Accelerating AutoDock VINA with GPUs

Shidi Tang,[†] Ruiqi Chen,[‡] Mengru Lin,[‡] Qingde Lin,[¶] Yanxiang Zhu,[‡] Jiansheng Wu,[*,†] Haifeng Hu,[*,§] and Ming Ling[¶]

[†]*School of Geographic and Biological Information, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*

[‡]*VeriMake Research, Nanjing Renmian Integrated Circuit Technology Co., Ltd., Nanjing 210088, China*

[¶]*National ASIC System Engineering Technology Research Center, Southeast University, Nanjing 210096, China*

[§]*School of Telecommunication and Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210003, China*

E-mail: jasen@njupt.edu.cn; huhf@njupt.edu.cn

Abstract

AutoDock VINA is one of the most-used docking tools in the early stage of modern drug discovery. It uses a Monte-Carlo based iterated search method and multi-threading parallelism scheme on multicore machines to improve docking accuracy and speed. However, virtual screening from huge compound databases is common for modern drug discovery, which puts forward a great demand for higher docking speed of AutoDock VINA. Therefore, we propose a fast method VINA-GPU, which expands the Monte-Carlo based docking lanes into thousands of ones coupling with a largely reduced number of search steps in each lane. Furthermore, we develop a heterogeneous OpenCL implementation of VINA-GPU that leverages thousands of computational cores of a GPU, and obtains a maximum of 403-fold acceleration on docking

runtime when compared with a quad-threaded AutoDock VINA implementation. In addition, a heuristic function was fitted to determine the proper size of search steps in each lane for a convenient usage. The VINA-GPU code can be freely available at `https://github.com/DeltaGroupNJUPT/VINA-GPU` for academic usage.

1 Introduction

Modern drug discovery is extremely time-consuming and expensive. It typically takes decades and spends billions of dollars on developing a new drug.[1] Computational molecular docking (MD) provides an efficient and inexpensive way in the early stage of drug design for the identification of leading compounds and their binding affinities.[2–4]

AutoDock suites consist of various docking tools including AutoDock4,[5] AutoDock VINA,[6] AutoDock FR,[7] AutoDock Crank Pep[8,9] and AutoDock-GPU[10,11] etc., among which AutoDock VINA is the most popular one and typically recommended as the first-line[12] in the process of molecular docking. It uses a Monte-Carlo based iterated local search method and a multithreading parallelism on CPUs to improve both docking speed and accuracy. Moreover, it wins the best docking power in the comparative assessment of scoring functions (CASF) benchmark CASF-2016[13] and the best scoring power under the comparison with ten docking programs on a diverse protein–ligand complex sets.[14]

Previous virtual screening pipeline typically operates only on a scale of $10^6 - 10^7$ compound molecules. But, the whole chemical space of small molecular compounds that are drug-like has been estimated to reach more than 10^{60} molecules.[15] So far, the scale of compounds for virtual screening is vital since the more candidate compounds to be screened, the lower rate of false positives and the more favorable quality of leading compounds could be found. Therefore, a great acceleration for virtual screening of huge databases has become a core problem to be anxiously solved.[16]

Caused by the overall serial design of the AutoDock VINA algorithm, its parallelization mainly relies on the stack of computing power as well as the resources and tasks optimal

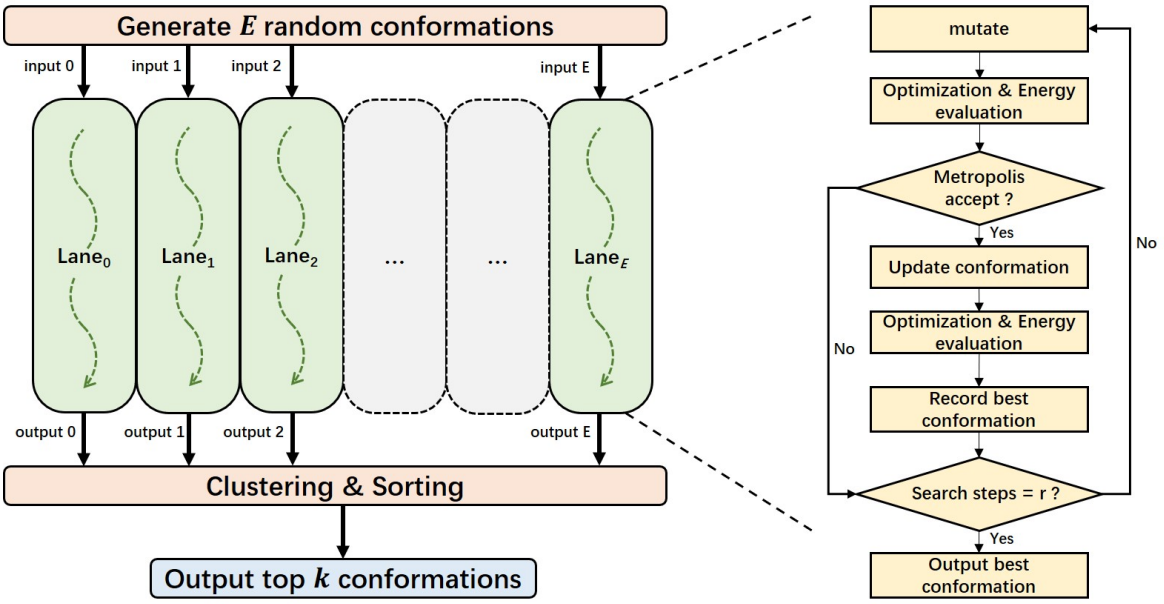


Figure 1: Overall workflow of VINA-GPU. VINA-GPU runs thousands of docking lanes stimulatingly with much reduced search steps in each lane.

allocation in real scenarios of the ongoing situation for large virtual screens. For instance, VirtualFlow[17] provides a drug discovery platform for large virtual screening with up to 16,0000 CPUs. This huge resource investment and expenditure, as well as the high entry threshold for users greatly weaken the popularity of AutoDock VINA and the flexibility of customer's usage (such as a self-defined target and small molecule dataset).

Nowadays, GPU cards with thousands of computing cores have shown a strong power in parallel computation and already been exploited in molecular docking programs.[10,11,18–27] For example, AutoDock-GPU provides an Open Computing Language (OpenCL)[28] implementation of AutoDock4 to exploit both GPU and CPU parallel architectures. By exploring three levels of parallelism (runs, individual, fine-grained tasks) on the Lamarckian Genetic Algorithm (LGA) algorithm, it reduces the total runtime up to 350-fold with respect to a single-threaded AutoDock4.[11] However, as a promising docking tool, only little attentions have been put into the GPU acceleration of AutoDock VINA. The Viking method tried to rewrite the pose search of AutoDock VINA using GPU.[25] Unfortunately, no positive acceleration results were observed. According to our analysis, the main obstacles to the par-

allelization of AutoDock VINA involve the following two aspects. Firstly, the Monte-Carlo based optimization process is serialized by iterations, in which the next iteration relies on the previous outputs. Secondly, each ligand file was formatted as a heterogeneous tree structure whose nodes are traversed recursively. Besides, the Compute Unified Device Architecture (CUDA)[29] used by Viking can only be implemented on NVIDIA GPU cards, which limits its cross-platform portability.

In this work, we propose an efficient parallel method, namely VINA-GPU, to accelerate AutoDock VINA with GPUs. It achieves a large-scale parallelism on a reduced-step Monte-Carlo based iterated method. Moreover, by converting the heterogeneous tree structure into a list format whose nodes are stored in the traversed order, a heterogeneous OpenCL implementation of VINA-GPU can be efficiently deployed. It leverages thousands of GPU computational cores to achieve a higher level of parallelization and acceleration with the cross-platform portability on both CPUs and GPUs. Experimental results show that VINA-GPU can reach a maximum of 403 times speed-up and the comparable docking performance when compared with a quad-threaded AutoDock VINA instance. The code of VINA-GPU is freely available for academic users at `https://github.com/DeltaGroupNJUPT/VINA-GPU`.

Our contributions in this paper are:

(1) We analyze the AutoDock VINA algorithm and explored the inherent dependency of Monte-Carlo based iterated local search method in details. Then, we propose an efficient method, namely VINA-GPU, to achieve large-scale parallelism via much reduced search steps in the Monte-Carlo iterations.

(2) To the best of our knowledge, it is the first to develop a heterogenous OpenCL implementation of AutoDock VINA on GPUs which obtains significant accelerations when compared with CPU-based pipelines.

(3) For a given ligand, a heuristic function is fitted to set the proper size of search steps for a convenient usage of our VINA-GPU tool.

2 Methodology

2.1 VINA-GPU

AutoDock VINA utilizes a Monte-Carlo based iterated local search method for optimization with gradients. It consists of a succession of search steps which starts from a random conformation and implements the optimization continuously. A ligand conformation can be simply represented by its position, orientation and torsion (POT). Each step is to randomly mutate the ligand conformation and then to calculate its energy followed by an optimization with the Broyden-Fletcher-Goldfarb-Shanno (BFGS)[30] method. The metropolis acceptance on its calculated energy is adopted to determine whether the conformation can be accepted. If accepted, the conformation will be finely optimized by BFGS for a second time and then added into a container that involves all the best 20 candidate conformations. Then, the conformation will be continuously mutated in the next step until reaching the pre-set search steps. The more steps it takes, the better conformations are likely to be found.

The AutoDock VINA algorithm is inherently serialized, in which a conformation returned by the previous step determines the input of next step. Such an inherent dependency heavily prevents the algorithm from a fine-grained parallelism such as, to implement each of search steps concurrently. Thus, it is a better choice to implement the parallelism scheme on a coarse-grained level of the Monte-Carlo based search method. In this work, we treat each Monte-Carlo based iterated docking as an independent computing lane, and the size of lanes represents the parallelism scale. AutoDock VINA provides a multithreading parallelism on multicore CPUs using the boost library[31] to gain an acceleration of docking speed. Multiple lanes can be executed concurrently on their corresponding threads each of which starts from a randomized conformation of the given ligand. However, the boost library can only be used on CPUs and the acceleration greatly depends on the number of CPU cores. For the vast majority of prevailing multi-core computers, the effect of parallelism acceleration is very limited, and far from meeting the current demands for virtual screening on ultra large

compound databases.

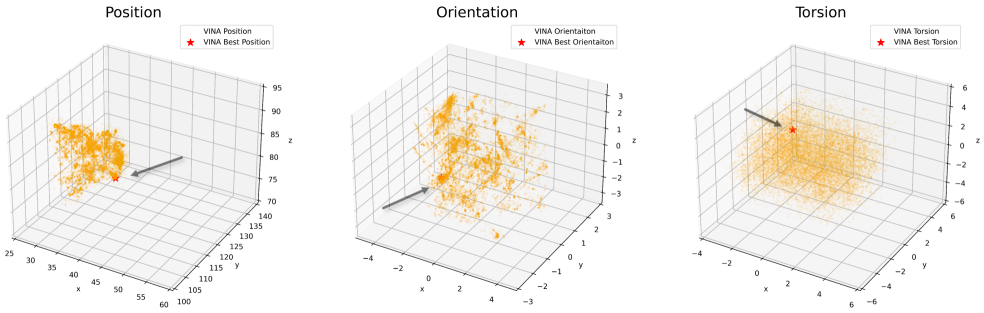
Therefore, we intend to explore a larger scale of parallelism on the course-grained level of AutoDock VINA. According to our analysis, the whole searching space is pre-determined by a "docking box" with its center, length, width and height defined by arguments (center_x, center_y, center_z, size_x, size_y, size_z). For a given conformation, it usually requires $\text{search\_steps}_{\text{VINA}}$ search steps (normally tens of thousands) in the process of optimization, which occupies most of the total docking runtime. The size of search steps depends on the complexity of the ligand and can be calculated by

$$\text{search\_steps}_{\text{VINA}} = 105 \times \text{N}_{\text{atom}} + 1050 \times \text{N}_{\text{rot}} + 11550 \tag{1}$$

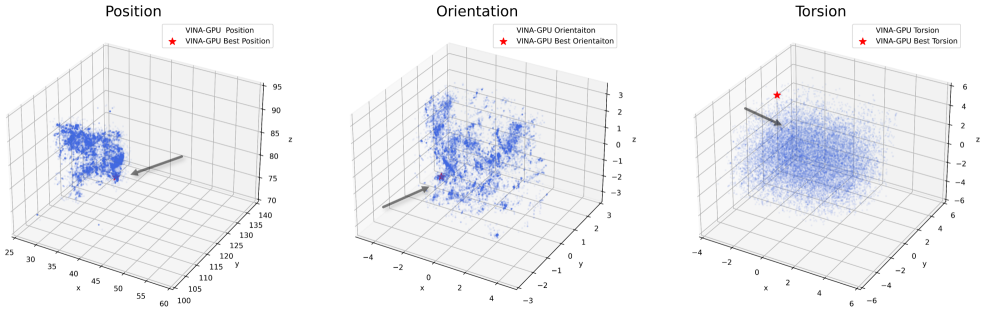
where N_{atom} and N_{rot} are the number of atoms and rotatable bonds in the ligand. It is worth noting that eq 1 does not take any searching space information into consideration, probably because AutoDock VINA treats every lane as an independent one, thus, each of lanes shares the same size of searching space. It might be true for searching such a large number of steps if only a few lanes (8 by default in AutoDock VINA) are calculated simultaneously using multi-threading. When the number of lanes reaches to thousands, however, the search steps for each lane might be greatly reduced. Because, to calculate thousands of lanes simultaneously needs to initialize thousands of random conformations in the searching space. We can regard it as dividing the whole searching space into thousands of sub-spaces where each initial conformation is to be optimized. A high-dimensional space $\boldsymbol{S}$ can be used to denote the overall solution space covering all the possible conformations, each of which is described as

$$\boldsymbol{Pose}_i = \{x, y, z, a, b, c, d, \psi_1, \psi_2, ..., \psi_{N_{rot}}\}, (i = 1, 2, \dots) \tag{2}$$

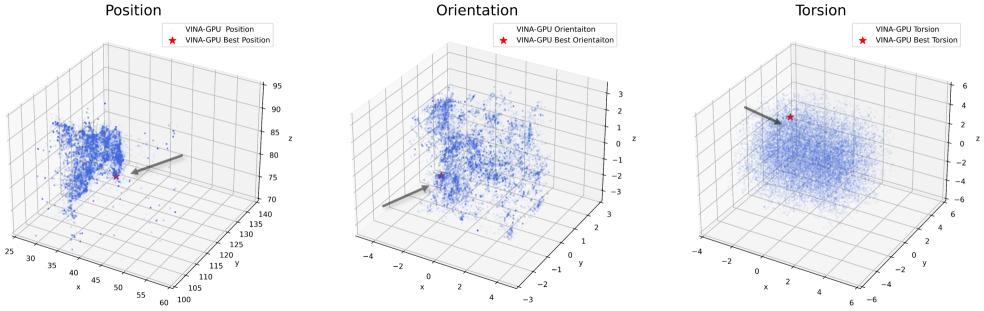
where x, y, z correspond to the position of the conformation in the box; a, b, c, d denote the orientation in the quaternion form; $\psi_1, \psi_2, ..., \psi_{N_{rot}}$ are related to torsions of N_{rot} rotatable



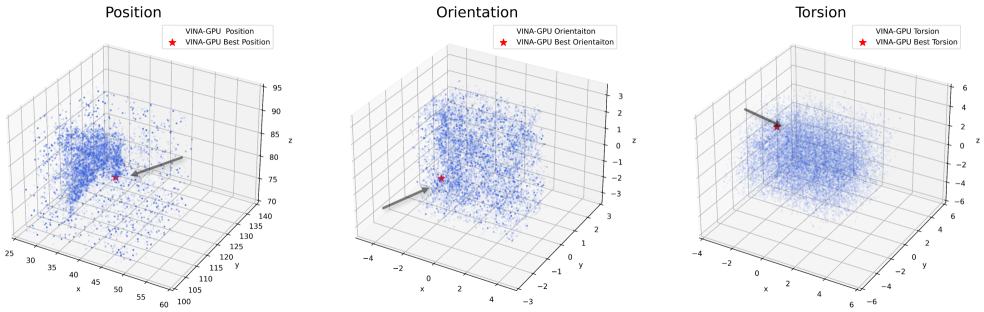
(a) VINA: 1 lanes with 22365 steps each lane



(b) VINA-GPU: 10 lanes with 2237 steps each lane



(c) VINA-GPU: 100 lanes with 224 steps each lane



(d) VINA-GPU: 1000 lanes with 22 steps each lane

Figure 2: Conformation spaces (of PDBid: 2bm2) searched by AutoDock VINA or VINA-GPU method. VINA-GPU method is used under different number of lanes with different search steps in each lane. Each of the conformations is represented by its position, orientation and torsion and is plotted with dots, separately. The principal component analysis (PCA) method is used to reduce the dimensionality into three. The best conformations found are highlighted with red stars (pointed by arrows).

bonds. Therefore, $\boldsymbol{S}$ can be represented as the sum of three spaces

$$\boldsymbol{S} = \{\boldsymbol{P} + \boldsymbol{O} + \boldsymbol{T}\} \tag{3}$$

where $\boldsymbol{P}$, $\boldsymbol{O}$ and $\boldsymbol{T}$ are 3-dimensional position, 4-dimensional orientation and $\mathrm{N_{rot}}$-dimensional torsion spaces respectively. By dividing $\boldsymbol{S}$ into n sub-spaces, we have

$$\boldsymbol{S} = \{\boldsymbol{S}_{sub_1}, \boldsymbol{S}_{sub_2}, ..., \boldsymbol{S}_{sub_n}\} \tag{4}$$

where

$$\boldsymbol{S}_{sub_i} = \{\boldsymbol{P}_{sub_i} + \boldsymbol{O}_{sub_i} + \boldsymbol{T}_{sub_i}\} \tag{5}$$

Each subspace $\boldsymbol{S}_{sub_i}$ couples with a randomly initialized conformation $\boldsymbol{Pose}_i \in \boldsymbol{S}_{sub_i}$. Hence, for each lane, the search space in $\boldsymbol{S}_{sub_i}$ is much smaller than that in $\boldsymbol{S}$. As a result, we can reduce the number of corresponding search steps in each lane for finding the best conformation within $\boldsymbol{S}_{sub_i}$. Latter experiments at the end of this section also verify our hypothesis.

Based on the discussions above, we proposed a novel method VINA-GPU, which calculates thousands of lanes concurrently with much reduced search steps in each lane to achieve a massive parallelism and acceleration. Fig 1 shows the overall workflow of VINA-GPU. It first generates a large number of E random conformations depending on how many lanes can be deployed. Then, each of the E conformations is assigned to an independent lane, in which a significantly reduced number of search steps (r) of Monte-Carlo based iterated docking is performed similar with AutoDock VINA. Finally, the best conformations among all E lanes are clustered and sorted to output the top k ones.

To demonstrate the equivalence of AutoDock VINA and VINA-GPU, the total conformations explored by both of them are plotted in Fig 2. A representative complex (PDBid: 2bm2, $\mathrm{N_{atom}} = 33$, $\mathrm{N_{rot}} = 7$) is selected for its relatively medium complexity. Each conformation of 2bm2 is represented by its POT and shown in the cartesian coordinates separately.

The principal component analysis (PCA) method is used to reduce the dimensionality into three. Conformations searched by AutoDock VINA and VINA-GPU (with different lanes and various sizes of search steps in each lane) are highlighted with orange and blue dots, respectively. The best conformations are shown in red star (pointed by an arrow) and the sizes of total conformations among all lanes are kept almost the same ($\sim$ 22365).

As shown in Fig 2a and Fig 2b, the total conformations explored by VINA-GPU and AutoDock VINA occupy almost the same position, orientation and torsion. With the increase of parallelism scale and the reduce of search steps in each lane, this situation keeps unchanged (Fig 2c and Fig 2d). Moreover, the best conformations found by VINA-GPU are similar to those of AutoDock VINA. It maybe indicates that VINA-GPU shares the same possibility with AutoDock VINA of discovering these good conformations by expanding the scale of parallelism and reducing the size of search steps.

2.2 OpenCL Implementation

Open Computing Language (OpenCL) is an open, royalty-free standard for parallel programming of diverse accelerators such as CPUs, GPUs.[28] Based on the discovery that expanding the lanes while reducing search steps in each lane could explore the same conformations compared with AutoDock VINA, the OpenCL implementation of VINA-GPU is to achieve acceleration by mapping all lanes into computing cores of GPU. Our implementation focuses on the most time-consuming Monte-Carlo based optimization process, which takes $\sim$ 90% of the total runtime of AutoDock VINA. Thus, we propose a heterogeneous architecture (see Fig 3) which is separated into host (CPU) and device (GPU) sides.

Firstly, on the host side, it begins with the ligand and receptor file reading and then setups the OpenCL environment including identifying and choosing platforms and devices; creating context, command queue, program and kernel; passing arguments to the kernel. Moreover, it prepares the data needed for calculations on the device side, which involves the grid cache (for calculating the energy of a conformation by trilinear interpolation), random maps (for

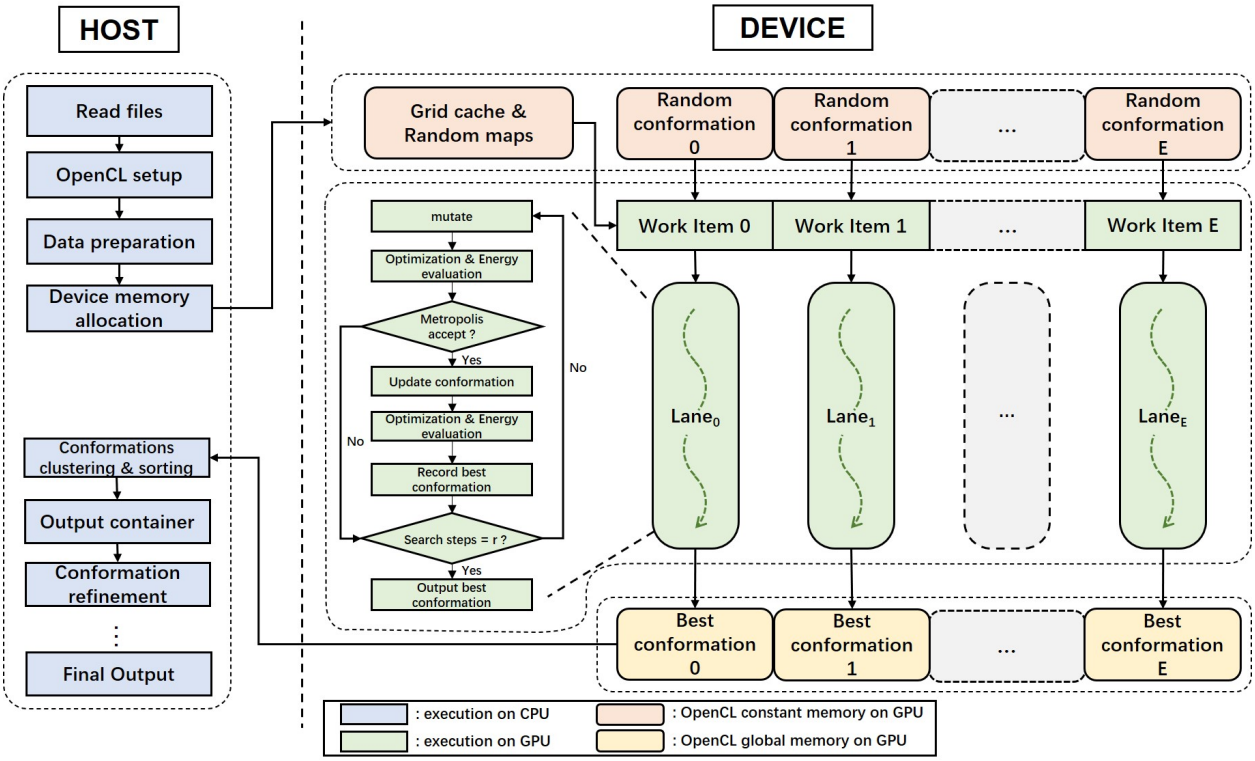


Figure 3: The OpenCL architecture for implementing VINA-GPU, which consists of a host (CPU) and a device (GPU) side of execution.

generating probability random numbers) and random conformations (for each docking lane to start from). The read-only grid cache, random maps and random conformations are allocated in the constant device memory while the read-write best conformations returned by docking lanes are allocated in the global device memory. Such a memory management could efficiently boost the read and write speed. Secondly, on the device side, each work-item performs a corresponding docking lane described in Fig 1. E output conformations from E work-items (lanes) are clustered and copied back to the host. Finally, on the host side, E conformations from the device are sorted in the container. The best 20 (default) ones are to be finely refined before the final output files generated.

It is worth noting that AutoDock VINA calculates the energy in a recursive process. It treats each conformation as a heterogeneous tree structure whose nodes are stored with its frame information and a pointer to its children node. Each node is traversed by a depth-first search method. However, OpenCL does not support any recursion in kernels because the

allocation of stack space for thousands of threads is too expensive. Besides, various ligands could generate totally different heterogenous trees which are not suitable for the OpenCL implementation. Therefore, during the process of data preparation on the host side, we re-construct the heterogenous tree into a list type (see Fig 4) where each node is stored in its traversed order. By doing so, nodes can be traversed simply from the head to the tail of the node list. In addition, a children map is created to denote the relationship between nodes. For example, node 0 has two children-nodes (node 1 and node 4), because, in the children map, row 0 has two "T"s (indicating "True") in the 1st and 4th column and "F"s (indicating "False") in other columns. Thus, the recursive traverse of the heterogenous tree can be converted into an iterative traverse of the node list and children map.

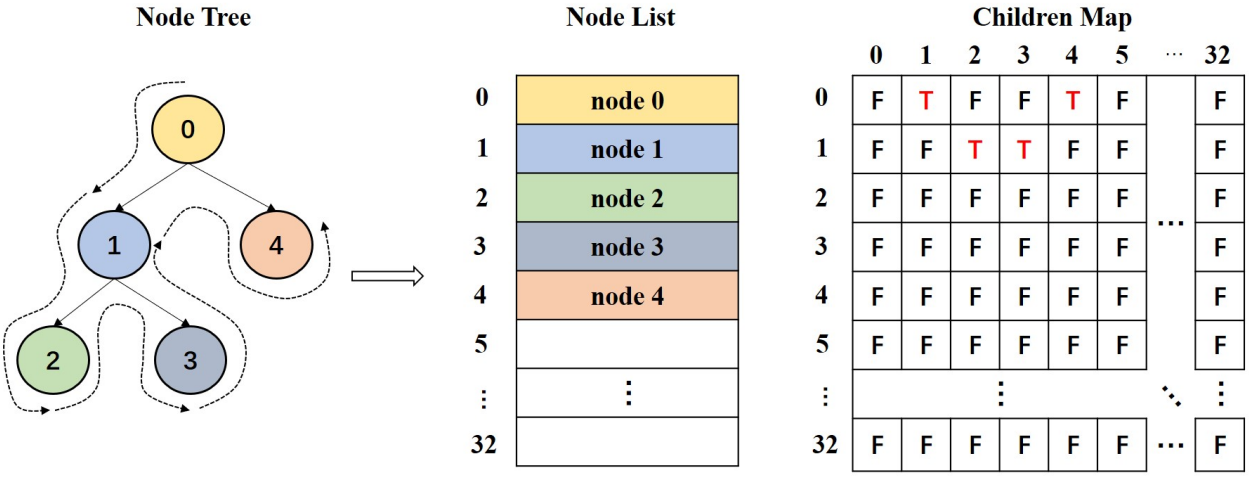


Figure 4: Transformation from the node tree structure into the node list format. The heterogenous tree is re-constructed in its traversed order(depth-first). An additional children map is built to reflect the relationship between nodes.

3 Experiment Results and Discussion

3.1 Datasets

Our datasets are extracted from the test set of 140 complexes for AutoDock-GPU.[32] It comprises of 85 complexes from Astex Diversity Set,[33] 35 complexes from CASF-2013,[34]

and 20 complexes from Protein Data Bank.[35] It covers a wide range of ligand complexities and targets properties. For each complexes, an X-ray pose and an initial random pose of the ligand as well as its corresponding receptor are included (in *.pdbqt* format). Besides, we create a *config.txt* file for each complex (see the example in Appendix Table 4), which involves the 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). Properties of the 140 complexes are described in Appendix Table 1.

3.2 Accuracy Evaluation Criteria

We intend to evaluate the docking accuracy of VINA-GPU with comparisons to AutoDock VINA baseline. AutoDock VINA provides several configurable arguments for users to customize, including the center and the volume of the searching box, the number of CPU cores to be used (*cpu*) and docking lanes (*exhaustiveness*) etc. For the most time-and-performance-effective arguments, *cpu* and *exhaustiveness*, are set as 4 and 8 respectively. The corresponding docking results and docking runtimes of AutoDock VINA are set as the baselines.

For VINA-GPU, we consider its docking result successful based on either of the following two criteria. The first one is the final docking score, which represents the docking affinity of ligand and receptor (the lower the better). If the difference between the lowest docking scores output by VINA-GPU and AutoDock VINA on the same complex is no more than 1 kcal/mol, we consider that the accuracy of VINA-GPU is similar to that of AutoDock VINA, hence being a successful docking. The other criterion is the distance between the results and the known X-ray structures, simply referred to as the root-mean-square deviation (RMSD). It represents the conformation difference between the output one and the ground truth (also the lower the better). We consider it a successful docking if the least RMSD is no more than 2 Å.

3.3 Experimental Setup

For AutoDock VINA, the program is executed on Intel (R) Xeon (R) Gold 6130 CPU @ 2.1GHz using Windows 10 Operating System with 32 GB memory. *cpu* is set to 4 and *exhaustiveness* is set to 8. Extras including box center (center_x, center_y, center_z) and volume (size_x, size_y, size_z) are set in the *config.txt* file (also see Appendix Table 4).

For VINA-GPU, the program is developed with OpenCL v.3.0 and executed on three different computational performance GPUs (Nvidia Geforce GTX 1060, Nvidia Geforce RTX 2080Ti, Nvidia Geforce RTX 3090) with corresponding three different CPUs (Intel (R) Core (TM) i7-8700 @ 3.2GHz, Intel (R) Xeon (R) Gold 6130 @ 2.1GHz, AMD EPYC 7532 @ 2.4GHz) because different server platforms are used in our experiments. Details of the three GPU cards are shown in Table 1.

Besides the center and volume of the box, we provide a flexible configuration for users, including the total number of work-items (*threads*) and the number of search steps in each work item (*search_steps*). Specifically, according to the computational performance of each GPU card (from lower-end to higher-end), we set *threads* as 2500, 5000, 7500 for Nvidia Geforce GTX 1060, Nvidia Geforce RTX 2080Ti and Nvidia Geforce RTX 3090, respectively. The *search_steps* is determined cautiously based on our experiments discussed in section 3.5.

Table 1: Details of GPU cards used in this work

Device	Peak Compute Performance		CUDA cores	On-board memory	Architecture	Carried CPU
	FP32	FP64				
NVIDIA Geforce GTX 1060	4.375 TFLOPS	136.7 GFLOPS	1280	6GB	Pascal	Intel (R) Core (TM) i7-8700 @ 3.2GHz
NVIDIA Geforce RTX 2080Ti	13.45 TFLOPS	420.2 GFLOPS	4352	11GB	Ampere	Intel (R) Xeon (R) Gold 6130 @ 2.1GHz
NVIDIA Geforce RTX 3090	35.58 TFLOPS	556.0 GFLOPS	10496	24GB	Turing	AMD EPYC 7532 @ 2.4GHz

3.4 Docking Accuracy Evaluation

Based on the two criteria defined in Section 3.2, we are able to explore the docking performance of VINA-GPU. For each complex among our datasets of 140 complexes, on three different GPU cards, we set different *search_steps* (from 1 to 100) to explore their influences on the docking performance of VINA-GPU. The results of selected six typical complexes with various complexities (described in Table 2) are demonstrated in Fig 5 and Fig 6 in terms of scores and RMSDs. The AutoDock VINA baselines are plotted with solid lines (red) and the corresponding score and RMSD criteria determined in Section 3.2 are plotted with dotted lines (red). The average results from three GPU cards are plotted in solid lines (blue).

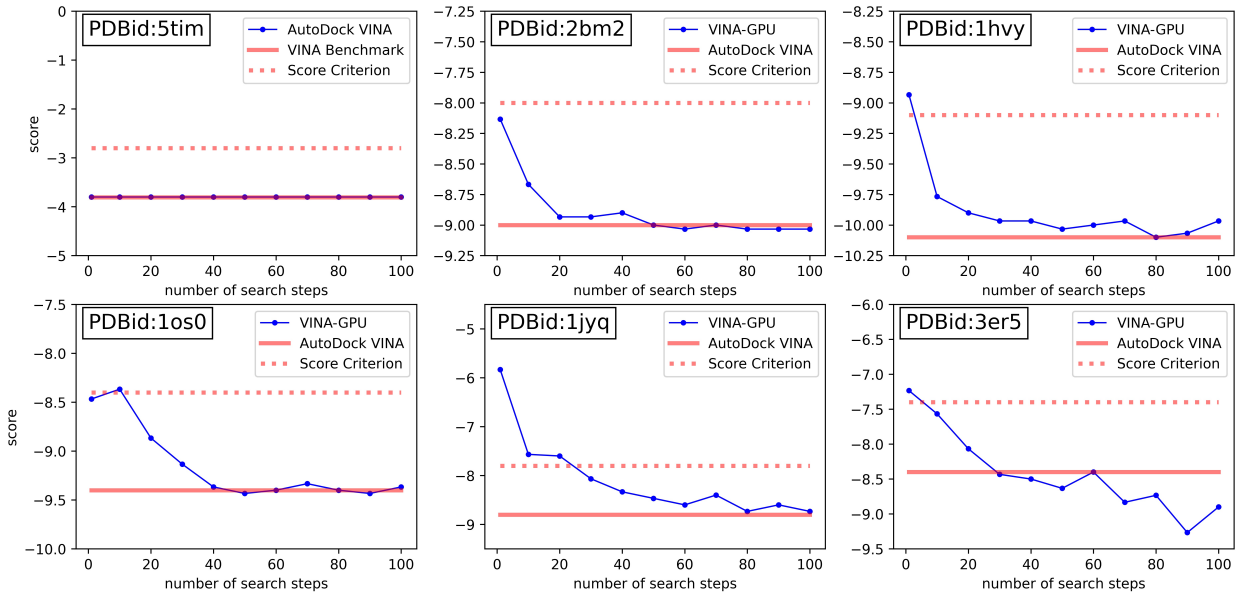


Figure 5: Docking scores of VINA-GPU on six PDB structures with various levels of complexity. The solid red lines mean the docking scores returned from AutoDock VINA with *cpu* and *exhaustiveness* set as 4 and 8. The dotted lines denote the score upper bounds of an acceptable docking pose. The blue solid lines denote the average scores of VINA-GPU from three GPU cards. The horizontal axis indicates the number of search steps in each Monte-Carlo based docking lane of VINA-GPU.

In Fig 5 we can see that for simple complexes (such as 5tim), it is enough to take only one search step to obtain the same docking score as AutoDock VINA. For more complicated complexes (such as 2bm2, 1hvy, and 1os0), a couple of more steps are needed until converging

to the AutoDock VINA baselines. For some larger complexes (such as 3er5), increasing *search_steps* could efficiently improve the results but can not converge in several tens of steps. Notice that for some of the large complexes (such as 3er5), the score of AutoDock VINA baseline is beyond that of VINA-GPU. This is because the Monte-Carlo based method does not guarantee to find a global minimum, resulting in the instability of scores generated by each docking run. Thus, in some cases, VINA-GPU shows better performance in terms of the docking score. In total, all six complexes succeed to meet the score criterion.

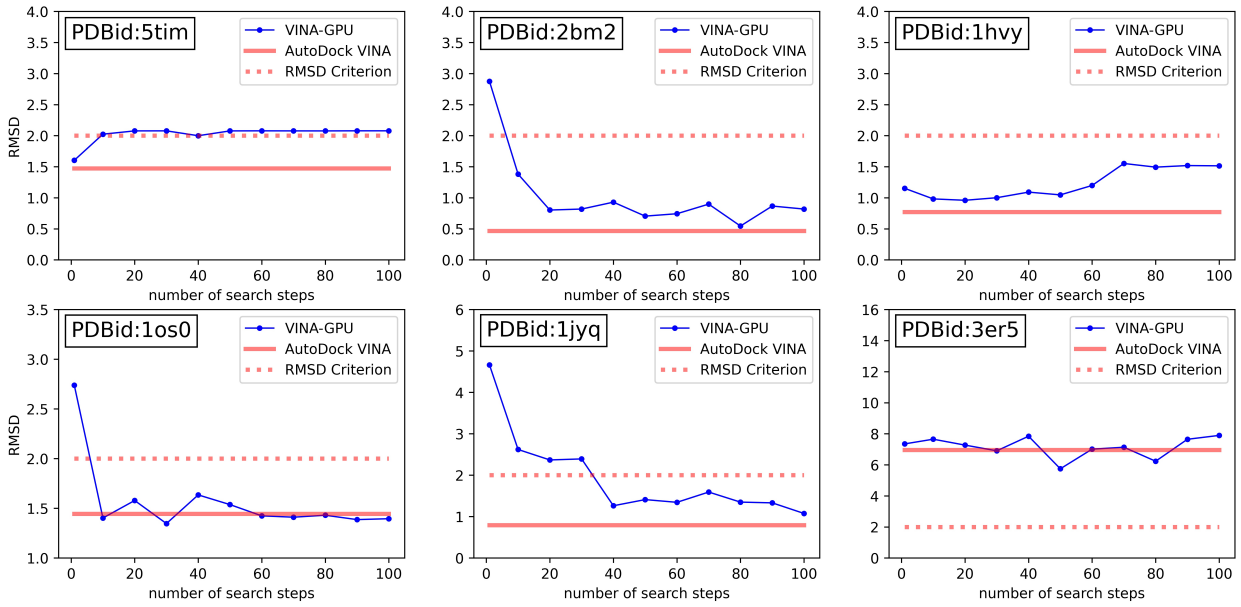


Figure 6: RMSDs of VINA-GPU on six PDB structures with various levels of complexity. The solid red lines mean the RMSDs calculated from the result of AutoDock VINA with *cpu* and *exhaustiveness* set as 4 and 8. The dotted lines denote the RMSD upper bounds of an acceptable docking pose. The blue solid lines denote the average RMSDs of VINA-GPU from three GPU cards. The horizontal axis indicates the number of search steps in each Monte-Carlo based docking lane of VINA-GPU.

In Fig 6, we can see that for some of the complexes, such as 2bm2, 1os0, and 1jyq, as the number of search steps goes larger, their RMSDs converge to the AutoDock VINA baselines. For the others, they either converge to a higher RMSD (such as 5tim and 1hvy) or fluctuate around the baseline (such as 3er5). This is probably because the way in which VINA-GPU works, which clusters all the best conformations, is based on scores, rather

than RMSDs. Thus, VINA-GPU tends to keep those conformations with lower scores as candidates. However, for some of the complexes such as 1hvy, a lower score does not come with a lower RMSD. Appendix Table 5 shows nine models generated from AutoDock VINA (PDBid: 1hvy) with their scores and corresponding RMSDs. It indicates that lower scores could sometimes bring higher RMSDs (see Appendix Table 5 model 1 and model 2) and same scores could also vary largely in RMSD (see Appendix Table 5 model 5 and model 6). Such inconsistency between scores and RMSDs causes that, in Fig 5 and Fig 6, similar scores bring higher RMSDs in the cases of 5tim and 1hvy, while lower scores bring fluctuant RMSDs in the case of 3er5.

Generally, all 140 complexes in our datasets succeed to meet the score criterion. As for the RMSD criterion, we did not take into consideration those complexes for which even AutoDock VINA could not give out a satisfying RMSD result (such as 3er5 in Fig 6). Because in this case, the problem is due to the AutoDock VINA scoring function, which is also used by VINA-GPU. At last, only 5 out of 101 complexes cannot be docked correctly by VINA-GPU, while AutoDock VINA could.

3.5 Search Steps Determination

Intuitively, searching with more steps in each lane offers a larger possibility of discovering a better conformation, but it suffers from longer runtime. Hence, under the precondition of a successful docking, we intend to set the smallest number of search steps to be the best *search_steps* on each complex to gain the maximum acceleration with an acceptable docking result. Based on the discussions in section 3.4, it is straightforward to determine the best *search_steps* according to the score criterion but it is hard based on the RMSD criterion due to the inconsistency of those two. As a result, we simply determine the best *search_steps* of aforementioned six complexes by the score criterion (described in Table 2). The complete best numbers of search steps for 140 complexes are shown in Appendix Table 1.

For a more general use scenario of VINA-GPU, in which a proper *search_steps* has

Table 2: Properties of six complexes and their best search steps

PDBid	N_{atom}	N_{rot}	Best *search_steps*
5tim	5	0	1
2bm2	33	7	10
1hvy	34	9	10
3s8o	44	12	10
1jyq	60	20	30
3er5	108	31	10

to be set for a given new complex, we intend to fit an empirical formula to determine the *search_steps* with respect to N_{atom} and N_{rot}. Specifically, we randomly split our data set into a training set and a test set, including 75% and 25% of the 140 complexes respectively. Then, we fit eq 6 with the lest squares method based on the known N_{atom} and N_{rot} in addition to the best *search_steps* of the complexes in the training set. Moreover, the proper *search_steps* of complexes in the test set are predicted using eq 6. The corresponding docking results of test set using VINA-GPU are described in Appendix Table 2 and Appendix Table 3. We can see that all 35 docking results of the complexes in the test set meet the score criterion (highlighted in bold). We also filter out the complexes in which AutoDock VINA could not find a conformation that meet the RMSD criterion. At last, the remaining 27 out of 29 complexes meet the RMSD criterion and their docking results (in RMSD) are also highlighted in bold.

$$\text{search\_steps} = 0.24 * N_{atom} + 0.29 * N_{rot} - 5.74 \tag{6}$$

3.6 Runtime Performance Comparison

Based on the best *search_steps* selected in Section 3.5, we are able to explore the VINA-GPU runtime acceleration compared with AutoDock VINA runtime. Considering that our implementation of VINA-GPU possesses the heterogeneous nature which splits the whole computation into host and device sides (see Fig 3), we intend to explore the runtime acceleration in 1) the device runtime ($Acceleration_d$) and 2) the total (device + host) runtime

($\text{Acceleration}_{\text{total}}$), which can be simply calculated by

$$\text{Acceleration}_{\text{d}} = \frac{\text{Runtime}_{\text{VINA}_{\text{mc}}}}{\text{Runtime}_{\text{VINA}-\text{GPU}_{\text{d}}}} \tag{7}$$

$$\text{Acceleration}_{\text{total}} = \frac{\text{Runtime}_{\text{VINA}_{\text{total}}}}{\text{Runtime}_{\text{VINA}-\text{GPU}_{\text{total}}}} \tag{8}$$

where $\text{Runtime}_{\text{VINA}_{\text{mc}}}$ and $\text{Runtime}_{\text{VINA}_{\text{total}}}$ are the Monte-Carlo based method runtime and total runtime of AutoDock VINA respectively. $\text{Runtime}_{\text{VINA}-\text{GPU}_{\text{d}}}$ and $\text{Runtime}_{\text{VINA}-\text{GPU}_{\text{total}}}$ are the device runtime and the total runtime of our implementation of VINA-GPU respectively Besides, we divide the 140 complexes into three subsets (small, medium, large) according to their N_{atom} (5-23, 24-36, 37-108) to discover the accelerations on different complexities of complexes. Accelerations are illustrated with violin plots (Fig 7), in which the average accelerations on each GPU card are plotted with a white dot.

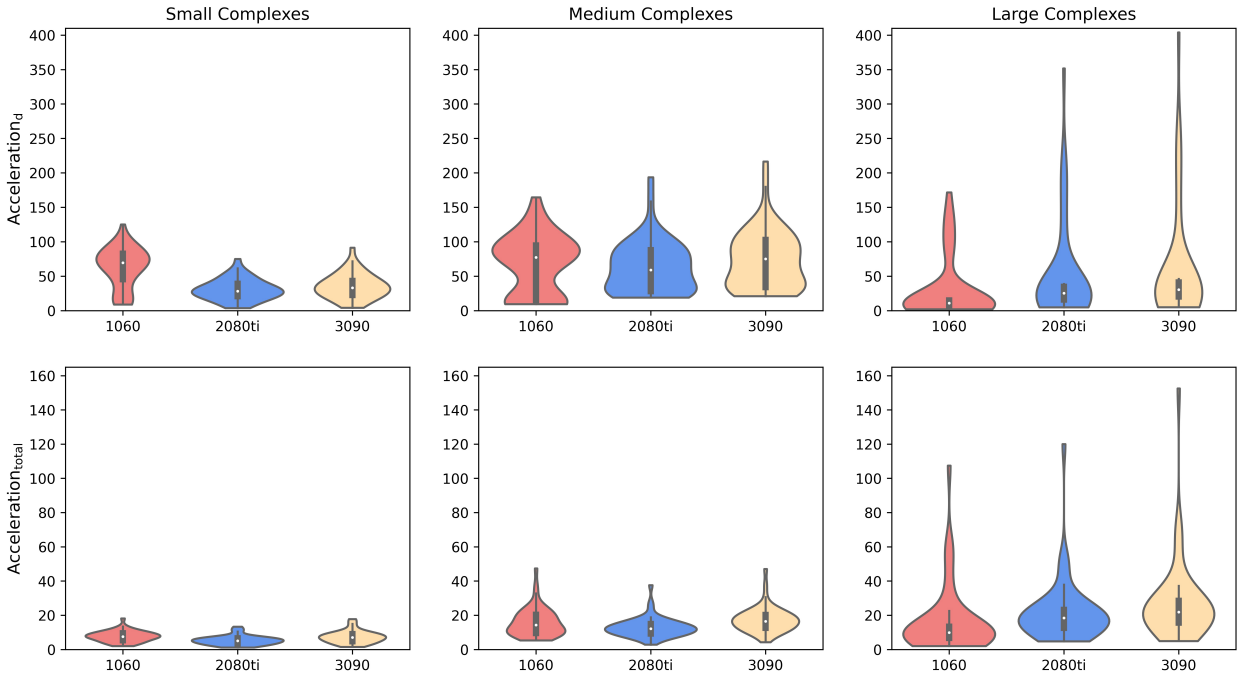


Figure 7: Acceleration$_{\mathrm{d}}$ (a maximum of 403X) and Acceleration$_{\mathrm{total}}$ (a maximum of 153X) of 140 complexes on three GPU cards. To explore the acceleration on different complex complexities, we divide our datasets into three subsets (small, medium, large) according to their N_atom (5-23, 24-36, 37-108). The accelerations are plotted in blue, orange and red with GPU cards of 1060, 2080ti and 3090, respectively. Notice that the acceleration axes are not in the same scale.

In Fig 7, VINA-GPU achieves a maximum of 403X and average 54X, 49X, 57X accelerations (under 3 GPU cards) for Acceleration$_{\mathrm{d}}$ and a maximum 152X and average 13X, 13X, 17X accelerations (under 3 GPU cards) for Acceleration$_{\mathrm{total}}$. Notice that, firstly, VINA-GPU on NVIDIA Geforce GTX 1060 shows larger accelerations compared with other higher-end GPU cards in some cases, especially in small complexes. This is probably because for those small complexes, calculations on GPU only take a small part of the total runtime. Higher-end GPUs with more advanced architectures need more time to set up, resulting in relatively lower accelerations; Secondly, the accelerations are higher for medium complexes, lower for small and large complexes. This is because medium complexes benefit more from VINA-GPU method, which largely reduces their required *search_steps* (often to 1). But for small and large complexes, they either spend little time from the first place or require more *search_steps*

(usually several 10s); Lastly, the overall $\text{Acceleration}_{\text{total}}$ are much lower than $\text{Acceleration}_{\text{d}}$. This is because the CPU runtimes on three different devices are almost same, the Amdahl's law[36] leads to a lower acceleration in terms of the total runtime.

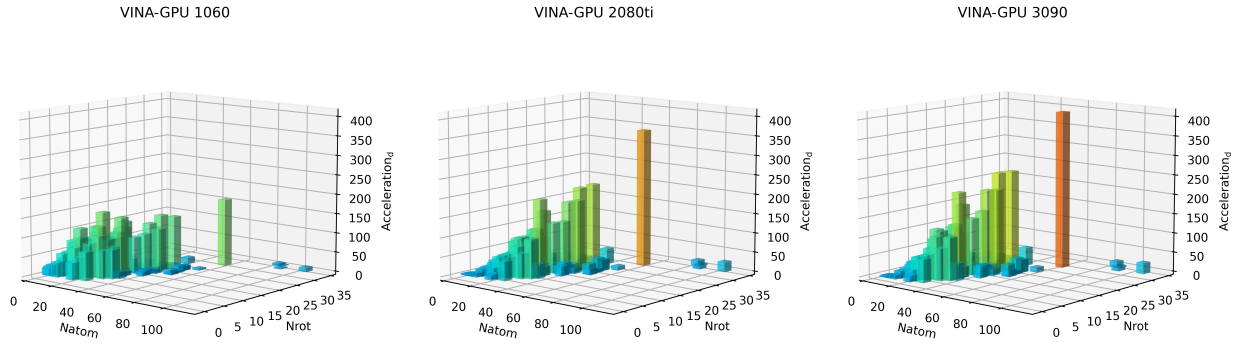


Figure 8: $\text{Acceleration}_{\text{d}}$ of 140 complexes with different N_{atom} and N_{rot}. The acceleration axis ranges from 0 to 410. Each bar represents a complex with its acceleration. The color from blue to red indicates the acceleration from low to high.

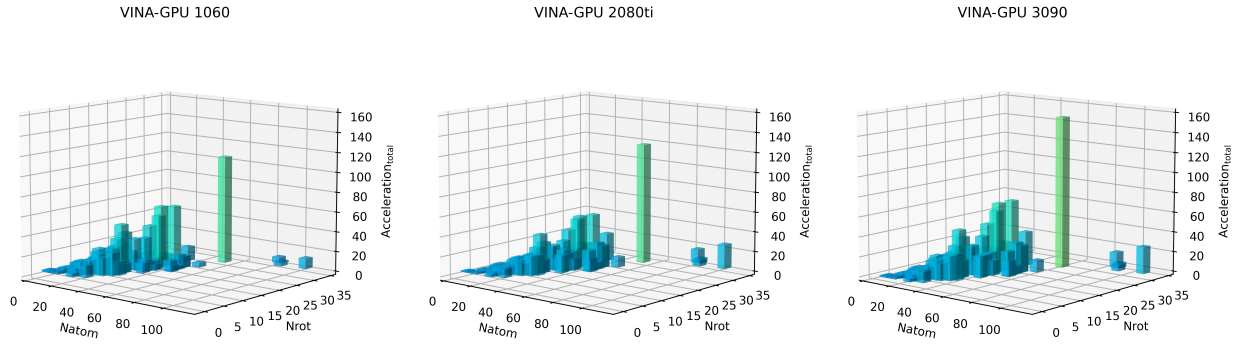


Figure 9: $\text{Acceleration}_{\text{total}}$ of 140 complexes with different N_{atom} and N_{rot}. The acceleration axis ranges from 0 to 160. Each bar represents a complex with its acceleration. The color from blue to red indicates the acceleration from low to high.

From a more direct perspective to explore the acceleration for each of the 140 complexes on different GPU cards, we plot each of the complexes with their N_{atom} and N_{rot} and corresponding accelerations in Fig 8 and Fig 9. Each bar represents a complex with its acceleration. Notice that the two figures have different scales in acceleration axis and the color from blue to red indicates acceleration from low to high.

3.7 Conclusion and Future Work

This work describes an accelerated Monte-Carlo based iterated searching method, namely VINA-GPU, and proposes a heterogeneous OpenCL implementation of VINA-GPU on GPUs. By expanding the scale of parallelism on the relatively course-grained Monte-Carlo based docking lanes and reducing the number of search steps needed in each lane, VINA-GPU significantly accelerates docking runtime without losing docking accuracies. Moreover, for a more general use of VINA-GPU, a heuristic way has been given out for determine a proper number of search steps for a given complex.

In the future, our work will focus on 1) leveraging the portability of OpenCL to accelerators such as CPUs and FPGAs 2) optimizing VINA-GPU by exploring fine-grained parallelism.

Acknowledgement

We acknowledge the support from Mr. Xianqiang Shi for providing the Huawei cloud service and Ms. Yemin Diao for offering us workplace.

References

(1) DiMasi, J. A.; Grabowski, H. G.; Hansen, R. W. Innovation in the pharmaceutical industry: new estimates of R&D costs. *Journal of health economics* **2016**, *47*, 20–33.

(2) Lengauer, T.; Rarey, M. Computational methods for biomolecular docking. *Current opinion in structural biology* **1996**, *6*, 402–406.

(3) Cherkasov, A.; Muratov, E. N.; Fourches, D.; Varnek, A.; Baskin, I. I.; Cronin, M.; Dearden, J.; Gramatica, P.; Martin, Y. C.; Todeschini, R., et al. QSAR modeling: where have you been? Where are you going to? *Journal of medicinal chemistry* **2014**, *57*, 4977–5010.

(4) Golbraikh, A.; Shen, M.; Xiao, Z.; Xiao, Y.-D.; Lee, K.-H.; Tropsha, A. Rational selection of training and test sets for the development of validated QSAR models. *Journal of computer-aided molecular design* **2003**, *17*, 241–253.

(5) Morris, G. M.; Huey, R.; Lindstrom, W.; Sanner, M. F.; Belew, R. K.; Goodsell, D. S.; Olson, A. J. AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *Journal of computational chemistry* **2009**, *30*, 2785–2791.

(6) Trott, O.; Olson, A. J. AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry* **2010**, *31*, 455–461.

(7) Ravindranath, P. A.; Forli, S.; Goodsell, D. S.; Olson, A. J.; Sanner, M. F. AutoDockFR: advances in protein-ligand docking with explicitly specified binding site flexibility. *PLoS computational biology* **2015**, *11*, e1004586.

(8) Zhang, Y.; Sanner, M. F. AutoDock CrankPep: combining folding and docking to predict protein–peptide complexes. *Bioinformatics* **2019**, *35*, 5121–5127.

(9) Zhang, Y.; Sanner, M. F. Docking flexible cyclic peptides with AutoDock CrankPep. *Journal of chemical theory and computation* **2019**, *15*, 5161–5168.

(10) Santos-Martins, D.; Eberhardt, J.; Bianco, G.; Solis-Vasquez, L.; Ambrosio, F. A.; Koch, A.; Forli, S. D3R Grand Challenge 4: prospective pose prediction of BACE1 ligands with AutoDock-GPU. *Journal of computer-aided molecular design* **2019**, *33*, 1071–1081.

(11) Santos-Martins, D.; Solis-Vasquez, L.; Tillack, A. F.; Sanner, M. F.; Koch, A.; Forli, S. Accelerating AutoDock4 with GPUs and gradient-based local search. *Journal of Chemical Theory and Computation* **2021**, *17*, 1060–1073.

(12) Goodsell, D. S.; Sanner, M. F.; Olson, A. J.; Forli, S. The AutoDock suite at 30. *Protein Science* **2021**, *30*, 31–43.

(13) Su, M.; Yang, Q.; Du, Y.; Feng, G.; Liu, Z.; Li, Y.; Wang, R. Comparative assessment of scoring functions: the CASF-2016 update. *Journal of chemical information and modeling* **2018**, *59*, 895–913.

(14) Wang, Z.; Sun, H.; Yao, X.; Li, D.; Xu, L.; Li, Y.; Tian, S.; Hou, T. Comprehensive evaluation of ten docking programs on a diverse set of protein–ligand complexes: the prediction accuracy of sampling power and scoring power. *Physical Chemistry Chemical Physics* **2016**, *18*, 12964–12975.

(15) Bohacek, R. S.; McMartin, C.; Guida, W. C. The art and practice of structure-based drug design: a molecular modeling perspective. *Medicinal research reviews* **1996**, *16*, 3–50.

(16) Nikam, A.; Nara, A.; Paliwal, D.; Walunj, S. Acceleration of drug discovery process on GPU. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT). 2015; pp 77–81.

(17) Gorgulla, C.; Boeszoermenyi, A.; Wang, Z.-F.; Fischer, P. D.; Coote, P. W.; Das, K. M. P.; Malets, Y. S.; Radchenko, D. S.; Moroz, Y. S.; Scott, D. A., et al. An open-source drug discovery platform enables ultra-large virtual screens. *Nature* **2020**, *580*, 663–668.

(18) Mermelstein, D. J.; Lin, C.; Nelson, G.; Kretsch, R.; McCammon, J. A.; Walker, R. C. Fast and flexible gpu accelerated binding free energy calculations within the amber molecular dynamics package. 2018.

(19) Stone, J. E.; Hynninen, A.-P.; Phillips, J. C.; Schulten, K. Early experiences porting the NAMD and VMD molecular simulation and analysis software to GPU-accelerated OpenPOWER platforms. International conference on high performance computing. 2016; pp 188–206.

(20) Kaufmann, M. M.; Boston, G. Computing Gems Emerald Edition (Applications of GPU Computing Series). 2011.

(21) LeGrand, S.; Scheinberg, A.; Tillack, A. F.; Thavappiragasam, M.; Vermaas, J. V.; Agarwal, R.; Larkin, J.; Poole, D.; Santos-Martins, D.; Solis-Vasquez, L., et al. GPU-accelerated drug discovery with docking on the summit supercomputer: porting, optimization, and application to COVID-19 research. Proceedings of the 11th ACM international conference on bioinformatics, computational biology and health informatics. 2020; pp 1–10.

(22) Fan, M.; Wang, J.; Jiang, H.; Feng, Y.; Mahdavi, M.; Madduri, K.; Kandemir, M. T.; Dokholyan, N. V. Gpu-accelerated flexible molecular docking. *The Journal of Physical Chemistry B* **2021**, *125*, 1049–1060.

(23) Ding, X.; Wu, Y.; Wang, Y.; Vilseck, J. Z.; Brooks III, C. L. Accelerated CDOCKER with GPUs, parallel simulated annealing, and fast Fourier transforms. *Journal of chemical theory and computation* **2020**, *16*, 3910–3919.

(24) Imbernón, B.; Serrano, A.; Bueno-Crespo, A.; Abellán, J. L.; Pérez-Sánchez, H.; Cecilia, J. M. METADOCK 2: a high-throughput parallel metaheuristic scheme for molecular docking. *Bioinformatics* **2021**, *37*, 1515–1520.

(25) Shin, J. H.; Kim, J.; Chae, J.; Yun, S. J. GPU-accelerated autodock vina: Viking. 2020.

(26) Solis-Vasquez, L.; Santos-Martins, D.; Tillack, A. F.; Koch, A.; Eberhardt, J.; Forli, S. Parallelizing Irregular Computations for Molecular Docking. 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3). 2020; pp 12–21.

(27) Kannan, S.; Ganji, R. Porting autodock to CUDA. IEEE Congress on Evolutionary Computation. 2010; pp 1–8.

(28) Munshi, A.; Gaster, B.; Mattson, T. G.; Ginsburg, D. *OpenCL programming guide*; Pearson Education, 2011.

(29) Garland, M.; Le Grand, S.; Nickolls, J.; Anderson, J.; Hardwick, J.; Morton, S.; Phillips, E.; Zhang, Y.; Volkov, V. Parallel computing experiences with CUDA. *IEEE micro* **2008**, *28*, 13–27.

(30) Nocedal, J.; Wright, S. *Numerical optimization*; Springer Science & Business Media, 2006.

(31) Schäling, B. *The boost C++ libraries*; Boris Schäling, 2011.

(32) Santos-Martins, D.; Solis-Vasquez, L.; Tillack, A. F.; Sanner, M. F.; Koch, A.; Forli, S. Test set of 140 complexes for AutoDock-GPU. 2020.

(33) Hartshorn, M. J.; Verdonk, M. L.; Chessari, G.; Brewerton, S. C.; Mooij, W. T.; Mortenson, P. N.; Murray, C. W. Diverse, high-quality test set for the validation of protein-ligand docking performance. *Journal of medicinal chemistry* **2007**, *50*, 726–741.

(34) Li, Y.; Han, L.; Liu, Z.; Wang, R. Comparative assessment of scoring functions on an updated benchmark: 2. Evaluation methods and general results. *Journal of chemical information and modeling* **2014**, *54*, 1717–1736.

(35) Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The protein data bank. *Nucleic acids research* **2000**, *28*, 235–242.

(36) Hill, M. D.; Marty, M. R. Amdahl's law in the multicore era. *Computer* **2008**, *41*, 33–38.

Appendices

Table 1: Complete 140 complexes and their corresponding best *search_steps* determined

PDBid	N_atom	N_rot	best search steps	PDBid	N_atom	N_rot	best search steps
5tim	5	0	1	1jla	28	7	1
1aet	6	0	1	1r58	28	10	10
1ac8	8	0	1	1t40	29	6	1
2j5s	11	2	1	1yvf	29	6	10
1w1p	12	0	1	1unl	29	7	1
1n2j	12	5	1	1xoz	30	1	1
1p2y	13	1	1	1l7f	30	8	1
2rdr	13	5	1	1vcj	30	10	10
1lrh	14	2	1	1opk	31	4	1
1tni	14	5	1	1m2z	31	5	1
1jd0	15	2	1	2br1	31	7	1
1n1m	15	3	1	2bsm	31	7	1
1uwc	15	4	1	1z95	31	8	1
1hww	16	3	1	1qi0	31	12	10
1q4g	17	3	1	1a30	31	13	10
1tt1	17	4	1	3l4u	31	17	10
1of6	17	5	1	1s3v	32	6	1
1x8x	17	5	1	1hwi	32	10	10
1hnn	18	2	1	2bm2	33	7	10
1r9o	18	3	1	1s19	33	8	1
1n2v	18	3	1	1v0p	33	8	1
1sq5	18	8	1	1r1h	33	10	1

4elm2	18	14	1	1oyt	34	4	1
2ifb	18	14	10	1hvy	34	9	10
1uou	19	2	1	1pmn	35	6	1
1hq2	19	2	1	1sj0	36	8	1
1tow	19	4	1	3oe5	36	11	10
1l2s	20	4	1	1lpz	37	6	10
1w2g	20	4	1	1y6b	37	8	10
1sg0	20	5	1	1uml	37	11	1
1xm6	20	5	1	3bpc	37	15	10
3uex	20	16	10	1ywr	38	5	1
1of1	21	4	1	1mzc	38	8	10
1ig3	21	6	10	10gs	38	14	10
1gpk	22	1	1	3coy	39	11	10
1j3j	22	3	1	2x97	39	13	1
1p62	22	4	1	1os0	39	13	30
1tz8	22	6	1	4tmn	39	14	30
1u1c	22	7	1	1t46	40	6	10
1mmv	22	8	10	3bkk	40	14	1
1u4d	23	0	1	3ov1	41	12	20
1sqn	23	2	1	2xhm	42	14	10
1yv3	23	2	1	1h23	42	15	10
4fev	23	3	1	4djr	43	15	10
2brt	23	4	10	7cpa	43	15	10
1k3u	23	6	10	3l3n	43	16	1
1q41	24	1	1	1w3l	43	16	10
1t9b	24	3	1	2r23	43	19	20

1ia1	24	3	1	3s8o	44	12	10
1hp0	24	5	1	5kao	44	15	1
1nav	24	6	1	1sln	44	17	10
1meh	24	7	1	2d1o	44	23	10
1gkc	24	8	10	1kzk	45	11	10
3kwa	24	13	1	3zso	45	13	10
1gm8	25	4	1	5wlo	46	10	10
1yqy	25	5	10	1ygc	47	13	20
1v48	25	6	10	1u33	47	18	1
1jje	25	7	10	4gid	53	17	10
1v4s	26	4	1	3utu	54	16	30
1ke5	26	4	1	1hfs	54	18	10
1q1g	26	5	1	2xy9	54	18	10
1g9v	26	6	1	3pww	55	14	10
1lbk	26	11	10	1u1b	55	16	50
2wbg	26	11	10	3cyx	56	14	20
1owe	27	3	1	1jyq	60	20	30
1oq5	27	5	1	3drf	63	26	1
3tmn	27	7	1	2vaa	87	32	30
1xoq	27	7	1	2er7	89	32	100
1r55	27	9	10	4er4	93	30	80
1n46	28	5	1	3er5	108	31	10

Table 2: VINA-GPU docking results (in score) using the predicted *search_steps*

PDBid	N_atom	N_rot	predicted search steps	VINA score	score criterion	VINA-GPU score

1aet	6	0	1	-4	-3	-4
2j5s	11	2	1	-5.9	-4.9	-5.9
1p2y	13	1	1	-7.3	-6.3	-7.3
1jd0	15	2	1	-6	-5	-5.9
1of6	17	5	1	-8.4	-7.4	-8.3
1n2v	18	3	1	-7.2	-6.2	-7.1
2ifb	18	14	4	-7.4	-6.4	-6.3
1w2g	20	4	1	-9	-8	-9
1of1	21	4	1	-9.1	-8.1	-9
1mmv	22	8	3	-7.4	-6.4	-6.9
1u4d	23	0	1	-9.1	-8.1	-9.1
1yv3	23	2	1	-12.8	-11.8	-12.8
1k3u	23	6	2	-9.9	-8.9	-10.2
1meh	24	7	3	-7.5	-6.5	-7.2
1yqy	25	5	2	-9.4	-8.4	-10
2wbg	26	11	5	-8.6	-7.6	-8.3
1xoq	27	7	4	-9.8	-8.8	-9.8
3tmn	27	7	4	-7.1	-6.1	-6.8
1unl	29	7	4	-9.2	-8.2	-9.2
1opk	31	4	3	-12.7	-11.7	-12.6
1z95	31	8	5	-10.4	-9.4	-10.5
1a30	31	13	6	-7.1	-6.1	-7.1
2bm2	33	7	5	-9	-8	-8.8
3oe5	36	11	7	-10.1	-9.1	-9.8
1y6b	37	8	6	-9.9	-8.9	-9.5
1uml	37	11	7	-9.7	-8.7	-9.4

1mzc	38	8	6	-9.9	-8.9	-9.8
1os0	39	13	8	-9.4	-8.4	-8.8
1h23	42	15	9	-10.3	-9.3	-9.6
1w3l	43	16	10	-8.2	-7.2	-8.2
3s8o	44	12	9	-7.8	-6.8	-6.8
1u33	47	18	11	-7.9	-6.9	-7.8
4gid	53	17	13	-9.1	-8.1	-11.8
3pww	55	14	13	-11.3	-10.3	-10.9
2er7	89	32	25	-9.2	-8.2	-8.2

Table 3: VINA-GPU docking results (in RMSD) using the predicted *search_steps*

PDBid	N_atom	N_rot	predicted search steps	VINA RMSD	VINA-GPU RMSD
1aet	6	0	1	1.492192068	**1.491190799**
2j5s	11	2	1	0.495458649	**0.491982077**
1p2y	13	1	1	1.440848388	**1.458771717**
1jd0	15	2	1	1.726833885	**1.755009478**
1of6	17	5	1	0.863880643	**0.90604746**
1n2v	18	3	1	1.175179608	**1.695992024**
2ifb	18	14	4	1.769895854	**1.737840854**
1w2g	20	4	1	0.694794322	**0.727399684**
1of1	21	4	1	0.595048097	**0.647161752**
1mmv	22	8	3	0.672653092	**1.255357446**
1u4d	23	0	1	0.563029075	**0.520049454**
1yv3	23	2	1	0.449715369	**0.346917231**
1k3u	23	6	2	0.561363712	**1.023878198**

1meh	24	7	3	0.869477333	**1.472515054**
1xoq	27	7	4	0.746914369	**1.660166669**
1unl	29	7	4	1.016373659	**0.487551518**
1opk	31	4	3	0.684433696	**1.864573712**
1z95	31	8	5	1.117130959	**0.965139719**
1a30	31	13	6	1.745264921	6.497142511
2bm2	33	7	5	0.466084626	**1.098855879**
3oe5	36	11	7	0.629995635	**0.606515686**
1y6b	37	8	6	1.205263782	**0.664081178**
1uml	37	11	7	1.618065044	**1.704511605**
1mzc	38	8	6	1.243470017	**1.064932491**
1os0	39	13	8	1.445256024	**1.264158441**
1w3l	43	16	10	1.136399127	**0.852252296**
3s8o	44	12	9	0.937338859	4.65885539
1u33	47	18	11	1.682015502	**1.699537603**
3pww	55	14	13	0.857316553	**1.888033922**

Table 4: Example of 2bm2_config.txt file

```
receptor = ./2bm2_protein.pdbqt
ligand = ./2bm2_ligand.pdbqt
center_x = 40.415
center_y = 110.986
center_z = 82.673
size_x = 24.0
size_y = 26.25
size_z = 22.5
```

Table 5: Models (PDBid: 1hvy) generated from AutoDock VINA with scores and RMSDs

MODEL	score	RMSD
model 1	-10.1	8.523
model 2	-9.6	0.770
model 3	-9.4	2.160
model 4	-9.2	1.773
model 5	-9.1	8.616
model 6	-9.1	1.599
model 7	-9.0	8.426
model 8	-9.0	8.680
model 9	-8.7	1.888