra} are connected to the binding pose, the scoring function SF could be expressed in terms of POT variables:

$$\boldsymbol{SF} = f\ (x, y, z, a, b, c, d, \psi_1, \psi_2, \ldots, \psi_{N_{rot}}) \tag{4}$$

Following the energy evaluation, the ligand conformation is updated by minimizing the scoring function SF utilizing a Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization. First, initialize the hessian matrix $\boldsymbol{B}_0 \in \mathbb{R}^{(7+N_{rot})\times(7+N_{rot})}$. $\boldsymbol{B}_0$ is initiated with identity matrix $\boldsymbol{E}$, Initial random conformation $\boldsymbol{x}_i = \boldsymbol{C}_i'$, the $(k\ +\ 1)^{th}$ iteration of the BFGS is calculated as follows:

Calculating direction $\boldsymbol{d}_k$:

$$\boldsymbol{d}_k = -\boldsymbol{B}_k^{-1} \nabla \boldsymbol{SF}(\boldsymbol{x}_k) \tag{5}$$

Where $\boldsymbol{B}_k^{-1}$ is the inverse matrix of the Hessian matrix generated by the k^{th} iteration process. Calculating the step size in the direction $\boldsymbol{d}_k$ by using the Armijo criterion:

$$\alpha_k = argmin\ SF(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \tag{6}$$

Updating the conformation $\boldsymbol{x}_{k+1}$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

Updating the hessian matrix $\boldsymbol{B}_{k+1}$:

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k^T}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} \tag{7}$$

Where $\boldsymbol{y}_k = \nabla \boldsymbol{SF}(\boldsymbol{x}_{k+1}) - \nabla \boldsymbol{SF}(\boldsymbol{x}_k)$, $\boldsymbol{s}_k = \alpha_k \boldsymbol{d}_k$. The BFGS iteration process ends if the following conditions are met:

$$\|\nabla \boldsymbol{SF}(\boldsymbol{x}_{k+1})\| \le \varepsilon\ or\ k = search\_depth \tag{8}$$

Where $0 < \varepsilon \le 1$, the step size along the decreasing direction $\boldsymbol{d}_k$ of the SF values, is denoted by $\alpha(\alpha_k)$. The conformation of the BFGS output is $\boldsymbol{x}_{k+1}$.

The energy e_0 before the mutation and the energy e_{opt} after the optimization are compared using a metropolis acceptance criterion[21] to determine whether to accept or reject the optimized conformation. The accept probability P is shown as:

$$P = \begin{cases} 1 & e_0 > e_{opt} \\ \frac{\exp(e_0 - e_{opt})}{1.2} & e_0 \le e_{opt} \end{cases} \tag{9}$$

It suggests that the energy of the acceptable conformation is more probably to be lower. After being approved, BFGS will further evaluate and optimize the conformation. As convergence approaches, the following iteration keeps updating the earlier optimized conformations. Finally, the CPU receives all of the best conformations that work items have discovered. All the best conformations are clustered and sorted into the container according to docking scores. The top k conformations will be concretely refined, rescore, sort, and if two of these conformations have RMSDs less than 1Å, only the conformation with the lower score is retained before the final ligand file is generated.

The pseudocode of our Vina-GPU+ is proposed by Algorithm 1. In Algorithm 1, *Grid_cache*(.) means calculating the grid cache. *random_conformation(.)* represents the generation of random

initial conformations. $Mutate(.)$ indicates a random POT mutation of a ligand conformation; $BFGS(.)$ denotes the BFGS optimization method, as defined in Equations (5)–(8); $Scoring(.)$ signifies the potential energy of a binding pose given in Equations (3) and (4); $Metropolis(.)$ is the metropolis acceptance criterion mentioned in Equation (9); and $Clustering\&Sorting\&refinment\&RMSD\_filtering(.)$ is the aggregation, filtering (based on the RMSD) and reordering (based on the docking score) of all ligand conformations within all threads.

Algorithm 1 Vina-GPU+

Input: ligands:$\{\boldsymbol{L}_0, \boldsymbol{L}_1, \cdots, \boldsymbol{L}_M\}$, protein:$\boldsymbol{P}$

Output: top k ligand conformations:

$\{\boldsymbol{C}^*_{00}, \boldsymbol{C}^*_{01}, \cdots, \boldsymbol{C}^*_{0k}, \boldsymbol{C}^*_{10}, \boldsymbol{C}^*_{11}, \cdots, \boldsymbol{C}^*_{1k}, \cdots, \boldsymbol{C}^*_{M0}, \boldsymbol{C}^*_{M1}, \cdots, \boldsymbol{C}^*_{Mk}\}$

$\boldsymbol{G}_t = Grid\_cache(\boldsymbol{P})(t = 0,1, \cdots 16)$
for $\boldsymbol{L}_i(i = 0,1, \cdots M)$ **do**
  for $thread = 0,1,2, \cdots, n$ **do**
    $\boldsymbol{C}_{thread} = random\_conformation(\boldsymbol{L}_i)$
  end for
  for all $\boldsymbol{C}_{thread}(thread = 0,1,2, \cdots, n)$ **concurrently** ***do***
    for all $search\_depth = 0,1,2, \cdots, s$ **do**
      $Mutate(\boldsymbol{C}_{thread})$
      $BFGS(\boldsymbol{C}_{thread})$
      if $search\_depth == 0 \,||Metropolis(\boldsymbol{C}_{thread}, \boldsymbol{C}_{thread_{best}})$ **then**
        $\boldsymbol{C}_{thread_{best}} = \boldsymbol{C}_{thread}$
        $BFGS(\boldsymbol{C}_{thread_{best}})$
      end if
    end for
  end for
  $Clustering\&Sorting\&refinment\&RMSD\_filtering(\boldsymbol{C}_{0_{best}}, \boldsymbol{C}_{1_{best}}, \cdots, \boldsymbol{C}_{n_{best}})$
  return $\{\boldsymbol{C}^*_{i0}, \boldsymbol{C}^*_{i1}, \cdots \boldsymbol{C}^*_{ik}\} \in \{\boldsymbol{C}_{0_{best}}, \boldsymbol{C}_{1_{best}}, \cdots, \boldsymbol{C}_{n_{best}}\}$
end for
return $\{\boldsymbol{C}^*_{00}, \boldsymbol{C}^*_{01}, \cdots, \boldsymbol{C}^*_{0k}, \boldsymbol{C}^*_{10}, \boldsymbol{C}^*_{11}, \cdots, \boldsymbol{C}^*_{1k}, \cdots, \boldsymbol{C}^*_{M0}, \boldsymbol{C}^*_{M1}, \cdots, \boldsymbol{C}^*_{Mk}\}$

2.2 QuickVina 2-GPU

Figure 2 shows the heterogeneous OpenCL architecture of QuickVina 2-GPU, which is made of a host section (on CPU) and a device section (on GPU). The preparation and post-refinement of the conformations are mostly the responsibility of the host section. The device section concentrates on reducing unnecessary local search by implementing the first-order-consistency-check heuristics and decreasing the number of iterations by scaling up parallelism, to speeding up the most time-consuming Monte-Carlo iterated local search method.

2.2.1 Host section

There are two portions in the host section. The first portion contains four operations that are performed for input of device section and which include the read files, the OpenCL setup, the data preparation and the device memory allocation. In particular, the read files operation is to read the ligand and protein files in .pdbqt format, the box center(indicated by $center_x, center_y, center_z$) and the recommended volume of the docking box (indicated by $size_x, size_y, size_z$).The OpenCL setup operation is to setup the OpenCL environment (platform, device, context, queue, program and kernels). Furthermore, the host part prepares all the required data, the ‘data preparation’ operation,

including grid cache for calculating the energy of a conformation, random maps for generating probability random numbers and random initial conformations for the Monte-Carlo based method to start from. all data is rearranged to load in the GPU memory following the way of access (read-only or read-write). To improve read-write speed on the GPU through efficient memory management, store read-only data, such as grid caches, random maps, and random initial conformations, in the constant device memory; and store data that needs to be read and written, like the best conformation to be returned by the device part, in global device memory. The second portion includes multiple operations after the device section. Finally, all the best conformations found by work items are returned to the CPU. All the best conformations are clustered and sorted in the container by their docking scores. The best 20 conformations will be concretely refined, rescore, sort, and if two of these conformations have RMSDs less than 1Å, only the conformation with the lower score is retained before the final ligand file is generated.

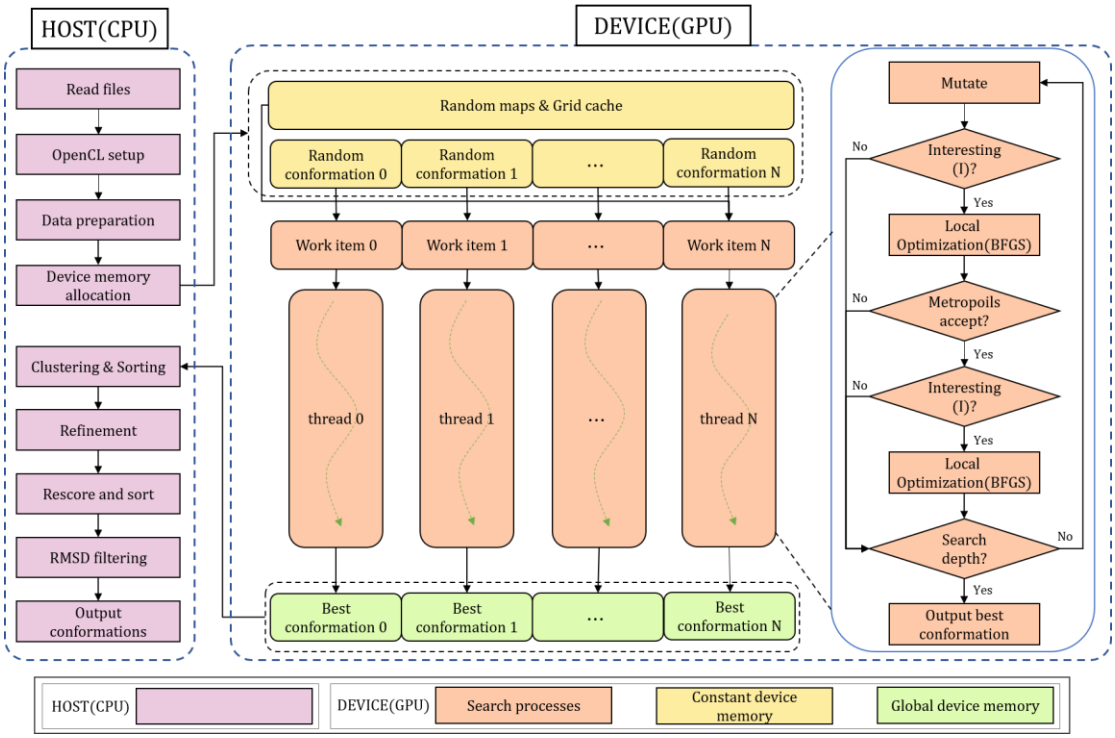

Figure 2. The OpenCL architecture for implementing QuickVina 2-GPU, which consists of a host (CPU) and a device (GPU) section of the execution. The device section implements thousands of docking threads, each of which is assigned with an OpenCL work item to perform a Monte-Carlo based local search method that contains heuristic formulas to reduce unnecessary local searches and the number of search iterations is greatly reduced.

2.2.2 Device part

On the device part, the allocated constant memory (highlighted in yellow) (Figure 2) is assigned for the initialization and the calculation during the reduced-step Monte-Carlo iterated local search processes (highlighted in orange) and the final best conformations are stored in global memory (highlighted in green).

QuickVina 2-GPU enables thousands of reduced-steps iterated local search processes running concurrently within the GPU computational cores. We denote each reduced-step iterated local search process as a docking thread. Within $thread = i$, an OpenCL work item is assigned to a randomly initialized conformation $\boldsymbol{C}_i$, which can be represented by its position, orientation and torsion (POT):

$$\boldsymbol{C}_i = \{x^i,\ y^i,\ z^i, a^i, b^i,\ c^i,\ d^i, \psi_1^i, \psi_2^i, \ldots, \psi_{N_{rot}}^i\} \tag{10}$$

where $x^i,\ y^i,\ z^i$correspond to the conformation position in a pre-determined searching space; $a^i, b^i,\ c^i,\ d^i$ denote its orientation as a rigid body in the quaternion form; $\psi_1^i, \psi_2^i, \ldots, \psi_{N_{rot}}^i$ represent the torsions of N_{rot} rotatable bonds. Then, C_i' can be randomly mutated on one POT element with the uniform distribution as a new conformation $\boldsymbol{C}_i'$:

$$\boldsymbol{C}_i' = R(\boldsymbol{C}_i) \tag{11}$$

where $R(\cdot)$ is a random jitter function to perturb the conformation. The scoring function which quantifies the potential energy of the binding pose continually assesses the conformation. Usually, the potential energy e is computed by adding intermolecular energy and intramolecular energy:

$$e = e_{inter} + e_{intra} \tag{12}$$

The interaction energy of the paired atoms inside the ligand is defined by the e_{intra}, while the interaction energy between the ligand and the receptor is denoted by the e_{inter}. Trilinear interpolation is used to calculate the e_{inter} by looking up the grid cache. The scoring function **SF** can be calculated by the energy calculation function $f(\cdot)$:

$$\boldsymbol{SF}_{\boldsymbol{C}_i'} = f(\boldsymbol{C}_i') \tag{13}$$

QuickVina 2-GPU restricts the application of local search to those docked conformation candidates deemed to be significant by the first-order-necessary-conditon heuristics and $i^{th}\ thread$ uses the conformation of other threads $\overline{\boldsymbol{C}_j}(j = 1,2,\cdots,N, i \neq j)$. The conformation $\boldsymbol{C}_i'$ is deemed as significant for local search if there exists a conformation $\overline{\boldsymbol{C}_j}$ among its $2N$ nearest neighbors such that with respect to each design variable,

$$sign\left\{\frac{\alpha \boldsymbol{SF}_{\boldsymbol{C}}}{\partial \boldsymbol{C}_i}|_{\boldsymbol{c}=\boldsymbol{C}_i'}\right\} \cdot sign\left\{\frac{\alpha \boldsymbol{SF}_{\boldsymbol{C}}}{\partial \boldsymbol{C}_i}|_{\boldsymbol{c}=\overline{\boldsymbol{C}_j}}\right\} \leq 0 \tag{14}$$

Where $\frac{\alpha \boldsymbol{SF}_{\boldsymbol{C}}}{\partial \boldsymbol{C}_i}|_{\boldsymbol{c}=m}$ is the partial derivative of the scoring function $\boldsymbol{SF}$ with respect to the design variable $\boldsymbol{C}_i$ at point m. $\text{sign}(\bullet)$ is sign function. If $\boldsymbol{C}_i'$ fails (14) with respect to the design variable $\boldsymbol{C}_i$, $\boldsymbol{C}_i'$ is still significant for local search if it passes the following test.

$$sign\left\{\frac{\alpha \boldsymbol{SF}_{\boldsymbol{C}}}{\partial \boldsymbol{C}_i}|_{\boldsymbol{c}=\boldsymbol{C}_i'}\right\} \cdot sign\left\{\left[\boldsymbol{SF}_{\boldsymbol{C}_i'} - \boldsymbol{SF}_{\overline{\boldsymbol{C}_j}}\right]\left[(\boldsymbol{C}_i')_i - (\overline{\boldsymbol{C}_j})_i\right]\right\} \leq 0 \tag{15}$$

If $\boldsymbol{C}_i'$'s derivative with respect to $\boldsymbol{C}_i$ is positive and $\boldsymbol{SF}_{\overline{\boldsymbol{C}_j}}$ is higher (or lower) than $\boldsymbol{SF}_{\boldsymbol{C}_i'}$ while $(\overline{\boldsymbol{C}_j})_i$ is to the left(or right) of $(\boldsymbol{C}_i')_i$, then there must be a stationary point between $(\overline{\boldsymbol{C}_j})_i$ and $(\boldsymbol{C}_i')_i$. Reversed relation between the score $\boldsymbol{SF}_{\boldsymbol{C}_i'}$ and $\boldsymbol{SF}_{\overline{\boldsymbol{C}_j}}$ applies when $\boldsymbol{C}_i'$'s derivative with respect to $\boldsymbol{C}_i$ is negative. A Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization is applied to update the ligand conformation by minimizing of the scoring function SF.

First initialize the hessian matrix$\boldsymbol{B}_0 \in \mathbb{R}^{(7+N_{rot})\times(7+N_{rot})}$. $\boldsymbol{B}_0$ is initiated with identity matrix $\boldsymbol{E}$, Initial random conformation $\boldsymbol{x}_i = \boldsymbol{C}_i'$, the $(k+1)^{th}$ iteration of the BFGS is calculated as follows:

Calculating direction $\boldsymbol{d}_k$:

$$\boldsymbol{d}_k = -\boldsymbol{B}_k^{-1} \nabla \boldsymbol{SF}(\boldsymbol{x}_k) \tag{16}$$

Where $\boldsymbol{B}_k^{-1}$ is the inverse matrix of the Hessian matrix generated by the k^{th} iteration

process. Calculating the step size in the direction $\boldsymbol{d}_k$ by using the Armijo criterion：

$$\alpha_k = argmin\ SF(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \tag{17}$$

Updating the conformation $\boldsymbol{x}_{k+1}$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

Updating the conformation $\boldsymbol{B}_{k+1}$:

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k^T}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} \tag{18}$$

Where $\boldsymbol{y}_k = \nabla \boldsymbol{SF}(\boldsymbol{x}_{k+1}) - \nabla \boldsymbol{SF}(\boldsymbol{x}_k)$, $\boldsymbol{s}_k = \alpha_k \boldsymbol{d}_k$. The BFGS iteration process ends if the following conditions are met:

$$\|\nabla \boldsymbol{SF}(\boldsymbol{x}_{k+1})\| \le \varepsilon \ or\ k = search\_depth \tag{19}$$

Where $0 < \varepsilon \le 1$, $\alpha(\alpha_k)$ means the step size in the direction $\boldsymbol{d}_k$, along the decrease of the SF value, the conformation of the BFGS output is $\boldsymbol{x}_{k+1}$. Next, a metropolis acceptance criterion is adopted to decide whether to accept the optimized conformation or not, by comparing the energy e_0 before the mutation and the energy e_{opt} after the optimization. Here, the accept probability P is represented by:

$$P = \begin{cases} 1 & e_0 > e_{opt} \\ \frac{\exp(e_0 - e_{opt})}{1.2} & e_0 \le e_{opt} \end{cases} \tag{20}$$

It indicates that the accepted conformation is more likely to have a lower energy. Once accepted, the conformation will be further evaluated and optimized by BFGS. Then, the next iteration continues to update the previous optimized conformations until convergence. Finally, all the best conformations found by work items are returned to the host part.

Algorithm 2 proposed the pseudocode of our QuickVina 2-GPU. In Algorithm 2, $Mutute(.)$ means a random mutation of the POT in a ligand conformation; $I\_Check(.)$ determines whether the current conformation requires further local search which is described in Equations (14) and (15);$BFGS(.)$ represents the BFGS optimization method which is described in Equations (16)–(19); $Scoring(.)$ is the potential energy of a binding pose described in Equations (10) and (13); $Metropolis(.)$ is the metropolis acceptance criterion described in Equation (20); and $Clustering\&Sorting\&refinment\&RMSD\_filtering$ (.) is the aggregation, filtering (based on the RMSD) and reordering (based on the docking score) of all ligand conformations among all threads.

Algorithm 2 QuickVina 2-GPU

Input: random ligand conformations: $\{\boldsymbol{C}_0, \boldsymbol{C}_1, \cdots, \boldsymbol{C}_M\}$

Output: top k ligand conformations: $\{\boldsymbol{C}_0^*, \boldsymbol{C}_1^*, \cdots, \boldsymbol{C}_k^*\}$

```
for all C_thread(thread = 0,1,2,⋯,n) concurrently do
    for all search_depth = 0,1,2,⋯,s do
        Mutate(C_thread)
        if I_Check(C_thread) then
            BFGS(C_thread)
            if Metropolis(C_thread, C_thread_best) then
                C_thread_best = C_thread
                if I_Check(C_thread_best) then
                    BFGS(C_thread_best)
                end if
            end if
```

12: **end if**

13: **end for**

14: **end for**

15: $Clustering\&Sorting\&refinment\&RMSD\_filtering(\boldsymbol{C}_{0_{best}}, \boldsymbol{C}_{1_{best}}, \cdots, \boldsymbol{C}_{n_{best}})$

16: return $\{\boldsymbol{C}_0^*, \boldsymbol{C}_1^*, \cdots, \boldsymbol{C}_k^*\} \in \{\boldsymbol{C}_{0_{best}}, \boldsymbol{C}_{1_{best}}, \cdots, \boldsymbol{C}_{n_{best}}\}$

2.3 QuickVina-W-GPU

The heterogeneous OpenCL implementation of QuickVina-W is depicted in Figure 3, which consists of a host part (on CPU) and a device part (on GPU). The host part is mainly in charge of the preparation and post-refinement of the conformations. The device part focuses on the acceleration of the most time-consuming Monte-Carlo iterated local search method by enlarging the scale of parallelism as well as reducing the number of iterations.

2.3.1 Host part

The host part consists of two sections (see Figure 3). The first section includes four operations, which are the read files, the OpenCL setup, the data preparation and the device memory allocation, and all operations are implemented for the input to the device part. Specifically, the read files operation is to read the ligand and protein files in .pdbqt format, the box center(indicated by $center_x, center_y, center_z$) and the recommended volume of the docking box (indicated by $size_x, size_y, size_z$). The OpenCL setup operation is to setup the OpenCL environment (platform, device, context, queue, program and kernels). Furthermore, the host part prepares all the required data, including grid cache (for calculating the energy of a conformation), random maps (for generating probability random numbers) and random initial conformations (for Monte-Carlo based method to start from). The data is then re-organized to load in the device memory according to how it is accessed (read-only or read-write). The read-only grid cache, random maps and random conformations are allocated in the constant device memory while the read-write of best conformations returned by the device part is allocated in the global device memory. Such kind of memory management could efficiently boost the speed of reading and writing on GPU. The second section includes multiple operations after the device part. Finally, all the best conformations found by work items are returned to the CPU. All the best conformations are clustered and sorted in the container by their docking scores. The best 20 conformations will be concretely refined, rescore, sort, and if two of these conformations have RMSDs less than 1Å, only the conformation with the lower score is retained before the final ligand file is generated.

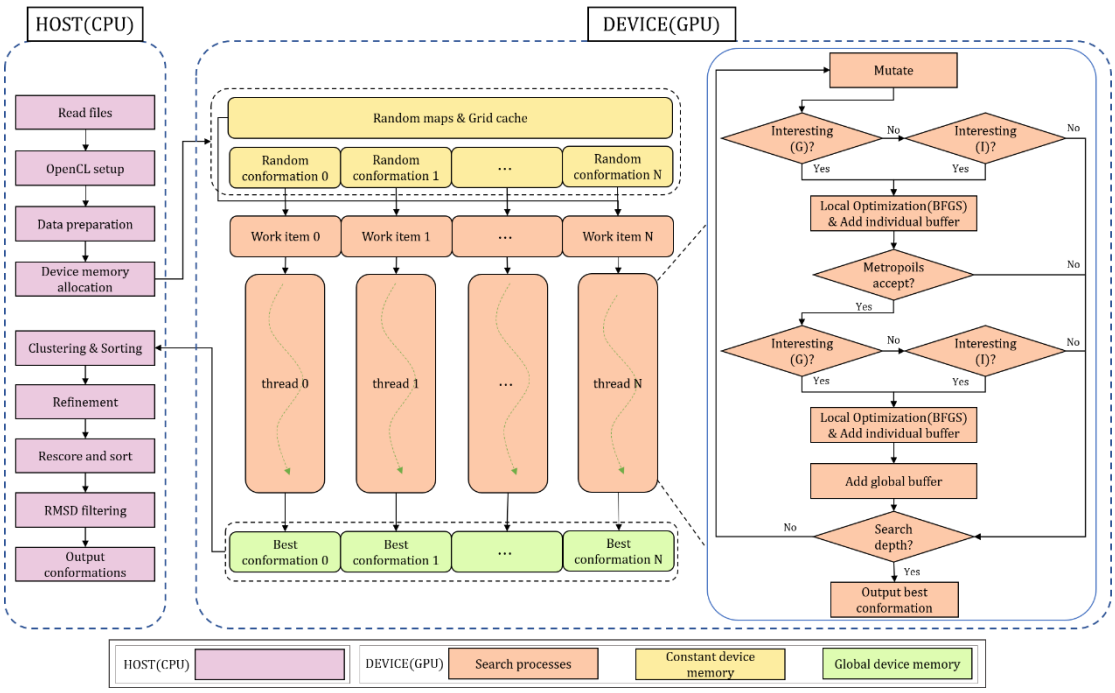

Figure 3. The OpenCL architecture for implementing QuickVina-W-GPU, which consists of a host (CPU) and a device (GPU) part of execution. The device part implements thousands of docking threads, each of which is assigned with an OpenCL work item to perform a Monte-Carlo based local search method that contains I Check and G Check to reduce unnecessary local searches and the number of search iterations is greatly reduced.

2.3.2 Device part

On the device part, the allocated constant memory (highlighted in yellow)(Figure 3) is assigned for the initialization and the calculation during the reduced-step Monte-Carlo iterated local search processes (highlighted in orange), the final best conformations are stored in global memory (highlighted in green), an OpenCL work item add a global buffer and individual buffer for every thread. The best conformation output in all threads is stored in the global buffer according to their three-dimensional position, the global buffer is implemented as an octree (octal tree) of history points. The octree root is a cell that spans over the whole search space; and the history points are distributed in the octree according to their spatial distribution in the three dimensions. Having history from other threads allows us to make use of other threads experience and make decisions in already explored energy landscape areas, while having history from an individual same thread allows us to make decisions in virgin areas. The application of the octree in a multithreaded collaborative Monte Carlo algorithm is to find its surrounding neighborhood conformations faster. Octrees are implemented recursively whereas the OpenCL standard cannot support any recursion in kernels because the allocation of stack space for thousands of threads is too expensive. Besides, the memory space for an octree is dynamically allocated while the memory size must first be fixed in the GPU, so the structure of an octree is not suitable for the OpenCL implementation. Therefore, we construct the structure of global memory in the GPU, where global memory is accessible by every thread.

The GPU is first allocated memory based on the number of threads and the size of the search step. Then we determine a binary code for each conformation according to three-dimensional position of the molecular conformation, and finally store it in the corresponding global memory according to the binary code. The molecular conformation in the Global Buffer corresponding to

the binary code is then the nearest neighbors.

First determine the number of bits encoded ($3n-3$) and the memory size based on the number of threads($thread$) and the number of search steps($search\_depth$), the bits encoded need to satisfy the following conditions:

$$2^{3n-1} < (thread * search\_steps) < 2^{3n} \quad (21)$$

Then coded by comparing the position of the conformation (x, y, z) with the central of the docking box $(center\_x, center\_y, center\_z)$ based on Figure 4(d), if $x > center\_x$, it is coded as "1" otherwise it is encoded as "0", after compiling the 3bit, you can determine which small box the conformation is in. Continue dividing the current box and calculate the centroid of box ($center\_x \pm \frac{size\_x}{4^a}, center\_y \pm \frac{size\_y}{4^a}, center\_z \pm \frac{size\_z}{4^a}$), where $a \in [1, n-1]$, a means that the box needs to be divided $(n-2)$ times more. Finally, the 3-bit codes of this (n-1) group are combined into a (3n-3)-bit code and the conformation is stored in the corresponding encoding location of the global buffer. For example, if thread and search_depth of QuickVina-W-GPU were set to 100 and 5, respectively. The maximum number of points to be stored in the Global Buffer is $5 * 100$, resulting in $n = 3$. Each conformation is given a Binary ID based on the three-dimensional location. In Figure 4(a), the large purple square indicates a docking box, dividing the box into eight red boxes like Figure 4(d), coded it as "111" based on the results of comparing the position of the conformation (x, y, z) with the central of the docking box $(center\_x, center\_y, center\_z)$, due to $(n-2) = 1$, the red box needs to divided one time more, coded it as "100" based on the position of black point. The result of combining 3bit codes of two group is "111100". The conformation will be placed in the memory space of "data0" at index "111100".

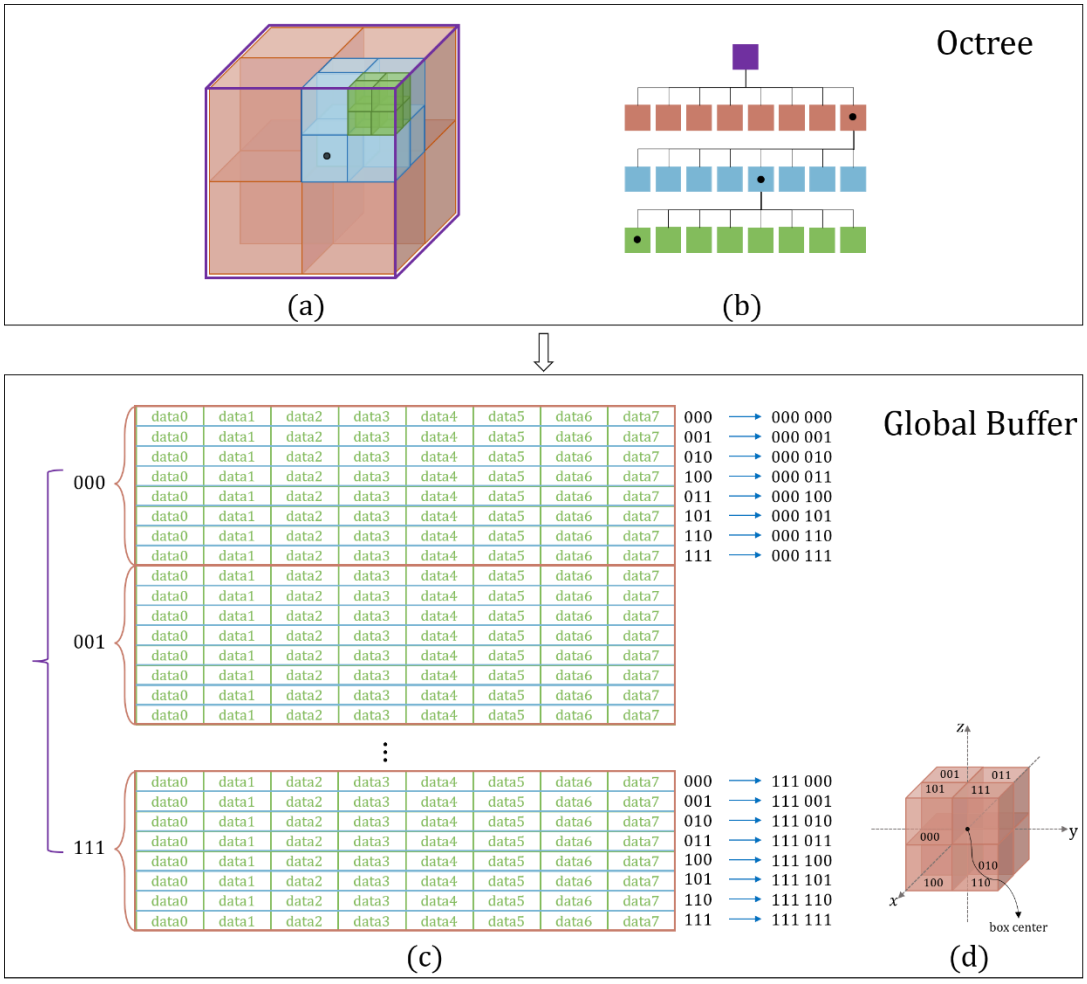

Figure 4. Transformation of the original octree structure into the global buffer format. For example, the black dot in the green box in Figure(b) will be encoded as "11100" as shown in Figure (d), and then placed in the corresponding position in the global buffer according to the encoding result.

QuickVina-W-GPU enables thousands of reduced-steps iterated local search processes running concurrently within the GPU computational cores. We denote each reduced-step iterated local search process as a docking thread. Within thread = i, an OpenCL work item is assigned to a randomly initialized co initialized conformation $\boldsymbol{C}_i$, which can be represented by its position, orientation and torsion (POT):

$$\boldsymbol{C}_i = \{x^i,\ y^i,\ z^i, a^i, b^i,\ c^i,\ d^i, \psi_1^i, \psi_2^i, \ldots, \psi_{N_{rot}}^i\} \tag{22}$$

where $x^i,\ y^i,\ z^i$correspond to the position of the conformation in a pre-determined searching space; $a^i, b^i,\ c^i,\ d^i$ denote its orientation as a rigid body in the quaternion form; $\psi_1^i, \psi_2^i, \ldots, \psi_{N_{rot}}^i$ represent torsions of N_{rot} rotatable bonds. Then, $\boldsymbol{C}_i'$ is the randomly mutated that each conformation $\boldsymbol{C}_i$ is to be in one of its POT with the uniform distribution.

$$\boldsymbol{C}_i' = R(\boldsymbol{C}_i) \tag{23}$$

where $R(\cdot)$ is a random jitter function to perturb the conformation. G-check and I-check check whether $\boldsymbol{C}_i'$ is significant or not for the thread. G-check is first performed to calculate the spatial distance between the perturbed molecular conformation and the conformation in the global memory, which stores the final conformation after all threads have been optimized twice by the BFGS, is calculated. The conformations with spatial distances less than the cut-off radius R are then filtered out, and then the Euclidean distance in N dimensions between the conformation to be detected and

the conformation within the cut-off radius R is calculated, and the first N nearest-neighbour conformations are found in order of Euclidean distance from closest to farthest, and if one of the N neighbour conformations satisfies the heuristic condition, otherwise I Check is performed. Search the molecular conformations in the thread's local cache, which stores all the conformations obtained during the thread's BFGS optimization. The N-dimensional Euclidean distance between the perturbed molecular conformation and the conformation in the local cache is calculated, and the top 3N nearest-neighbour conformations are found by sorting the N-dimensional Euclidean distances from closest to farthest. If one of the 3N neighbour point conformations satisfies the heuristic condition described in Equation (14) and (15), proceed to BFGS. If not, exit that search and start again at mutating.

The conformation will be continuously evaluated with a scoring function that quantifies the potential energy of the binding pose. Generally, the potential energy e is calculated with the sum of intermolecular energy and intramolecular energy:

$$e = e_{inter} + e_{intra} \tag{24}$$

where e_{inter} represents the interaction energy between the ligand and the receptor, and it is calculated using trilinear interpolation that approximates the energy of each atom pair by looking up the grid cache; and e_{intra} indicates the interaction energy of the pairwise atoms within the ligand. Considering that both e_{inter} and e_{intra} are related to the binding pose, the scoring function SF can be denoted as a function of POT variables:

$$\boldsymbol{SF}_{\boldsymbol{C}_i'} = f(\boldsymbol{C}_i') \tag{25}$$

A Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization is applied to update the ligand conformation by minimizing of the scoring function SF. First initialize the hessian matrix $\boldsymbol{B}_0 \in \mathbb{R}^{(7+N_{rot})\times(7+N_{rot})}$. $\boldsymbol{B}_0$ is initiated with identity matrix $\boldsymbol{E}$, Initial random conformation $\boldsymbol{x}_i = \boldsymbol{C}_i'$, the $(k+1)^{th}$ iteration of the BFGS is calculated as follows:

Calculating direction $\boldsymbol{d}_k$:

$$\boldsymbol{d}_k = -\boldsymbol{B}_k^{-1}\nabla \boldsymbol{SF}(\boldsymbol{x}_k) \tag{26}$$

Where $\boldsymbol{B}_k^{-1}$ is the inverse matrix of the Hessian matrix generated by the k^{th} iteration process. Calculating the step size in the direction $\boldsymbol{d}_k$ by using the Armijo criterion:

$$\alpha_k = argmin\ SF(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \tag{27}$$

Updating the conformation $\boldsymbol{x}_{k+1}$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

Updating the conformation $\boldsymbol{B}_{k+1}$:

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k^T}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} \tag{28}$$

Where $\boldsymbol{y}_k = \nabla \boldsymbol{SF}(\boldsymbol{x}_{k+1}) - \nabla \boldsymbol{SF}(\boldsymbol{x}_k)$, $\boldsymbol{s}_k = \alpha_k \boldsymbol{d}_k$. The BFGS iteration process ends if the following conditions are met:

$$\|\nabla \boldsymbol{SF}(\boldsymbol{x}_{k+1})\| \le \varepsilon \ or\ k = search\_depth \tag{29}$$

Where $0 < \varepsilon \le 1$, $\alpha(\alpha_k)$ means the step size in the direction $\boldsymbol{d}_k$, along the decrease of the SF value, the conformation of the BFGS output is $\boldsymbol{x}_{k+1}$. Next, a metropolis acceptance criterion is adopted to decide whether to accept the optimized conformation or not, by comparing the energy e_0 before the mutation and the energy e_{opt} after the optimization. Here, the accept probability P is represented by:

$$P = \begin{cases} 1 & e_0 > e_{opt} \\ \frac{\exp(e_0 - e_{opt})}{1.2} & e_0 \le e_{opt} \end{cases} \tag{30}$$

It indicates that the accepted conformation is more likely to have a lower energy. Once accepted, the conformation will be further evaluated and optimized by BFGS. Then, the next iteration continues to update the previous optimized conformations until convergence. Finally, all the best conformations found by work items are returned to the host part.

Algorithm 3 proposed the pseudocode of our QuickVina-W-GPU. In Algorithm 2, Mutute(.) means a random mutation of the POT in a ligand conformation; $I\_Check(.)$ and $G\_Check(.)$ determines whether the current conformation requires further local search which is described in Equations (14) and (15);BFGS(.) represents the BFGS optimization method which is described in Equations (26)–(29); Scoring(.) is the potential energy of a binding pose described in Equations (24) and (25); Metropolis(.) is the metropolis acceptance criterion described in Equation (30); and Clustering&Sorting&refinment&RMSD_filtering (.) is the aggregation, filtering (based on the RMSD) and reordering (based on the docking score) of all ligand conformations among all threads.

Algorithm 3 QuickVina-W-GPU

Input: random ligand conformations: $\{\boldsymbol{C}_0, \boldsymbol{C}_1, \cdots, \boldsymbol{C}_M\}$

Output: top $\boldsymbol{k}$ ligand conformations: $\{\boldsymbol{C}_0^*, \boldsymbol{C}_1^*, \cdots, \boldsymbol{C}_k^*\}$

for all $\boldsymbol{C}_{thread}(thread = 0,1,2,\cdots,n)$ **concurrently** ***do***
  for all $search\_depth = 0,1,2,\cdots,s$ **do**
    $Mutate(\boldsymbol{C}_{thread})$
    if $G\_Check(\boldsymbol{C}_{thread}) || I\_Check(\boldsymbol{C}_{thread})$ **then**
      $BFGS(\boldsymbol{C}_{thread})$
      $individual\_buffer.pushback(\boldsymbol{C}_{thread})$
      if $Metropolis(\boldsymbol{C}_{thread}, \boldsymbol{C}_{thread_{best}})$ **then**
        $\boldsymbol{C}_{thread_{best}} = \boldsymbol{C}_{thread}$
        if $G\_Check(\boldsymbol{C}_{thread_{best}}) || I\_Check(\boldsymbol{C}_{thread_{best}})$ **then**
          $BFGS(\boldsymbol{C}_{thread_{best}})$
          $global\_buffer.pushback(\boldsymbol{C}_{thread_{best}})$
        end if
      end if
    end if
  end for
end for
$Clustering\&Sorting\&refinment\&RMSD\_filtering(\boldsymbol{C}_{0_{best}}, \boldsymbol{C}_{1_{best}}, \cdots, \boldsymbol{C}_{n_{best}})$
return $\{\boldsymbol{C}_0^*, \boldsymbol{C}_1^*, \cdots, \boldsymbol{C}_k^*\} \in \{\boldsymbol{C}_{0_{best}}, \boldsymbol{C}_{1_{best}}, \cdots, \boldsymbol{C}_{n_{best}}\}$

3. Results and Discussion

3.1 Experimental Settings

All experiments were executed on the same computer with Intel (R) Core (TM) i9-10900K CPU @ 3.7 GHz using Windows 10 Operating System with 64 GB RAM and OpenCL v.3.0 with Nvidia Geforce RTX 3090 GPU under single-precision floating-point format (FP32). The center and volume of the docking box, the number of random seeds (cpu or thread) and the runtime options (exhaustiveness or $search\_depth$) were used to configure these experiments. We created a config.txt file for complexes, which contains the parameters and their definitions as shown in Table I. Our experimental parameters such as cpu, thread, exhaustiveness, and $search\_depth$ are set as shown in Table 2. For instance, the argument exhaustiveness and cpu were set to 128 and 20 in the AutoDock Vina, while the argument $search\_depth$ was set by the heuristic formula and thread were set to 8000

in the Vina-GPU.

Table 1. Parameters are included in the config.txt file

Argument	Description
receptor	the recrptor file (in .pdbqt format)
ligand/ ligand_directory	the ligand file (in .pdbqt format)/ this path contains all the ligand files
center_x/y/z	the center of searching box in the receptor
size_x/y/z	the volume of the searching box
cpu/thread	the number of random seeds
exhaustiveness/search_depth	the size of searching iterations

Table 2. Experimental parameters setting

Docking method	the number of random seeds	the runtime options
AutoDock Vina	$cpu = 20$	$exhaustiveness$ =128
Vina-GPU	$thread$ =8000	$search\_depth$ was set by the heuristic formula
QuickVina 2	$cpu = 8$	$exhaustiveness = 8$
QuickVina-W	$cpu = 16$	$exhaustiveness$ =16
Vina-GPU+	$thread$ =8000	$search\_depth\ was\ set\ by\ the\ heuristic\ formula$
QuickVina 2-GPU	$thread = 5000$	$search\_depth$ =1
QuickVina-W-GPU	$thread = 8000$	$search\_depth$ =5

For Vina-GPU and Vina-GPU+, the size of searching iterations in each thread ($search\_depth$) was set by the heuristic formula. The heuristic formula is given as follows,

$$search\_depth = max\ (1, floor(0.24 * N_{atom} + 0.29 * N_{rot} - 3.41)) \quad (31)$$

where N_{atom} is the number of atoms and N_{rot} is the number of rotatable bonds in a ligand. The function $floor(n)$ is the function that gives as output the greatest integer less than or equal to n when takes as input a real number n.

The docking accuracy is determined by their docking score and RMSD. The runtime acceleration (Acc) of our proposed method against the baseline method is defined by

$$Acc = \frac{t_{\text{baseline}}}{t_{\text{our}}} \quad (32)$$

We evaluated the similarity of top i compounds with the lowest docking scores on the baseline methods or our Vina-GPU 2.0 by Jaccard index[22] as defined by

$$J_i = \frac{\left|T^i_{baseline} \cap T^i_{our}\right|}{\left|T^i_{baseline} \cup T^i_{our}\right|} \quad (33)$$

where $i = 15, 50, 100, 200, 300$, and $T^i_{baseline}$ and T^i_{our} represent subset of top i compounds of the baseline methods and Vina-GPU 2.0, respectively.

3.2 Virtual Screening on RIPK1

To show the acceleration effect of our Vina-GPU 2.0 in implementing real virtual screens, a case was detailed on the receptor RIPK1 with the docking of DrugBank. The receptor RIPK1 is a key regulator of apoptosis and necrosis as well as inflammatory pathways, and it has emerged as one of the effective targets for the treatment of various diseases such as neurodegenerative diseases,

autoimmune diseases and inflammation. DrugBank[23] is one of the most popular drug databases that contains detailed information on drugs and drug targets. A total of 9125 molecules were downloaded from the DrugBank database at https://go.drugbank.com/releases/latest#structures.

Table 3. Comparison of docking runtime on RIPK1

Baseline method	Time (hours)	Vina-GPU 2.0	Time (hours)	Acc
AutoDock Vina	176.66	Vina-GPU+	2.50	70.67X
Vina-GPU	8.18	Vina-GPU+	2.50	3.27X
QuickVina 2	8.62	QuickVina 2-GPU	5.91	1.46X
QuickVina-W	26.37	QuickVina-W-GPU	8.16	3.23X

Acc: the runtime acceleration of Vina-GPU 2.0 against the baseline method.

Vina-GPU+ took only 2.50 hours to complete the whole docking process while AutoDock Vina took 176.66 hours and Vina-GPU took 8.18 hours, demonstrating that our Vina-GPU+ achieved acceleration of 70.67X and 3.27X. In addition, our QuickVina 2-GPU and QuickVina-W-GPU obtained acceleration of 1.46X and 3.23X, respectively.Among all the methods, the least time-consuming methods are Vina-GPU+ (2.5h), QuickVina 2-GPU (5.91h), QuickVina-W-GPU (8.16h) and Vina-GPU (8.18h) respectively (Table 3). Figure 5 shows the comparison of docking scores between our Vina-GPU 2.0 and the baseline methods. The color bar represents the number of atoms in a ligand. Based on the docking scores of the complexes, most complexes are distributed near the diagonal and fall into a lavender margin of 0.5 kcal/mol difference. Their Pearson correlation coefficients are 0.912, 0.953, 0.802 and 0.915 for AutoDock Vina VS Vina-GPU+, Vina-GPU VS Vina-GPU+, QuickVina 2 VS QuickVina 2-GPU, and QuickVina-W VS QuickVina-W-GPU, respectively. Except for QuickVina 2-GPU, our Vina-GPU 2.0 achieves close docking scores with the baseline methods. QuickVina 2 lost its docking accuracy to obtain docking conformations quickly, and the docking results returned each time were not stable, thus resulting in the docking structures derived from QuickVina 2 and QuickVina 2-GPU are not very similar. In addition, the docking scores of all 9125 molecules for our Vina-GPU 2.0 and baseline methods were shown in Supplementary Data S1. For example, the average docking score of AutoDock Vina and Vina-GPU+ are -6.16 and -5.98, respectively. For AutoDock Vina, the top three docking scores are -10, -9.4, -9.3 and their corresponding binding poses were DB12983, DB09187, and DB15997, shown by DrugBank accession number. For Vina-GPU+, they are DB12983, DB09187, DB13136 with the best docking scores (-10, -9.5, -9.3). Table 4 shows that all the Jacard indexes of the docking results of our Vina-GPU 2.0 and the baseline methods on RIPK1.

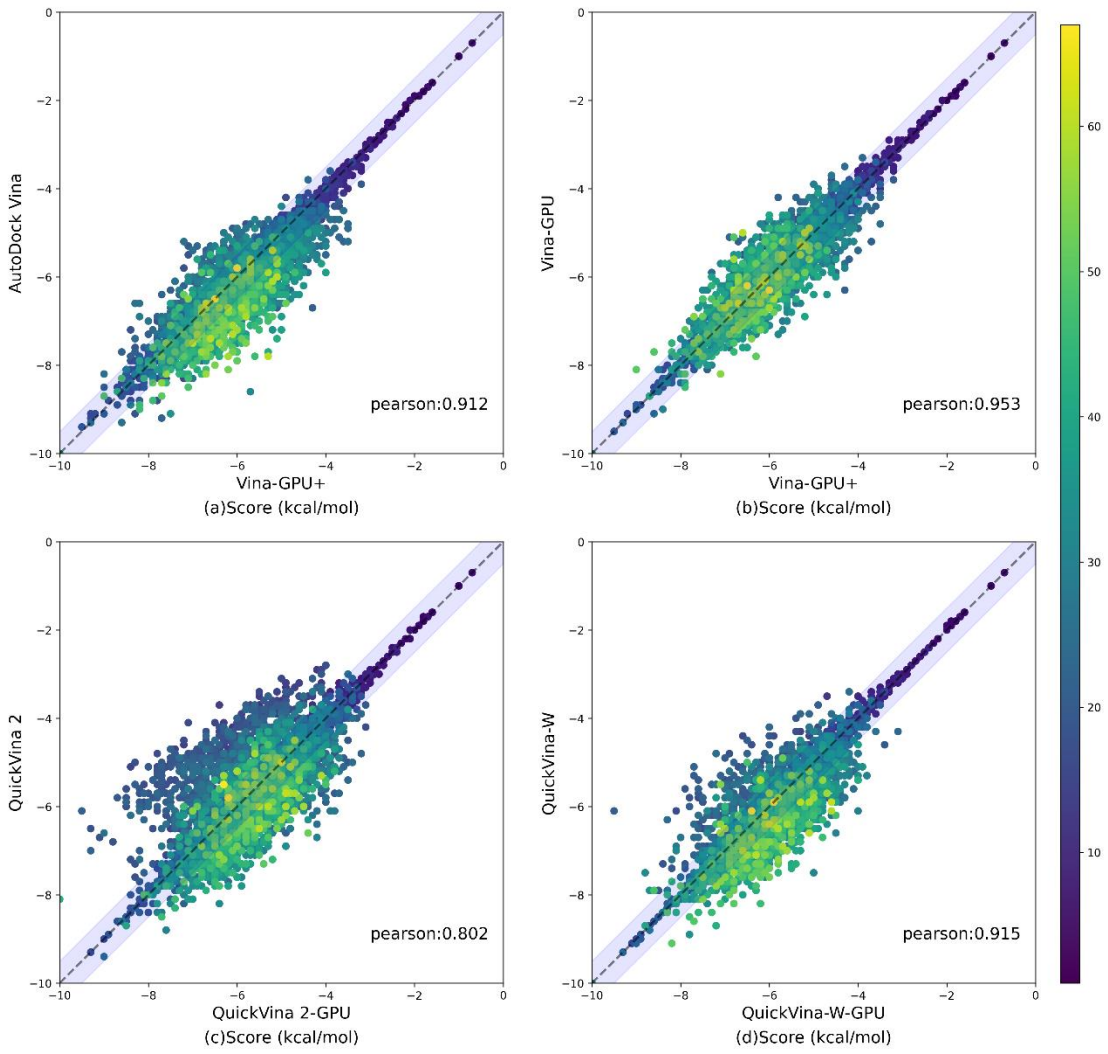

Figure 5. Comparable docking scores on RIPK1 between Vina-GPU 2.0 and the baseline methods on all 9125 compounds from the DrugBank dataset. The color bar encodes the number of atoms in one ligand. A margin of 0.5 kcal/mol difference on the docking score between Vina-GPU 2.0 and the baseline methods is highlighted with lavender. The Pearson correlation coefficient of their docking scores is 0.912,0.953,0.802 and 0.915, respectively (indicated by "pearson").

Table 4. The Jacard indexes J_i on the top i subsets of Baseline method and Vina-GPU 2.0 on RIPK1

(a) AutoDock Vina VS Vina-GPU+

Top i	$T^i_{baseline} \cap T^i_{\text{Vina}-GPU\ 2.0}$	$T^i_{baseline} \cup T^i_{\text{Vina}-GPU\ 2.0}$	Jacard Index
15	9	21	0.429
50	33	67	0.493
100	69	131	0.526
200	129	271	0.476
300	205	395	0.519

(b) Vina-GPU VS Vina-GPU+

Top i	$T^i_{baseline} \cap T^i_{\text{Vina}-\text{GPU}\ 2.0}$	$T^i_{baseline} \cup T^i_{\text{Vina}-\text{GPU}\ 2.0}$	Jacard Index
15	13	17	0.765
50	44	56	0.786

100	91	109	0.834
200	175	225	0.778
300	266	334	0.796

(c) QuickVina 2 VS QuickVina 2-GPU

Top i	$T^i_{baseline} \cap T^i_{\text{Vina-GPU 2.0}}$	$T^i_{baseline} \cup T^i_{\text{Vina-GPU 2.0}}$	Jacard Index
15	7	23	0.304
50	29	71	0.408
100	60	140	0.429
200	128	272	0.471
300	190	410	0.463

(d) QuickVina-W VS QuickVina-W-GPU

Top i	$T^i_{baseline} \cap T^i_{\text{Vina-GPU 2.0}}$	$T^i_{baseline} \cup T^i_{\text{Vina-GPU 2.0}}$	Jacard Index
15	11	19	0.579
50	41	59	0.695
100	84	116	0.724
200	166	234	0.709
300	233	367	0.635

3.3 Virtual Screening on RIPK3

The receptor RIPK3 is also a key regulator of apoptosis and necrosis as well as inflammatory pathways, and it has become one of the effective targets for the treatment of various diseases such as neurodegenerative diseases, autoimmune diseases and inflammation. Therefore, in this paper, another case was detailed on the receptor RIPK3 (PDBid) with the docking of DrugBank. Table 2 shows a detailed comparison of the docking times of our Vina-GPU 2.0 and the baseline methods.

Only 2.55 hours were taken to execute the whole docking process by Vina-GPU+ while 154.38 hours by AutoDock Vina and 8.87 hours by Vina-GPU, indicating that the acceleration of 60.54X and 3.48X are achieved by our Vina-GPU+ (Table 5). Also, the acceleration of 1.41X and 3.97X was obtained by our QuickVina 2-GPU and QuickVina-W-GPU, respectively. Among all the methods, the least time-consuming methods are Vina-GPU+ (2.55h), QuickVina 2-GPU (6.60h), QuickVina-W-GPU (8.81h) and Vina-GPU (8.87h) respectively (Table 3).

Figure 6 shows the comparison of docking scores between our Vina-GPU2.0 and the baseline methods, where most compounds lie around the diagonal line and within the margin (in lavender) of 0.5 kcal/mol difference on the docking score. The color bar encodes the number of atoms in one ligand. Their Pearson correlation coefficients are 0.976, 0.991, 0.921 and 0.975 for AutoDock Vina VS Vina-GPU+, Vina-GPU VS Vina-GPU+, QuickVina 2 VS QuickVina 2-GPU, and QuickVina-W VS QuickVina-W -GPU, respectively. It shows that our Vina-GPU2.0 achieves the very close docking scores with the baseline methods. the docking scores of all 9125 molecules for our Vina-GPU 2.0 and baseline methods were shown in Supplementary Data S2. For example, the average docking score of AutoDock Vina and Vina-GPU+ are -7.54 and -7.47, respectively. For AutoDock Vina, the top 3 binding poses with the lowest docking scores (-13.3, -13, -13) are DB14773, DB06896, DB11977 (DrugBank accession number). For Vina-GPU+, they are DB14773, DB06896, DB05454 with the best docking scores (-13.7, -13.4, -12.5). Table 6 shows that all the Jacard indexes of the docking results of our Vina-GPU 2.0 and the baseline methods on RIPK3.

Table 5. Comparison of docking runtime on RIPK3

Baseline method	Time(hours)	Vina-GPU 2.0	Time(hours)	Acc
AutoDock Vina	154.38	Vina-GPU+	2.55	60.54X
Vina-GPU	8.87	Vina-GPU+	2.55	3.48X
QuickVina 2	9.31	QuickVina 2-GPU	6.60	1.41X
QuickVina-W	34.94	QuickVina-W-GPU	8.81	3.97X

Acc: the runtime acceleration of Vina-GPU 2.0 against the baseline method.

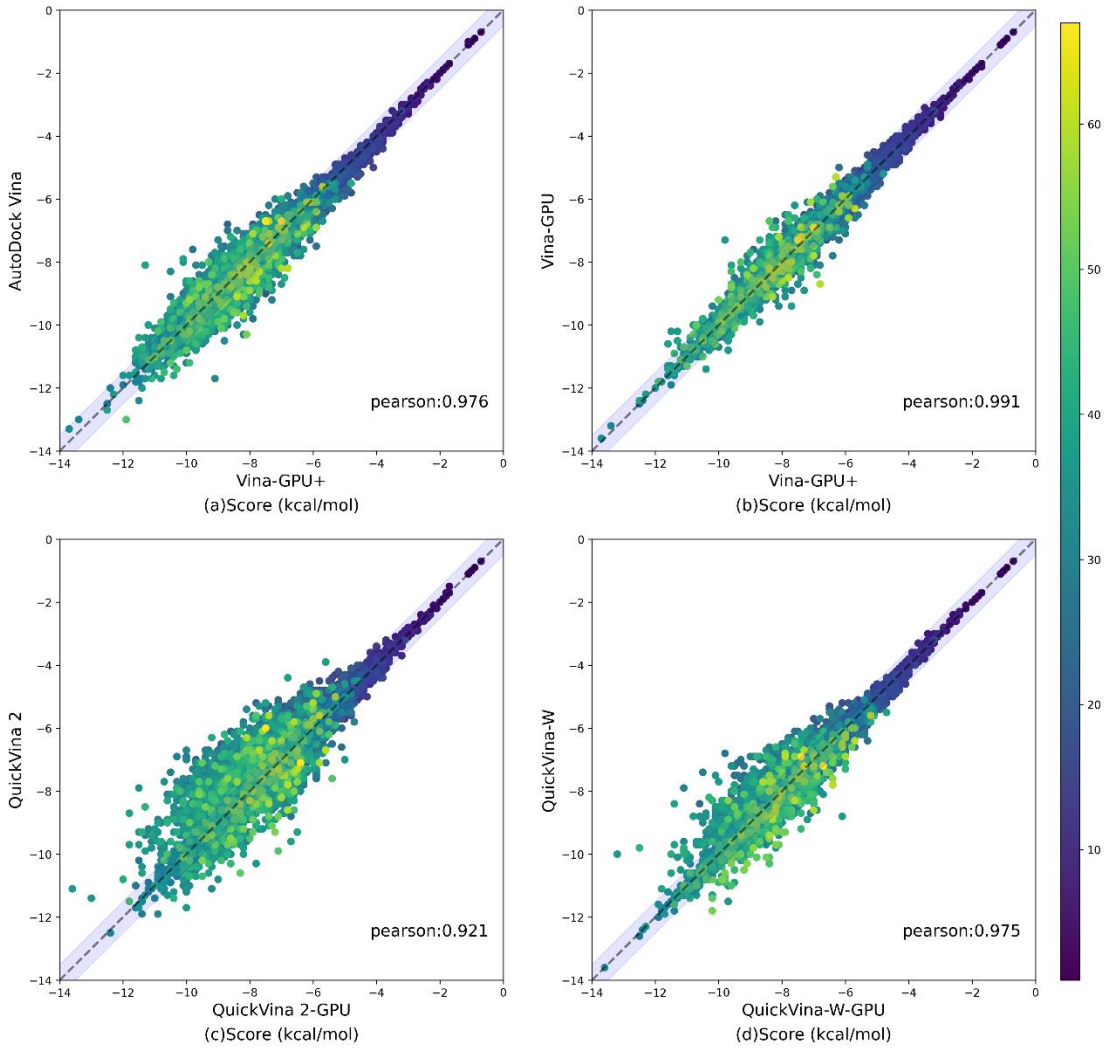

Figure 6. Comparable docking scores on RIPK3 between Vina-GPU 2.0 and the baseline methods on all 9125 compounds from the DrugBank dataset. The color bar encodes the number of atoms in one ligand. A margin of 0.5 kcal/mol difference on the docking score between Vina-GPU 2.0 and the baseline methods is highlighted with lavender. The Pearson correlation coefficient of their docking scores is 0.976,0.991,0.921 and 0.975, respectively (indicated by "pearson").

Table 6. The Jacard indexes J_i on the top i subsets of Baseline method and Vina-GPU 2.0 on RIPK3

(a) AutoDock Vina VS Vina-GPU+

Top i	$T^i_{baseline} \cap T^i_{\text{Vina}-GPU\ 2.0}$	$T^i_{baseline} \cup T^i_{\text{Vina}-GPU\ 2.0}$	Jacard Index
15	9	21	0.429
50	28	72	0.389
100	66	134	0.493
200	147	253	0.581

300	218	382	0.571

(b) Vina-GPU VS Vina-GPU+

Top i	$T^i_{baseline} \cap T^i_{\text{Vina-GPU 2.0}}$	$T^i_{baseline} \cup T^i_{\text{Vina-GPU 2.0}}$	Jacard Index
15	14	16	0.875
50	41	59	0.695
100	87	113	0.770
200	173	227	0.762
300	269	331	0.813

(c) QuickVina 2 VS QuickVina 2-GPU

Top i	$T^i_{baseline} \cap T^i_{\text{Vina-GPU 2.0}}$	$T^i_{baseline} \cup T^i_{\text{Vina-GPU 2.0}}$	Jacard Index
15	6	24	0.25
50	24	76	0.316
100	51	148	0.351
200	111	289	0.384
300	172	428	0.402

(d) QuickVina-W VS QuickVina-W-GPU

Top i	$T^i_{baseline} \cap T^i_{\text{Vina-GPU 2.0}}$	$T^i_{baseline} \cup T^i_{\text{Vina-GPU 2.0}}$	Jacard Index
15	12	18	0.667
50	36	64	0.563
100	77	123	0.626
200	155	245	0.633
300	238	362	0.657

3.4 Docking acceleration on the AutoDock-GPU dataset

The benchmark dataset from the AutoDock-GPU[24] study is comprised of 85 complexes from the Astex Diversity Set[25], 35 complexes from CASF-2013[26], and 20 complexes from the Protein Data Bank[27]. They cover a wide range of ligand complexities and targets properties. Table 5 shows the total docking runtime of our Vina-GPU 2.0 and the baseline methods on all 140 complexes of the AutoDock-GPU dataset. Only 0.25 hours were taken to execute the whole docking process by Vina-GPU+ while 9.15 hours by AutoDock Vina and 0.29 hours by Vina-GPU, indicating that the acceleration of 36.60X and 1.16X are achieved by our Vina-GPU+ (Table 3). Since the molecular docking on the AutoDock-GPU dataset is not a single receptor docking with multiple ligands, Vina-GPU+ generates the same number of grid cache as Vina-GPU, and the 1.17X speedup is due to the fact that the grid cache of Vina-GPU+ is computed in parallel in the GPU. Also, the acceleration of 1.75X and 4.20X was obtained by our QuickVina 2-GPU and QuickVina-W-GPU, respectively. Among all the methods, the least time-consuming methods are QuickVina 2-GPU (0.08h) and QuickVina 2 (0.14h), respectively (Table 7). QuickVina-W-GPU requires inter-thread communication. Compared to AutoDock Vina, Vina-GPU and Vina-GPU+, QuickVina 2-GPU contains heuristics that reduce unnecessary local searches, and therefore, QuickVina 2-GPU docking is the fastest.

Table 7. The total docking runtime on the AutoDock-GPU dataset

Baseline method	Time (hours)	Vina-GPU 2.0	Time (hours)	Acc

AutoDock Vina	9.15	Vina-GPU+	0.25	36.60 X
Vina-GPU	0.29	Vina-GPU+	0.25	1.16X
QuickVina 2	0.14	QuickVina 2-GPU	0.08	1.75X
QuickVina-W	0.63	QuickVina-W-GPU	0.15	4.20X

Acc: the runtime acceleration of Vina-GPU 2.0 against the baseline method.

Figure 7 shows the runtime acceleration (Acc) on each complex by our Vina-GPU 2.0 against the baseline methods. The average acceleration is highlighted by a white dot in the center, and the shape of the violin shows the distribution of Acc values. Compared to AutoDock Vina, Vina-GPU+ achieves the maximal acceleration of 58.26X, as well as the average of 31.90X (Figure 8). Compared to Vina-GPU, Vina-GPU+ achieves the maximal and average acceleration of 2.53X and 1.61X, respectively. Compared with QuickVina 2, QuickVina 2-GPU reaches the maximal acceleration of 5.14X and the average of 1.74X. Compared with QuickVina-W, QuickVina-W-GPU obtains the maximal acceleration of 28.46X and the average of 5.52X.

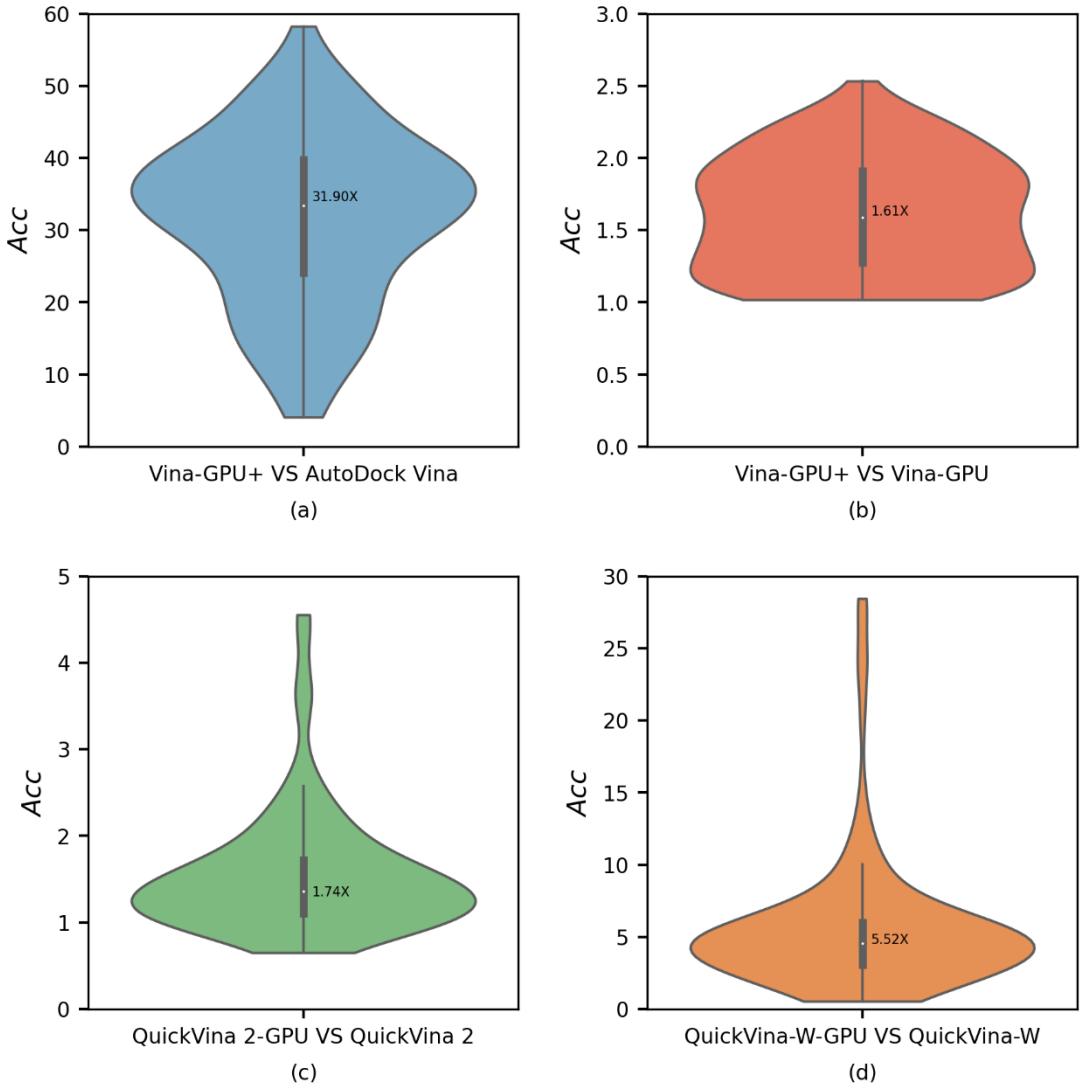

Figure 7. Acceleration of docking time (*Acc*) of our Vina-GPU 2.0 against the baseline methods (Vina-GPU, QuickVina 2, QuickVina-W) on 140 complexes from the AutoDock-GPU dataset. The average *Acc* is highlighted with a white dot in the center. (a) Vina-GPU+ VS AutoDock Vina; (b) Vina-GPU+ VS Vina-GPU; (c) QuickVina2-GPU VS QuickVina2; (d) QuickVina-W-GPU VS QuickVina-W.

3.5 Docking accuracy on the AutoDock-GPU dataset

We compare the overall docking accuracy of our Vina-GPU 2.0 with the baseline methods in terms of the docking score and RMSD performances on all 140 complexes. The color bar encodes the number of atoms in a ligand. For AutoDock Vina VS Vina-GPU+, docking scores of most complexes distribute around the diagonal line and fall into the lavender margin of 0.5 kcal/mol difference and their Pearson correlation coefficient of the scores is 0.971 (Figure 8a), which denotes a significant positive correlation. The average docking score of AutoDock Vina and Vina-GPU+ are -8.92 and -8.65, respectively. These results show that our Vina-GPU+ achieves the very close docking scores with AutoDock Vina. Similar phenomena and conclusions were found in the docking scores of other comparison methods (Figure 8a).

A docking conformation is typically acceptable when its RMSD difference with the ground truth structure is smaller than 2Å, and the red dashed line distinguishes whether a docking conformation is acceptable or not from the RMSD aspect (Figure 8b). Results demonstrates that most complexes fall into the lower left region where both our Vina-GPU 2.0 and the baseline methods succeed to obtain the acceptable docking except for the QuickVina-W VS QuickVina-W-GPU. For instance, for Vina-GPU+, 102 out of 140 RMSD results are within 2 Å, while 111 out of 140 for AutoDock Vina. The average RMSD of AutoDock Vina and Vina-GPU+ are 1.54 and 1.67, respectively. Due to the RMSD results of QuickVina-W is large, and QuickVina-W-GPU uses the GPU to accelerate QuickVina-W while ensuring that the accuracy of both is comparable, the red dashed line is very close to the x and y axes in Figure 8(b). These results show that our Vina-GPU 2.0 achieves the similar comparable docking RMSD with the baseline methods. Thus, these findings indicate that Vina-GPU 2.0 exhibits the comparable docking accuracy with respect to the baseline methods on both docking score and RMSD.

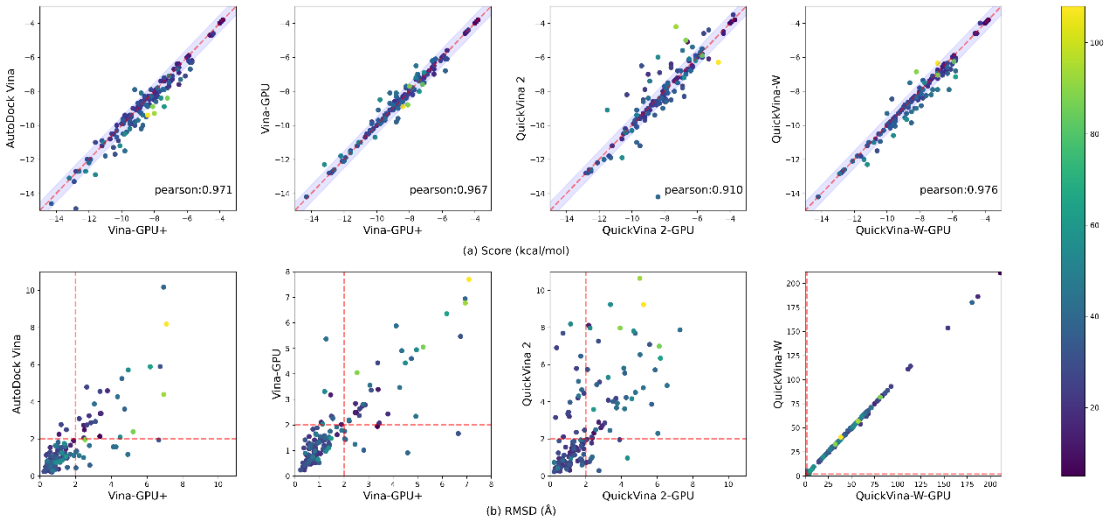

Figure 8. Comparable docking accuracy between Vina-GPU 2.0 and the baseline methods on all 140 compounds. The color bar encodes the number of atoms in one ligand. (a) A margin of 0.5 kcal/mol difference on the docking score between Vina-GPU 2.0 and the baseline methods is highlighted with lavender in Figure 8a. The Pearson correlation coefficient of their docking scores is 0.971,0.967,0.910 and 0.976 respectively (indicated by "pearson"). (b) The RMSD value that indicates an acceptable binding pose (< 2 Å) are separated by a red dashed line in Figure 8b.

3.6 GUI

We developed a user-friendly and installation-free graphic user interface (GUI) for its

convenient operation on Windows 10 Operating System (Figure 9). The GUI contains three docking methods, Vina-GPU+, QuickVina 2-GPU and QuickVina-W-GPU. The users can select docking methods in interest for molecular docking and enter their parameters involving Receptor file path (Receptor), Ligand file path (Ligand), Output Files path (Output File path), 3D coordinates of the center of the docking box (Box center), the size of the docking box (Box size), thread, and search depth. All input parameters required for molecular docking are provided in the GUI, and all input parameters can be customized by users or by default values determined by heuristic formulas. Once all input parameters have been entered, please click on the Start button to start docking. The progress bar will show current docking progress. To facilitate the processing of output conformations, the GUI can generate a table of all output conformations. The table contains the names of all output conformations and their docking scores which are sorted from the lowest to the highest docking score. This allows the user to quickly obtain the best output conformation. The codes, tools and datasets of VINAs-GPU 2.0 are freely available at https://github.com/DeltaGroupNJUPT/VINAs-GPU 2.0, coupling with explicit instructions and examples.

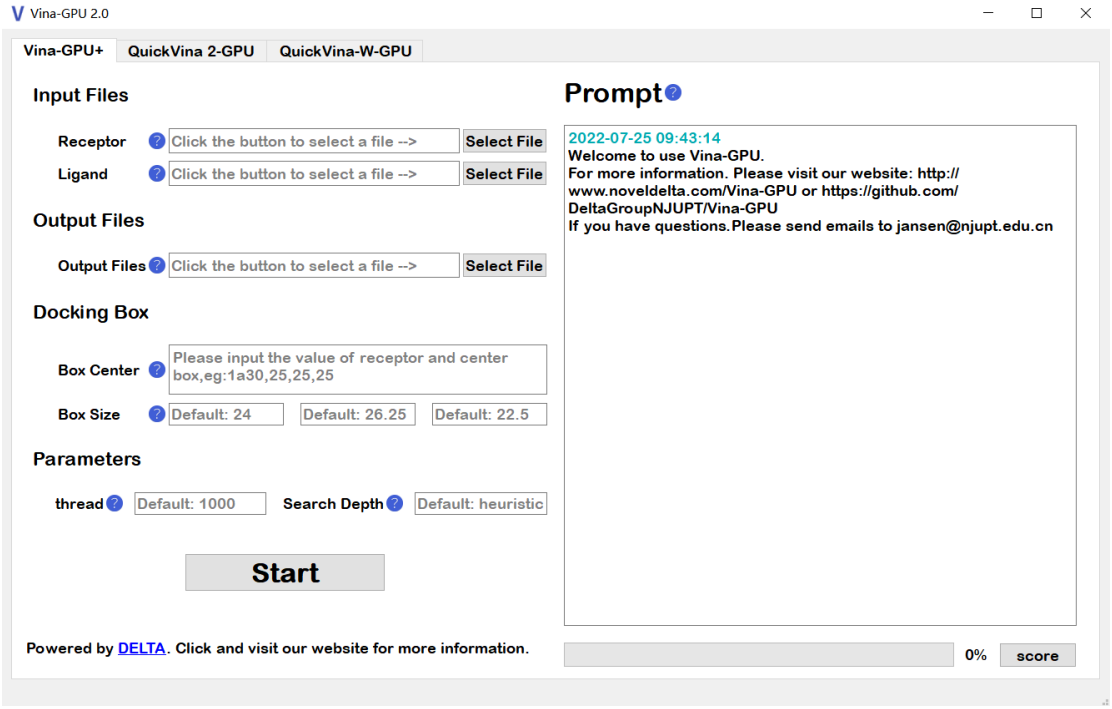

Figure 9. Graphic user interface (GUI) of Vina-GPU 2.0.

Conclusion

The further speedup of AutoDock Vina and its derivatives with GPUs is beneficial for systematically pushing their popularization in large-scale virtual screens due to their high benefit-cost ratio and easy operation for users. Thus, we proposed the Vina-GPU 2.0 method, to further accelerate AutoDock Vina and the most common derivatives with new docking algorithm (QuickVina 2 and QuickVina-W) with GPUs. Caused by the discrepancy of their docking algorithms, our Vina-GPU 2.0 adopts different GPU acceleration strategies. In real virtual screening for two hot protein kinase targets RIPK1 and RIPK3 from the DrugBank database, our Vina-GPU 2.0 reaches an average of 65.6-fold, 1.4-fold and 3.6-fold docking acceleration against the original AutoDock Vina, QuickVina 2 and QuickVina-W while ensuring their comparable docking accuracy. In addition, we develop a friendly and installation-free graphical user interface (GUI) tool for their convenient

usage. The codes and tools of Vina-GPU 2.0 are freely available at https://github.com/DeltaGroupNJUPT/ Vina-GPU 2.0, coupling with explicit instructions and examples. In future studies, the following aspects would be taken into consideration for pushing the popularization of AutoDock Vina and its numerous derivatives in large virtual screens. (1) We will implement GPU acceleration for more derivatives, such as Vina-Carb, AutoDock VinaXB, Vinardo and Smina。

Acknowledgements

We acknowledge the support from Yanxiang Zhu and Ruiqi Chen for providing nice suggestions.

Funding

This work was supported in part by the National Natural Science Foundation of China (61872198, 81771478 and 61971216) and the Basic Research Program of Science and Technology Department of Jiangsu Province (BK20201378).

Conflict of Interest: none declared.

References

[1] Bohacek, Regine S., Colin McMartin, and Wayne C. Guida. "The art and practice of structure‐based drug design: a molecular modeling perspective." Medicinal research reviews 16.1 (1996): 3-50.

[2] Lyu, J. , et al. "Ultra-large library docking for discovering new chemotypes." Nature 566.7743(2019):1.

[3] Trott, Oleg, and Arthur J. Olson. "AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading." Journal of computational chemistry 31.2 (2010): 455-461.

[4] Amr Alhossary, Stephanus Daniel Handoko, Yuguang Mu, and Chee-Keong Kwoh. "Fast, accurate, and reliable molecular docking with QuickVina 2. " Bioinformatics (2015): 2214–2216.

[5] Hassan, N. M. , et al. "Protein-Ligand Blind Docking Using QuickVina-W With Inter-Process Spatio-Temporal Integration." Scientific Reports 7.1(2017):15451.

[6] Gorgulla, Christoph, et al. "An open-source drug discovery platform enables ultra-large virtual screens." Nature 580.7805 (2020): 663-668.

[7] Li, H. , K. S. Leung , and M. H. Wong . "idock: A multithreaded virtual screening tool for flexible ligand docking." IEEE Symposium on Computational Intelligence in Bioinformatics & Computational Biology IEEE, 2012.

[8] Jaghoori, Mohammad Mahdi , B. Bleijlevens , and S. D. Olabarriaga . "1001 Ways to run AutoDock Vina for virtual screening." Journal of Computer-Aided Molecular Design 30.3(2016):237-249.

[9] Yu Y, Cai C, Zhu Z, Zheng H. Uni-Dock: A GPU-Accelerated Docking Program Enables Ultra-Large Virtual Screening. ChemRxiv. Cambridge: Cambridge Open Engage; 2022; This content is a preprint and has not been peer-reviewed.

[10] Tang, Shidi et al. “Accelerating AutoDock Vina with GPUs.” Molecules (Basel, Switzerland) vol. 27,9 3041. 9 May. 2022, doi:10.3390/molecules27093041

[11] Handoko, S. D. , et al. "QuickVina: accelerating AutoDock Vina using gradient-based heuristics for global optimization. " IEEE/ACM Transactions on Computational Biology & Bioinformatics 9.5(2012):1266-1272.

[12] Nivedha, A. K. , et al. "Vina-Carb: Improving Glycosidic Angles During Carbohydrate Docking." Journal of

Chemical Theory & Computation 12.2(2016).

[13] Koebel, Mathew R. , et al. "AutoDock VinaXB: implementation of XBSF, new empirical halogen bond scoring function, into AutoDock Vina." Journal of Cheminformatics 8.1(2016):27.

[14] Quiroga, Rodrigo , and M. A. Villarreal . "Vinardo: A Scoring Function Based on Autodock Vina Improves Scoring, Docking, and Virtual Screening." Plos One 11.5(2016):e0155183.

[15] Eberhardt, J. , et al. "AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings." (2021).

[16] Koes, David Ryan , M. P. Baumgartner , and C. J. Camacho . "Lessons learned in empirical scoring with smina from the CSAR 2011 benchmarking exercise. " Journal of Chemical Information & Modeling 53.8(2013):1893-1904.

[17] Mifflin, Lauren et al. “Receptor-interacting protein kinase 1 (RIPK1) as a therapeutic target.” Nature reviews. Drug discovery vol. 19,8 (2020): 553-571.

[18] Morgan, Michael J, and You-Sun Kim. “Roles of RIPK3 in necroptosis, cell signaling, and disease.” Experimental & molecular medicine, 10.1038/s12276-022-00868-z. 12 Oct. 2022.

[19] Zhan, Chaoning et al. “MLKL: Functions beyond serving as the Executioner of Necroptosis.” Theranostics vol. 11,10 4759-4769. 4 Mar. 2021.

[20] Newton, Kim. “RIPK1 and RIPK3: critical regulators of inflammation and cell death.” Trends in cell biology vol. 25,6 (2015): 347-53.

[21] Metropolis, Nicholas , et al. "Equation of State Calculations by Fast Computing Machines." The Journal of Chemical Physics 21(2004).

[22] Jaccard, P. . "THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE. " New Phytologist 11.2(2010):37-50.

[23] Wishart, David S , et al. "DrugBank 5.0: A major update to the DrugBank database for 2018." Nucleic Acids Research 46.Database issue(2017).

[24] Santos-Martins, D. , et al. "Accelerating AutoDock4 with GPUs and Gradient-Based Local Search." Journal of Chemical Theory and Computation (2021).

[25] Hartshorn, Michael J , et al. "Diverse, High-Quality Test Set for the Validation of ProteinLigand Docking Performance." Journal of Medicinal Chemistry 50.4(2007):726-741.

[26] Li, Y. , et al. "Comparative Assessment of Scoring Functions on an Updated Benchmark: 2. Evaluation Methods and General Results." Journal of Chemical Information & Modeling 54.6(2014):1717-1736.

[27] Sussman, J. L. , et al. "The Protein Data Bank." Genetica 106.1(1999):149-158.