The Pennsylvania State University
The Graduate School

ACCELERATE THE DOCKING APPLICATION ON HETEROGENEOUS COMPUTING SYSTEM

A Thesis in
Computer Science and Engineering
by
Mengran Fan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2020

Abstract

With the recent development of structural biology, the bottlenecks of drug discovery have changed from the limited number of known protein structures to finding the stable docked molecular for the target proteins. Compared to directly adopt molecular docking chemical experiments, computational molecular docking offers a lower cost and less time-consuming method. However, due to the sequential algorithm designs and large memory consumption problem, most of the computational molecular docking applications are not implemented with the high-performance computing techniques, thus not fully take the benefits of computer cluster and graphics processing unit (GPU) architecture. In this thesis, we proposed a generalized molecular docking parallel strategy for Message Passing Interface (MPI) and GPU accelerating. Specifically, We maintain fault-tolerant and loading balance inside the MPI version. The GPU version carefully optimizes the main time-consuming parts targeting GPU architecture, including the reduction part, energy calculation, and memory access part. To evaluate the effectiveness of the proposed strategy, we select MedusaDock, one of the most famous computational molecular docking software, as an example to accelerate molecular docking processes.

For original version MedusaDock, the docking process typically takes around a week to process one iteration on the PDBBind dataset with nearly 3900 protein-ligand pairs. Even worse, only a small portion of the protein-ligand pairs can find their docked orientation and more pairs still need much more iterations to get converge.

By applying the parallel strategy on MedusaDock, our experiment result shows that both MPI and GPU version MedusaDock achieve superior performances than the original MedusaDock. The MPI version MedusaDock running on 9 nodes with 20 processes is 83.9x times faster than the original MedusaDock. If running on multiple nodes on the cluster the total time consumption can correspond reduced depending on the node number. The GPU version achieves around 1.8x times improvement on overall performance.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to take this opportunity to express my sincere thanks and appreciation to many people who have helped me throughout my graduate program study and research.

Firstly, I would like to sincerely thank my advisor Professor Mahmut Kandemir, for providing this wonderful opportunity to work with him, for his professional, patient, invaluable advice and encouragement throughout all my graduate studies.

I also want to express my gratitude to Professor Kamesh Madduri, for his detail guidance, patient conducting and inspiration during my research.

Many thanks to Professor Nikolay Dokholyan, Dr. Jian Wang and Professor Mehrdad Mahdavi, who spend lots of time discussing research with me, providing valuable feedback and guidance.

I would also like to thank all the members of the MDL and HPCL lab for all the help and encouragement.

Finally, I would like to thanks my family for the constant support during my studies.

Chapter 1 Introduction

Over the past two decades, finding out the biological active compound processes usually suffer from long periods and intolerable high costs. Even with high-throughput screening technologies, development of a typical small-molecule drug usually takes years and costs millions of dollars thus not meet the requirements of the modern drug development. [1–3]. With the recent development of infrared spectroscopy, X-ray crystallography, nuclear magnetic resonance (NMR) spectroscopy and cryogenic electron microscopy (Cryo-EM) technologies [4,5], more and more protein and ligand structure details have been discovered, the bottlenecks of drug discovery have changed from the limited numbers of known protein structures to finding the stable docked molecular for the target proteins [6–8]. The computational molecular docking, which offers an efficient and low-cost method to find out the target small molecular, has become a very important topic in the structure-based drug discovery area recently. The discovery of the huge amounts of protein and ligand structure provides more data candidates for the computational molecular docking. Given the known protein and molecular structure, computational molecular docking predicts the binding affinities for the complex to identify the active ingredients. In detail, it simulates all the potential conformations of the small molecular ligands and protein side chain, calculates the estimating the molecular complexes' energy and outputs the binding poses. After that, by virtual screen all the binding poses of different ligand candidates, the active ingredients are identified.

Although computational molecular docking is far more efficient than the traditional chemical experimental method, in the high-performance computation perspective, the computational molecular docking software not fully take advantage of modern computer architecture. The computational molecular docking algorithms are hard to be implemented in modern computer architecture due to their sequential designed algorithm and large memory consumption. For example, the Monte Carlo and genetic algorithm searching algorithms are sequentially designed. After parallelization every generated pose needs to save its conformation information and its energy array, memory consumption will rise in response to the scale of the parallelization. Currently, most popular docking software such as GOLD [9], DARWIN [10], MCDOCK [11] and MedusaDock [12, 13] are implemented to target the normal CPU architecture and don't taken the advantage of the massive parallelization. Only Auto-Dock Vina has been published a multilevel parallel version, which implemented by MPI and OpenMP. In this thesis, we propose a general parallel strategy and optimization methods for docking software. The parallel strategy breaks down the sequential of the algorithm while guarantees the correctness of the algorithm. The optimization methods accelerate the memory access by leveraging different levels of GPU memory device, accelerate reduction by fully using the synchronization inside thread block and share memory, accelerate energy calculation by using the grid data structure to localize traversing atoms.

To evaluate the effectiveness of the proposed strategy, we select MedusaDock, one of the most famous computational molecular docking software, as an example to accelerate molecular docking processes. Comparing with other docking software, it repacks the side-chain of the protein during rand move the ligand and carefully considers all the possible conformations of the whole complex. MedusaDock also has a more effective energy function, which can predict the affinities of the docking complex more accurately. MedusaDock targets to dock the ligand into protein to get protein-ligand complex with the minimum total energy. To find out the best ligand, for different proteins, millions of ligands need to do dock with every different protein. Currently, for nearly 3900 protein-ligand pairs of ligand and protein datasets, total time consuming is around 134 hours, which means need 5.6 days to get all the results. Even though, only a few data pairs can find its docked conformation. Most data pairs need hundreds of and thousands of more iterations to find its docked conformation. Moreover, when the number of pairs goes up, the performance could drop dramatically. Considering the huge total time consumption of dealing with thousands of or millions of protein and ligand pairs, parallel the processing between the pairs can help the performance in this situation. We plan to take the benefits of cluster and GPU architecture to optimize the MedusaDock by using MPI and CUDA. We implement an MPI version of the MedusaDock, with loading balance under consideration. The result shows, when running on 9 nodes with 20 processes, it has 83.9x speedup. When the number of nodes in the cluster goes up, the performance will relatively improve. We have done a detail characterization of the MedusaDock software, the analysis demonstrates that random moving the ligand rotamers is the bottleneck of the software. We also redesign the searching algorithm to remove the data and control dependency, for parallel the searching process on the GPU architecture. Furthermore, we implement several GPU specific optimization methods to guarantee the software take full benefits of the GPU resource and efficiently run on the GPU architecture. Our experiment result shows, the GPU version overall performance improves 1.8x. Since the GPU version performance different case by case, the highest speedup for one protein-ligand pair is around 4x.

The rest of the thesis is organized as follows: Chapter 2 introduced the background of Molecular docking software and GPU architecture, summarized all related works. Chapter 3 characterized and analyzed the MedusaDock algorithm and proposed our motivation. Chapter 4 explained the parallel strategy of the MPI version. Chapter 5 introduced the parallel strategy of the GPU version, explained all the optimization methods implemented targeting on GPU architecture. Chapter 6 presents and analyzed the experimental results of the GPU version. Chapter 7 draws the conclusion and future research directions.

Chapter 2 | Background and Related Work

2.1 Computational Molecular Docking

The computational molecular docking, which offers an efficient and low-cost method to find out the target small molecular, has become a very important topic in structure-based drug discovery area in recent years. It finds out the minimum energy binding pose and predicts the best drug candidates among millions of small molecules.

All the energy-based computational docking approaches consist of searching algorithms and score functions. These components make the energy-based docking approaches conventionally feasible in structure-based Docking area [6]. One major part of the energy-based docking approaches is the searching algorithms. The general step of energy-based docking approaches always starts with sampling all the potential conformations of the docking molecular. Then various searching algorithms are adopted to find out the best conformation, by traversing the sample results as less as possible. Since the huge amount of the sample result makes the searching step to be very time consuming, many searching algorithms have been explored because of the purpose of stochastic global optimization, by different well-known docking applications. Genetic algorithm is adopted by GOLD [9], Auto-Dock Vina [14], DARWIN [10] etc. It is good at finding the region, where extremes are located but difficult to find the precise location. Fragment-based methods is adopted by FlexX [15] and Dock [16]. It could have more hydrophilic hits, but limited chemical space and low structural diversity. Monte Carlo algorithm is adopted by ProDock [17], MCDOCK [11] and MedusaDock [12, 13].

Another major part of the energy-based docking approaches is energy evaluation, also widely known as score function. It usually estimates the complex energy to predict the affinity between the two docking molecules. The physics-based score function can be divided into Force Fields, Solvent models. Force Fields based computes the interactions between the atoms of protein and ligand. It is bad at large ligand, since the force field errors would accumulate, and is very computationally expensive. Solvent models try to estimate the torsion entropy of the ligand, but it is difficult to estimate potential energy and locate the related parameters. Physics-based score function is implemented by CHARMM [18], AMBER [19]. Empirical methods based on score function circumvents the speed and sampling problems. It sums up the important energetic factors. Since its parameter training using known protein-ligand binding structure, it could be overtrained and it has low transfer-ability to structurally distinct targets. This method is adopted by ChemScore [20] , GlideScore [21], AutoDock [22]. Knowledge-based methods based score function is very similar to the empirical methods. It based on the Inverse Boltzmann statistic principle. It derives the desired pairwise potentials from three-dimensional structures. It assumes that the frequency of different atom pairs in different distances is related to the interaction of two atoms and converts the frequency into the distance-dependent potential of mean force. The limitation of this method is difficult to locate the reference state. This method is implemented by PMF [23], Bleep [24], DrugScore [25]. MedusaDock uses physical interaction model. Its score function consists of Rosetta force field, VDW interaction model and parameters, an explicit hydrogen-bonding model and EEF1 pairwise implicit solvent model. To relieve the computation-intensive, MedusaDock takes the grid-based calculation.

MedusaDock starts with a stochastic rotamer library of ligands generation, then shifts and rotates the rotamer into different poses. Based on the Monte Carlo algorithm and score function MedusaDock tries to search for the best-fit poses within the docking boundary box. During the searching process, not only the ligand but also the protein will repack its side-chain, to imitate the realistic protein-ligand docking procedure. Also to optimize the searching algorithm and reduce the computation from score function, MedusaDock separates the searching step into coarse docking and fine docking. In coarse docking, it can quickly find out the near best poses and then carefully verify and search around during the fine docking step. Which largely reduces the search and sampling time.

2.2 Acceleration of the Computational Docking Approaches

Most popular computational docking software are only implemented target CPU architecture. Only a few of them have done parallel acceleration targeting normal CPU node, cluster and GPU architecture.

Auto-Dock Vina is a popular open-source molecular docking software, and it keeps updating its version to optimize the software as much as possible by paralleling various aspects of the program [26]. By using OpenMP and MPI, its official version established a multilevel parallelism mechanism. Auto-Dock Vina uses OpenMP to implement the multi-thread Lamarckian Genetic Algorithm, which is used for searching ligand conformations. Auto-Dock Vina uses OpenMP to accelerate the individual docking task. And it takes the benefits of MPI to parallel doing virtual screens. The virtual screening process, which runs Auto-Dock Vina on different protein-ligand data pairs to locate the leading compound, is a computational way to find out the leading compound by searching the ligand libraries. Since currently, auto-Dock is the most popular docking software, there are multiple works leverage GPU architecture to accelerate Auto-Dock. Micevski D. et al mapped GPU thread to ligand atoms directly [27], since the number of the ligand atoms is limited, this method has a very low utilization of the GPU architecture. Kannan S. et al paralleled genetic algorithm(GA), and for score function, they paralleled the atom pairs level, they got 10x to 47x speedup [28]. But they did not have local search, which meant less degree of freedom, and they did not evaluate the accuracy of the result. Their result only based on the speedup kernel without including CPU host data prepare and transfer time. Serkan A. et al paralleled GA and got around 14x speedup [29]. They only test on three protein-ligand pairs. No GPU architecture relevant optimization methods included.

There is some other docking software, which is less popular but did try to accelerate the algorithm. VSDocker is similar to Auto-Dock Vina, it also utilizes the MPI to parallel process virtual screening process on the cluster [30]. But VSDocker doesn't have intra level parallelism on one CPU node. Most popular docking software takes the benefits of the MPI to accelerate the code. PIPER had paralleled the FFT correlation process and the score function on GPU architecture, it can get around 4.8x speedup. Most of its accelerations come from cuFFT [31]. There is also some other docking software accelerated on FPGA [32, 33]. Most such GPU accelerating work only tested on a very small portion of picked out protein-ligand pairs, and their performance result only based on the optimized kernel, not including the whole software execution time. Moreover, because docking software implementation is quite different, for example, their searching algorithm and score function are various largely. Therefore, it makes the GPU parallel strategy and optimization method various case by case.

2.3 MPI and MPICH

Message Passing Interface (MPI) is a standardized and portable message-passing standard to function on a wide variety of parallel computing architectures. The basic operation of MPI is sending and receiving messages, automatically combined communication and synchronization. MPI can be used to parallel on individual computer, cluster and heterogeneous network. It is a very flexible interface to parallel the program on various platforms. The method provided by MPI is very limited. Mainly consists of sending messages, receiving messages, broadcast and Allreduce.

MPICH is open-source software and a portable implementation of MPI. It is a very popular version of MPI, also used as the foundation for the vast majority of MPI implementations

2.4 GPU architecture and CUDA

The graphics processing unit (GPU) has been becoming an integral part of today's mainstream computing systems. The modern GPU is not only a powerful graphics engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth that substantially outpaces its CPU counterpart. GPU has been known to provide high performance and energy efficiency for a variety of applications in different domain, for example computer vision [34,35], graphics [36], machine learning [37], Biomedical [38] etc. CUDA and OpenCL are two different models for GPU programming [39]. The OpenCL standard offers a common API for program execution on systems composed of different types of computational devices such as multi-core CPUs, GPUs, or other accelerators [40]. On the contrary, CUDA is a parallel computing platform and application programming interface (API) model specific to NVIDIA GPUs. It also includes cuBLAS, cuFFT, CUTLASS, and cuDNN these optimized algorithm libraries, which help the program developer to easily take full benefits of the NVIDIA GPU architecture resource.

Chapter 3
Motivation and Application Analysis

3.1 Motivation

MedusaDock is to dock the ligand into protein to get protein-ligand complex with the minimum total energy. To find out the best ligand, for different proteins, millions of ligands need to be docked with every different protein. Currently, for nearly 3900 protein-ligand pairs of ligand and protein datasets, the total time consuming is around 134 hours. The data show that it takes 5.6 days to get all the results. When the number of pairs goes up, the performance might drop dramatically. Consider the huge total time consumption of dealing with thousands of or millions of protein and ligand pairs, parallel the processing between the pairs can help to improve the performance in this situation.

As Figure 3.1 shows below, for one ligand and protein docking, the time consuming is from 7sec to 49min. There are 13.7% pairs of ligand-receptor and their docking time is larger than 200sec. So improving the performance inside one pair processing is necessary. We plan to implement one pair docking parallel on GPU, to reduce the one pair docking time.

Therefore, based on the current results and analysis, improving the performance of the MedusaDock is meaningful work. Not only to get the result from a huge dataset but also to reduce the time for one pair of ligand-receptor docking from tens of minutes to a few minutes. Based on the theory of the MedusaDock, MPI and GPU architecture with its thousands of parallel threads can help to improve the performance largely. Therefore, the progress can divide into two steps: 1) improve one pair docking time, parallel MedusaDock on GPU; 2) parallel doing MedusaDock for virtual screen process.

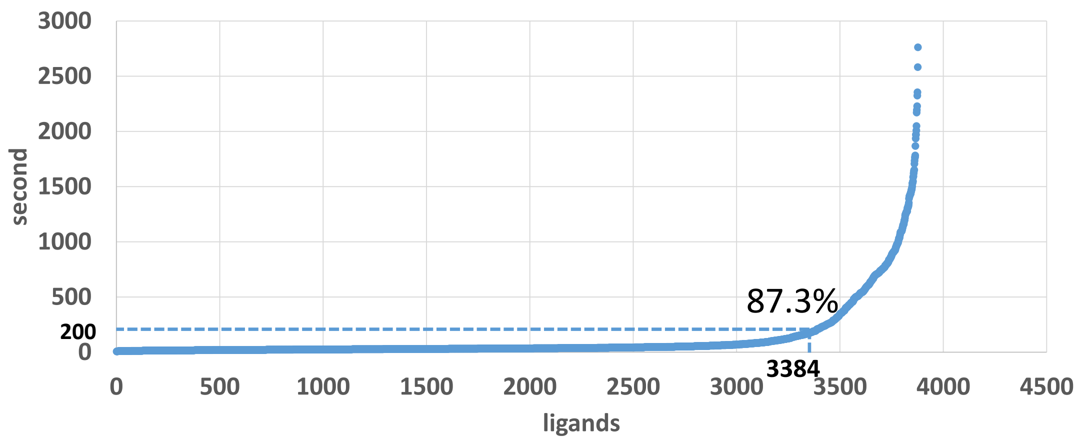


Figure 3.1. Time consuming of nearly 3900 protein-ligand pairs dataset.

3.2 Application Analysis and Profiling

3.2.1 Application Analysis

From the theoretically perspective, MedusaDock mainly consists of a searching algorithm and score function. The searching algorithm tries to efficiently explore the conformational space for docking. There are three main parts during searching the potential conformations: stochastic rotamer library of ligands generation, repack the side chain of the protein and rigid body docking. The first two steps are all preparing for the rigid body docking. Rigid body docking process explores the protein-ligand complex by keeping the structure of the molecules rigid. The purpose of the rigid body docking is to rapidly explore all the potential complex. Since the first two steps already try to consider all the flexibility of docking conformations. Comparing with other docking software MedusaDock has more flexibility on docking conformations by repacking the side chain of the protein. Although MedusaDock uses the Monte Carlo algorithm as the search algorithm, which has severe data and branch dependency problems. But, due to large amounts of the conformations, parallel processing the searching step on GPU architecture still will be our basic parallel strategy. The parallel strategy will be fully discussed in Chapter 5.

The score function is used to give a score based on the energy of the protein-ligand complex. And screen out several minimum energy complexes to predict the docking conformation. The lower energy complexes mean more stability of the protein-ligand complex. Although it can not predict the affinity between the ligand and the protein, by using this way, at least we can screen out the several highly possible potential ligand candidate from millions of candidates for the chemical experiment verification. Since the chemical experiment is very time consuming and money cost, the result of MedusaDock shows that the energy function performs well for predicting the protein-ligand docking conformation. It means the energy function has a positive correlation with the affinity of the protein-ligand complex.

MedusaDock uses a physical interaction model as a score function. The model is comprised of Rosetta force field [41], VDW interaction model and parameters, an empirical-based explicit hydrogen-bonding model and a knowledge-based EEF1 pairwise implicit solvent model. Its solvent model considering both hydrogen-bonding and desolvation effects. Since the score function needs to work with the searching algorithm to screen out the best conformation, this part needs to be paralleled processing with the searching algorithm on GPU architecture.

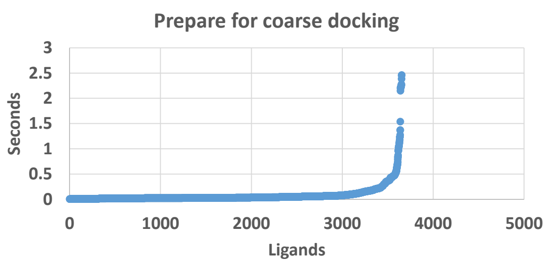
From the program structure perspective, MedusaDock can divide into three main steps: 1) prepare for coarse docking; 2) Coarse docking; 3) Fine docking.

Preparing for coarse docking mainly prepares STROLL(stochastic rotamer library of ligands). Since the small molecular could have multiple rotatable bonds, the ligand rotamers could be very flexible. To enumerate all possible rotamers of the ligand is not practical. Thus, MedusaDock uses a stochastic manner to generate the ligand rotamer library. And they use the same manner to repack the side chain of the protein, to guarantee both flexibility and simultaneous of the ligand and protein conformations. They proposed an efficient rotamer library, which sufficiently sampled the conformation space. They computed the Kabsch algorithm and compute the root-mean-square deviation (kRMSD) value for different STROLL generations and proved the kRMSD value < 2 Å to verify the sufficient of the library.

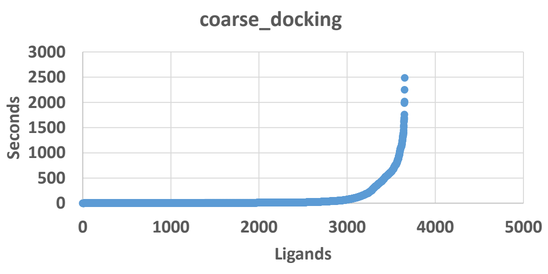
To accelerate the searching step MedusaDock separates the searching step into coarse docking and fine docking. MedusaDock uses Monte Carlo algorithm to do the search step. In coarse docking, it searches for the best-fit poses within the docking boundary box. It proposes N_c cluster centroids, which is used as typical initial ligand conformations, for each N_c cluster centroids, coarse docking is performed. During each docking process, the ligand and protein structure will keep rigid, so the complex does not need to be the lowest energy pose. But during the whole coarse docking process, multiple docking processes are performed, several low energy poses will be proposed. The lowest energy complex should be among these low energy poses. For the N_c coarse-docked poses, MedusaDock will sort and group similar poses. During this step, MedusaDock keeps calculating the RMSD value of low energy poses and removing similar poses. Similar poses will cost unnecessary expensive calculations of the next fine docking step. The coarse docking step finally chooses the top N_f (10% N_c) groups of lower energy poses for the next fine docking step. During the coarse docking step, the ligand rigid-body motion and the receptor side-chain rotamers are iteratively sampled.

For the fine docking step, both the ligand and the receptor side-chain rotamers are sampled simultaneously. In this step, they will change all the ligand conformations in each N_f group within 2 Å to enrich the ligand conformations. After the enrichment step, every N_f group will only keep the lowest energy complex. This step will finally get N_f minimized poses. The fine docking step tries to carefully verify and search around the conformation, which is proposed by coarse docking. This two-level searching strategy can largely reduce the search and sampling time.

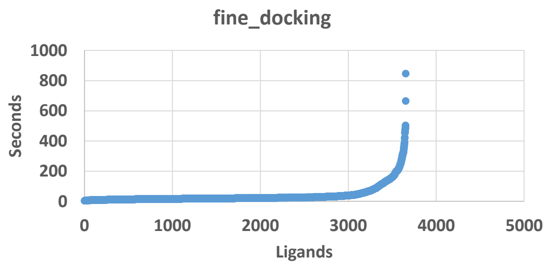
3.2.2 Application Profiling



(a) Prepare for coarse docking



(b) Coarse docking time



(c) fine docking time

Figure 3.2. Application performance analysis

Table 3.1. Top four time consuming examples in different part.

Prepare for coarse docking (s)	coarse_docking (s)	fine_docking (s)	Total_time (s)
2.27809	1987.13	497.509	2471.873
2.38414	2017.74	503.152	2512.497
2.45547	2252.32	664.319	2724.924
2.46173	2488.09	846.462	2985.653

As figure 3.2 and table 3.1 shows, the time-consuming estimation of the MedusaDock main steps lists below:
1) Prepare for coarse docking, mostly input cases spent less than 1sec. No pairs spent time more than 3 seconds. Most cases spend less than 1s. This step is not a time-consuming step.
2) Coarse docking, time-consuming is from 2sec to 41min. When the total time consumption is less than 30s, usually fine docking consumes the most time. When the total time consumption is more than 40s, usually coarse docking consumes 2x-5x time comparing with fine docking. Therefore, coarse docking is the main bottleneck of the performance. The time consumption reason of coarse docking comes from the complex structure of the ligand or receptor. Since both ligand rotamer and different conformation of the receptor side chain need to be explored in this step, when the searching space goes up, the calculation cost will increase. Some time-consuming cases could have several hundreds of ligand rotamers need to be explored, while some only have few.
3) Fine docking, time-consuming is from 3sec to 14min. Only 3.3% of docking pairs consume time more than 3 min in this step. This step has fewer iterations comparing with the coarse docking step, but it consumes more time for one iteration.

The time consuming of coarse docking and fine docking steps highly depends on the structure of the ligand and the receptor (receptor represents the target protein). The more complicated structure of the molecular means more ligand rotamers and more side-chain conformations can be generated for coarse docking and fine docking. This flexibility will largely increase the workload of exploring the searching space. These two steps become the bottleneck of the MedusaDock performance. Coarse docking is to search for the best-fit poses within the docking boundary box. At the end of the coarse docking, it needs to sort and group coarse-docked poses to find out the top N_f conformation groups with minimum energy first. In the fine docking step, both ligand and receptor side-chain rotamers are sampled simultaneously [12] for each conformation. And all the conformations will be carefully checked around during the fine docking. As the number of conformations inside every group goes up, the total conformations which need to be explored will exponentially grow. They consist of the time-consuming reasons.

Here, we explain the detailed causes of the time consumption inside these two steps. One major cause in coarse docking is the large number of ligand rotamers and their poses. Parallel processing of all the rotamers with different poses can help a lot with the performance. However, MedusaDock uses Monte Carlo algorithm to generate poses for each ligand rotamers. Which means data dependence exists between the poses.

Parallel Monte Carlo algorithm on GPU, has been studied in these [42–44] papers. For MedusaDock, we need to find out the best way to parallel. So that we can improve the performance as well, as improve the accuracy. Moreover, computing the kRMSD matrix inside coarse docking can also run in parallel to save time. To not miss the best pose for all rotamers, there is a step inside coarse docking, recursive checking previous rotamers' energy under the current best pose, which is one of the most time-consuming parts of the coarse docking. Moreover, this step has unavoidable data dependence. We plan to change the program itself taking advantage of the GPU architecture. Generate enough poses for each ligand rotamer without recursive check the previous one.

For the fine docking step, the whole step needs to repeat N_f times to find out the best result. N_f is the top few low energy rotamer groups, gotten from coarse docking. We can parallel this step to reduce the consumed time. Searching for similar poses is the most time-consuming step inside fine docking. This step will involve the Monte Carlo algorithm and reduction algorithm parallel on GPU.

Chapter 4 Parallel strategy and Optimization on GPU architecture

4.1 Parallel strategy

Monte Carlo algorithm inside the MedusaDock is to search the parameter space. It starts from a random or specific state, which includes the information of position, orientation, and conformation of the ligand and protein side chain. Then it makes random state changes, accepts up-hill moves with probability dictated by energy function. The algorithm will stop when it reaches the local minimum energy. This is the crucial time-consuming step in MedusaDock since it includes both rand move ligand rotamers and energy calculation of the binding complex.

The figure below shows three representative energy profiling for each iteration of the search algorithm. From the figure, we can find out all three cases have reached the global minimum inside two thousand steps, and show repeat energy value distribution after that. We profile around 50 random picked protein-ligand pairs in this experiment, except 1ec3 case, all the other cases have reached its global minimum and started to repeat the energy distribution at around one thousand iterations. So we magnify the one thousand iterations part to analyze more detail of the energy distribution. The second line of the pictures shows the magnified partial result of the first line. 1c70 case represents the most common distribution, since we do not need the lowest energy in the coarse docking step, we only need relevant low energy poses. Therefore most cases at least need 300 iterations to get its low energy pose. Some cases even need more iterations. Based on this result and algorithm analysis in the previous chapter, We propose our parallel strategy on GPU architecture below. The property of GPU architecture can accelerate this searching step.

The parallel processing search algorithm will include: parallel random starting the

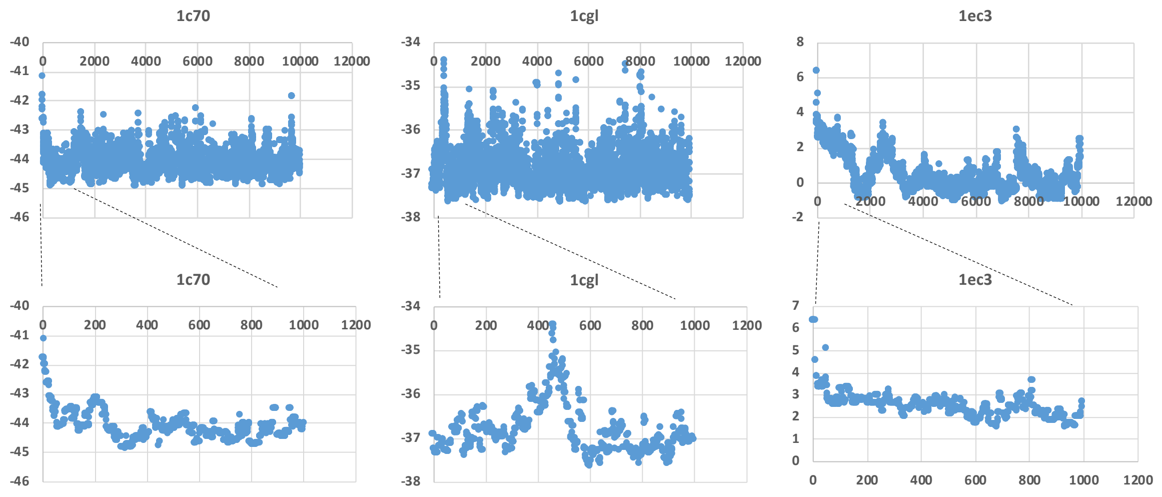


Figure 4.1. Energy profiling of the search algorithm. X-axis represents iterations of the search steps; Y-axis represents the energy of binding pose.

pose searching, parallel calculate the energy for each case and using a reduction algorithm to find out the minimum energy.

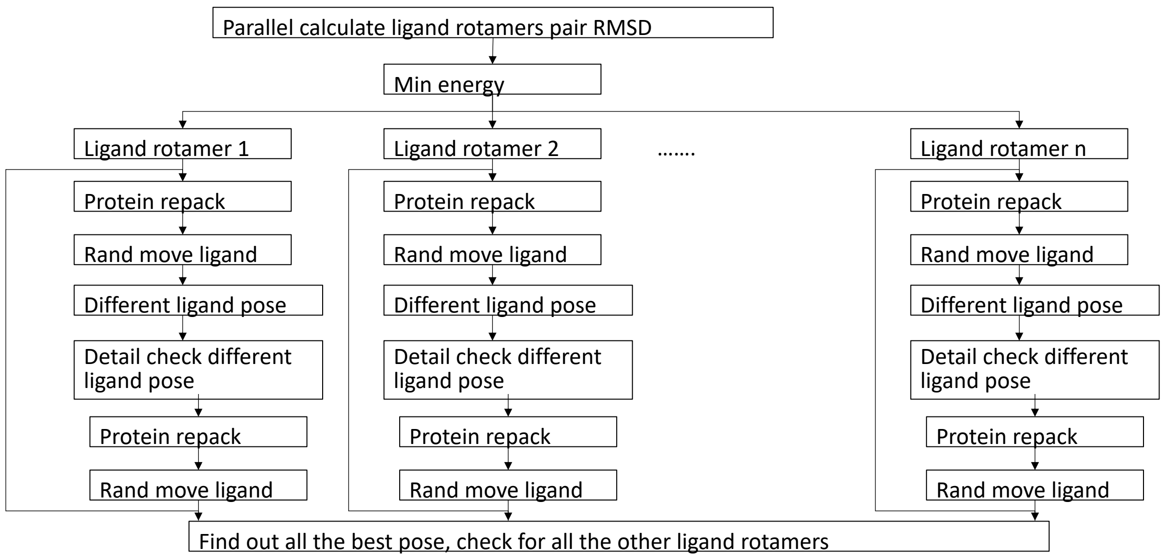
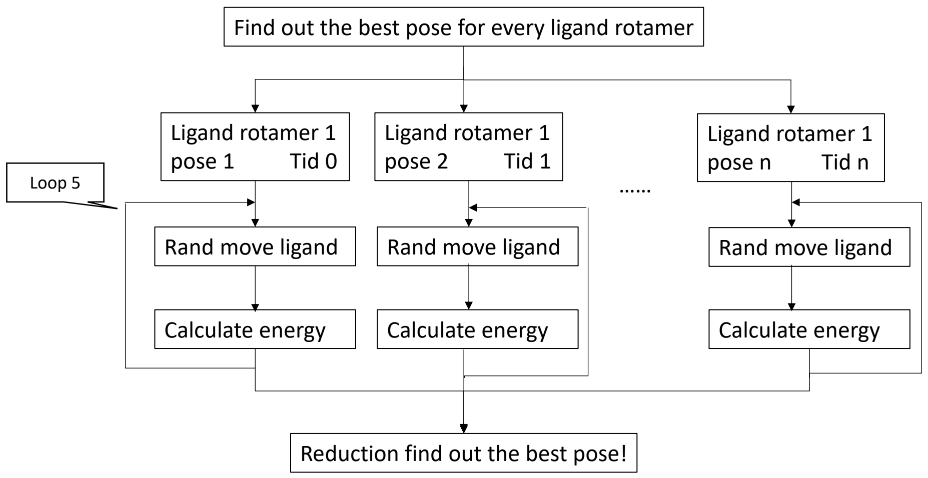


Figure 4.2. Program flow diagram.

Figure4.2 shows the program flow diagram of the original MedusaDock. We separate the coarse docking part and figure 4.3 shows the coarse docking parallelization flow diagram. To reduce the data transfer between the CPU host and GPU processor, we repack the side chain of the receptor first. Then parallel search the ligand poses for the specific ligand rotamer.



Find out the best pose for every ligand rotamer
Loop 5
Ligand rotamer 1 pose 1 Tid 0
Ligand rotamer 1 pose 2 Tid 1
......
Ligand rotamer 1 pose n Tid n
Rand move ligand
Rand move ligand
Rand move ligand
Calculate energy
Calculate energy
Calculate energy
Reduction find out the best pose!

Figure 4.3. Coarse docking parallelzation flow diagram.

We divide the search space into 8 x 8 x 8 small cubes. Then random initialize a start searching position in every cube. We initialize 7 x 8 x 8 x 8 threads for GPU implementation, the thread number is bounded by GPU architecture resources. Figure 4.4 shows the thread arrangement of the GPU kernel. We use Tesla P100 GPU architecture, so there are 56 SM units available in total. We group the thread into two levels, 2D gird level 8 x 7 and 2D block level 8 x 8. The reason why we only define 64 threads per thread block, it is because of the memory limitation of the GPU architecture.

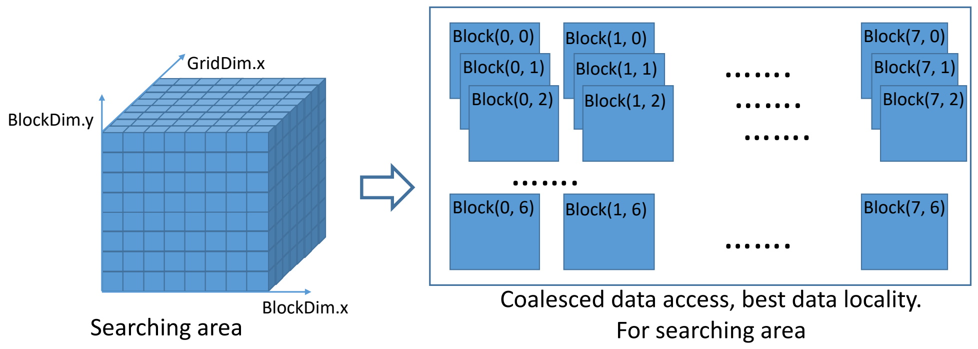


Figure 4.4. Thread map of the GPU kernel

We use 8 x 8 x 8 threads to map the small cube in the searching space, every thread takes the responsibility of one cube, random shift the ligand conformations from each start position. The thread mapping demonstrates in figure 4.4 is for best data locality and coalesced data access during the searching process. Every shift position will be rotated seven times to check different orientations of the conformation. Since 8 x 8 x 8 division of the searching space does not fine-grained enough. We still use the Monte Carlo algorithm to search from every start position for a few more steps.

Figure 4.5 demonstrates the parallel shift and searching process demo in a 2D 4 x 4 diagram. All the shift start in every cube could go in any direction since it is a 3D model in realistic. As the figure shows after several steps of exploring, there are several threads could reach the same local minimum points. Some other threads may get stuck in their local minimum points. By comparing these local minimum points, we try to get the global minimum points in the searching space. We test on 20 random picked protein-ligand pairs, based on our experiment result, after five Monte Carlo search steps, around 1/10 of total threads could reach the same local minimum point, and by comparing all the local minimum points, the local minimum point of the 1/10 threads is the minimum point. So we suppose that after five search steps, most cases could find its global minimum point.

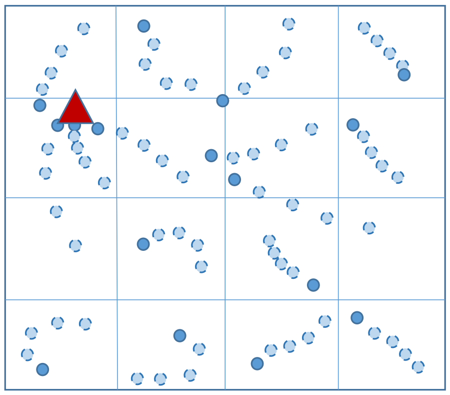


Figure 4.5. Parallel searching process

Since we need the energy of all the binding poses to find out the global minimum point, so we divide the coarse docking running on GPU into three parts as the figure 4.3 shows before. In the first part, random shift and rotate the ligand conformation in the searching space. In the second part, calculate the energy of the complex. In the third part, do reductions on GPU to get the global minimum energy pose.

4.2 Optimization methods on GPU architecture

4.2.1 Reduction

On GPU architecture synchronization is a very time-consuming operation, especially grid level synchronization. Therefore to minimize the grid level synchronization and even block level synchronization, we design a multi-level synchronization mechanism.

The first level is thread level synchronization. In our parallel search algorithm, we need to do five search steps for each initial point. To avoid grid synchronization after each search step, while keeping the lower energy pose. Every thread as figure 4.5 demonstrated will do its own search and keep the lower energy pose during the searching process, every post searching step will base on the previous lower energy pose inside the thread. By doing in this way, the five searching step only include random move the ligand conformation and complex energy calculation, the reduction step can be excluded. After all the searching steps, we only need one global synchronization, since the binding poses owned by each thread is already the minimum energy pose for each thread's local exploring area.

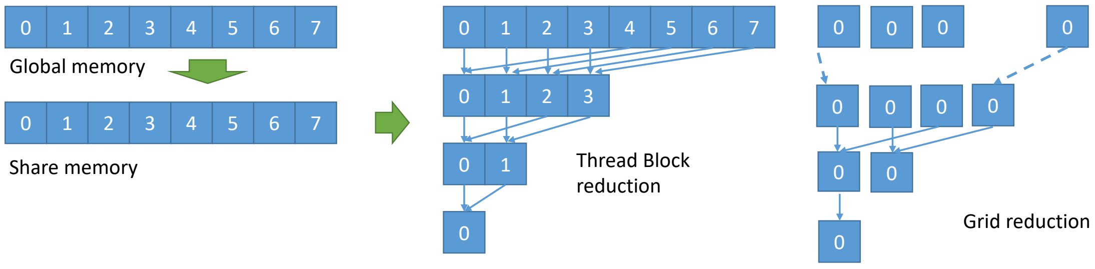


Figure 4.6. Thread block and grid level synchronization

Figure 4.6 shows the Second level synchronization, which consists of thread block level and grid level synchronization. To reduce the synchronization consumption across all the threads, we do thread block level synchronization first, to get the minimum energy pose inside every thread block. Then synchronize all the thread blocks to get the global minimum energy pose. The property of GPU architecture prohibits the register file from sharing across the threads, therefore only memory can be used to do reduction. The benefits of thread block level reduction are that we can take advantage of share memory to do the reduction. Share memory access is much faster than global memory access, and share memory can only be accessed by the thread inside one thread block. Therefore, we first copy the data to share memory and then do thread block level reduction. Then to synchronize across the grid, we need to copy the data back to global memory, since thread blocks can only communicate through global memory. Then we still copy the data back to share memory and use one thread block to do the final reduction. But before this step, we need to maintain data consistency. We first tried the Cooperative Group to synchronize across the grid for data consistency. Then we also tried to use one data allocation in global memory as a lock to manually synchronize across thread block. The experiment result shows that manual synchronization can have around 14% performance improvement compared to cooperative groups. So we keep the manual synchronization version.

4.2.2 Binding Pose Energy Calculation

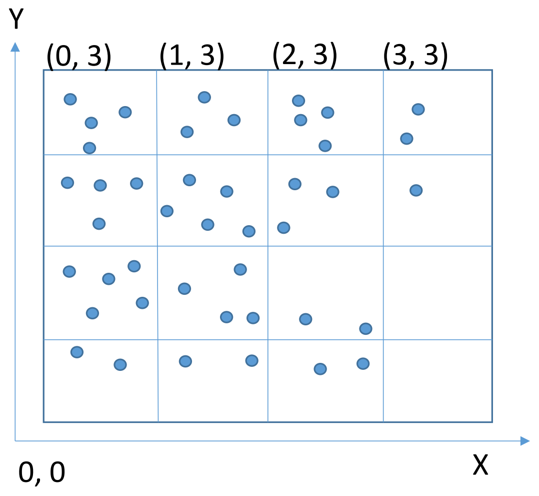


Figure 4.7. Atoms grid

Binding pose energy calculation includes all the atom pairs and proton pairs Van Der Waals force field calculation, solvent calculation and hydrogen bond energy calculation. In this part, if traverse all the atoms and protons in the receptor for every ligand atom and proton, there will be lots of unnecessary data access and such a huge amount of global memory access will also decrease the performance. Therefore, we build a 3D grid to accommodate the atoms and a 3D grid to accommodate the protons. Figure 4.7 demonstrates a atoms grid in 2D. Since the force field effect scope is limited, by the grid structure we can easily isolate the atoms, which inside the effect scope. For every atom inside ligand calculate its targeted grid area first, then by this coordinate information, we can largely reduce the force field effect scope and the global memory access.

After using the grid data structure, we need to isolate the hydrogen bond energy calculation out. Since the hydrogen bond needs to be established during the calculation, which will trigger data insert and delete inside the grid. For the attributes of the GPU architecture, before the GPU kernel launching, we need to copy data from CPU host to GPU device, in this step vector structure which including pointer can not be supported. So we transfer all the vector data structure to an array when copying to the GPU device, including grid data structure, which is implemented by vector. The cost of delete and insert for array is very expensive and every thread needs to have its own grid data structure, which largely increases memory cost. And we analyze 100 random picked cases, hydrogen bond part energy is only occupied less than 5% of the total binding pose energy. This part of the energy is not a crucial energy reason, therefore we isolate this part and put this part of the calculation to CPU host.

4.2.3 Memory Access Optimization

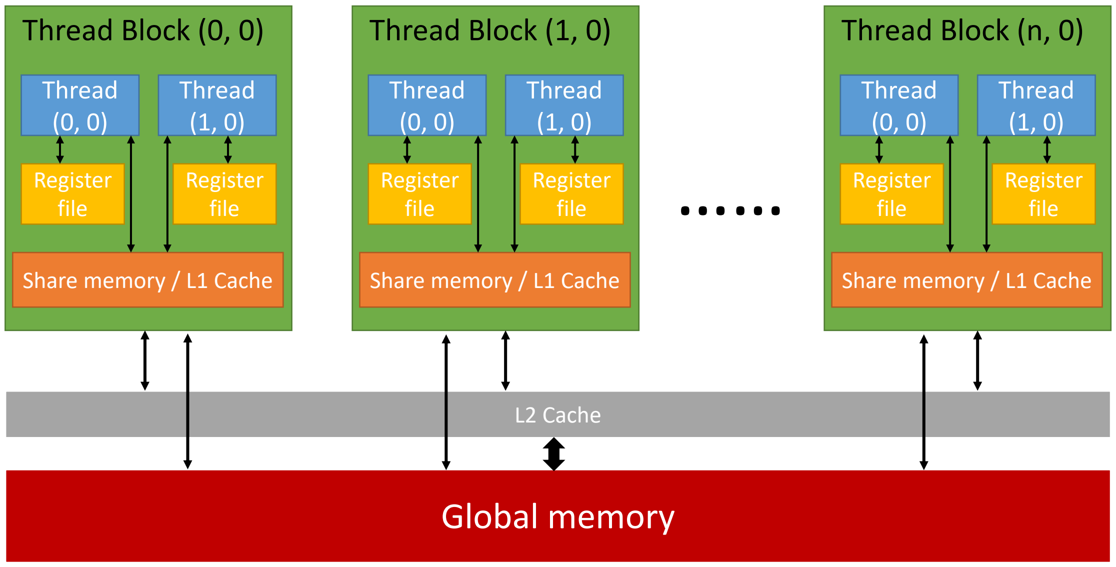


Figure 4.8. GPU architecture memory structure

During the implementation of the coarse docking GPU kernel, since the coarse docking process is memory intensive process, we fully take the benefits of the shared memory, register file and vectorized data access, to optimize the memory access performance. Figure 4.8 demonstrates the four hierarchies of the GPU architecture memory. Register file is private per thread and its size is limited per thread block. For Tesla P100

GPU architecture, there is 256 KB register file per SM. Register file is the fastest memory storage, compared with share memory and global memory. These three memory structures are the only programmable memory in GPU architecture, L1 and L2 cache is not programmable. Therefore, we put protons of the ligand on register file, this part of data same with atoms of the ligand will be frequently accessed during the random move process and energy calculation. But limited by the size of register file per SM, only proton data can be accommodated in register file. At first, we also tried to use share memory to accommodate the atom data. But for some complicated ligand structure data, the share memory will run out. Moreover, we still need to save partial share memory to do the reduction, share memory is far less enough in this scenario. For grid data, it needs to be shared across all the thread blocks, although this part of data needs to be frequently accessed during energy calculation, it can only be accommodated in global memory.

The share memory usage in reduction part has been explained in the reduction section. Except for local memory usage, we also try to use vectorized memory access as much as possible. This method can scalar loads data, when data access is aligned and consecutive, it can largely relieve the memory bandwidth bound.

Chapter 5 MPI version and its parallel strategy

After getting the optimal one pair docking performance on GPU, we aim to reduce the virtual screen time, in the scenario of screening out the leading compound from multiple ligand candidates. The ultimate goal is to combine the GPU implementation with MPI to parallel different pairs at the same time of paralleling inside one pair, which can fully use the CPU and GPU resources of the cluster.

To implement a totally parallel for multi-pair, our parallel strategy is to use MPI sent input pairs to different nodes and dispatch the task to the GPU on a GPU node, or using the CPU resource process directly. Figure 5.1 shows the parallel strategy of the MPI version. A master processor will be created and will take charge of tasks dispatching to the slave processors. The master processor also takes responsibility to manage load balance. It keeps communicating with the slave processor, whenever there is a slave processor available, the master processor will dispatch tasks to it. The master processor will guarantee all the slave processors are working as far as it has enough waiting tasks. To guarantee the slave processor doesn't need to wait after its task finished, double-check is required between the master and the slave processor. Every slave processor needs to sent messages back to the master processor when its finished and request a new task. In this way, the slave processor can get a new task immediately without waiting. Also if there is a slave processor fail, the master processor will dispatch tasks to other slave processors. When every slave processor finishes its job, the Master processor will be notified and summaries the result data by using distributed Allreduce. Since the purpose of the virtual screen is to find out the leading compound, after all the tasks finished finding out the few top ligands with its minimum energy docking pose is the last step. MPI Allreduce method works well in this scenario.

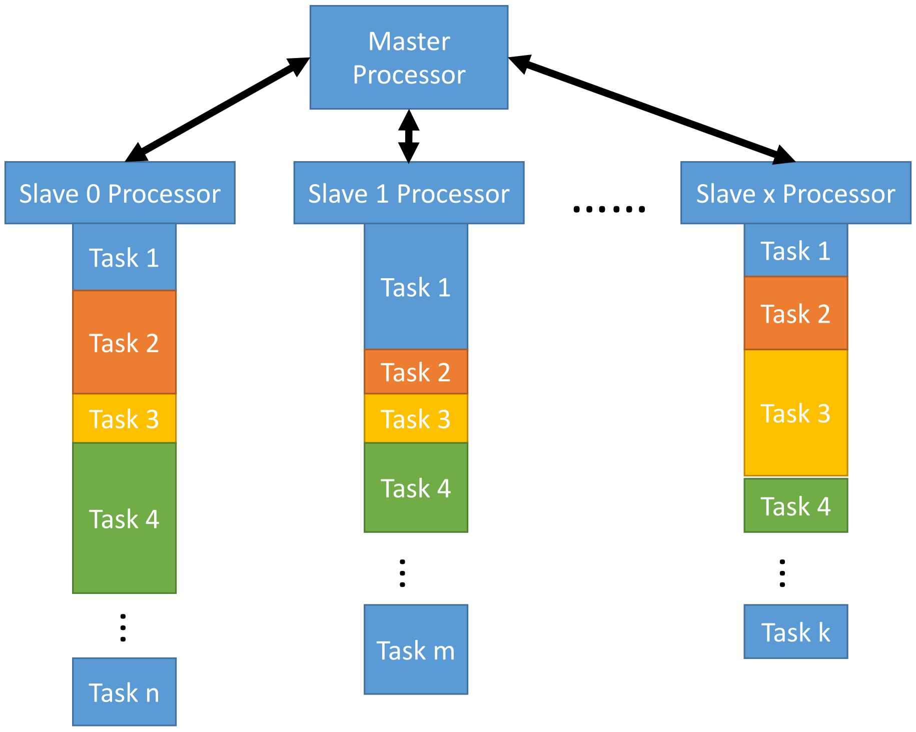


Figure 5.1. MPI parallel strategy

We use MPICH to parallel processing pairs. It compiled with intel ICC 16.0.3, running on Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 28 cores per node. When running on 9 nodes, 20 processors per node, total time consuming is 91 minutes, comparing with the original version the performance improve around 83.9x times. So it means currently using MPI version MedusaDock, we can finish the work in one or two hours for the PDBBind dataset which includes nearly 3900 protein-ligand pairs, this process used to take around a week. Moreover, if we increase the nodes and process numbers largely, the performance can improve correspondingly.

Chapter 6 Experimental Evaluation

6.1 Performance Result

Running environment configuration:
Using CUDA 8.0 to parallel processing inside pairs. Compiled with NVCC.
Platform configuration:
GPU: NVIDIA Tesla P100 GPU.
Host: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.4GHz (Broadwell), 28 cores per node.
Datasets: PDBbind (3875)

Figure 6.1 shows the performance speedup for the total execution time of every input case comparing with the original CPU version MedusaDock. For the whole datasets time consumption the GPU version improves around 1.8x times comparing with the original CPU version.

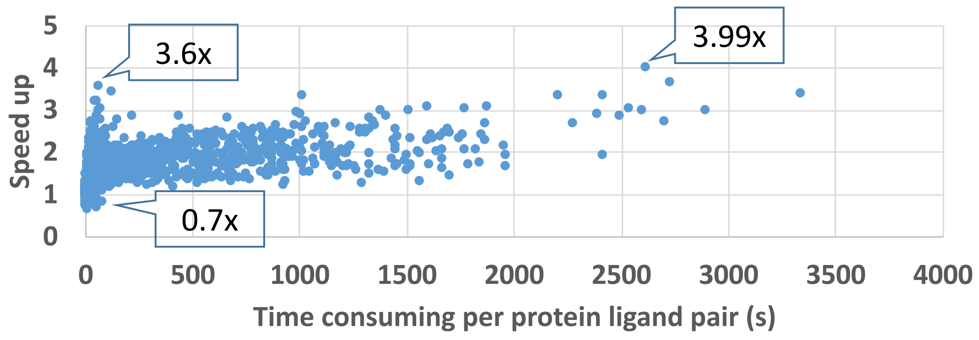


Figure 6.1. Performance improvement of the total docking time

From figure 6.1 we can conclude that the more time the protein-ligand pair consumes, the more likely it has higher speed up with GPU version. The Highest speedup as the figure shows is 3.99x. That is because the coarse docking time spends at most 5/6 of total docking time. So it means at most it can have 5x speedup at ideal situation. From the figure, we can read that different pairs with similar total docking times can have quite different speedup. The difference between these two cases comes from the different coarse docking time consumption percentage of the total docking time. While the preparation for coarse docking time is not a time-consuming step, the main difference comes from the ratio of coarse docking and fine docking. If we isolate the coarse docking part, their coarse docking time speed up is similar. Also, we still can find some cases slow down under the GPU version. For the slow down cases, the coarse docking part is not the bottleneck of the pair. At the profiling result from chapter 3 shows, the coarse docking step is not always the bottleneck, it varies case by case. Moreover, the GPU version has additional overhead, which includes data structure transfer, host to device memory copy, kernel launch, etc. Therefore, if the coarse docking step is not the bottleneck of the case, the performance drop is predictable. We use the Portable Batch System to get the CPU version performance. We also run the CPU version with partial datasets on GPU node, the result shows a little bit of performance drop. Hence, a less efficient CPU host could also be one of the performance drop reasons.

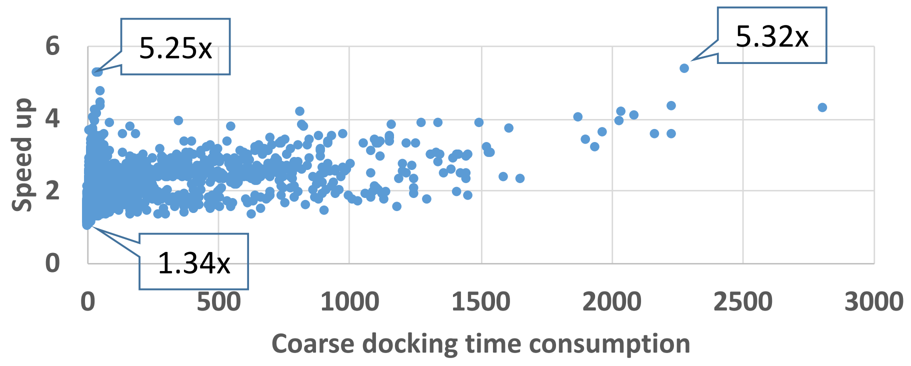


Figure 6.2. Performance improvement of the coarse docking time

Figure 6.2 shows the performance improvement of the coarse docking time. Coarse docking part is the main part, which accelerates by GPU architecture. Coarse docking part performance also shows positive correlation between execution time and speedup. The highest speedup is 5.32x comparing with CPU version. The reason why we can not get higher speedup is because of the repacking the side chain of the protein step. After we move the random move ligand conformation onto GPU architecture repack the side chain of the protein step instead coarse docking part becomes the bottleneck. The performance of this step highly depends on the complexity and flexibility of the protein. From figure 6.2, we can also find that similar coarse docking part time-consuming pairs can have quite different speeds up. It shows that although the coarse docking time is similar, the number of ligand rotamers is still quite different. Since our GPU version generates different poses for specific ligand rotamer in parallel, if the coarse docking time is similar, the less number of ligand rotamer, the more speedup that case can get.

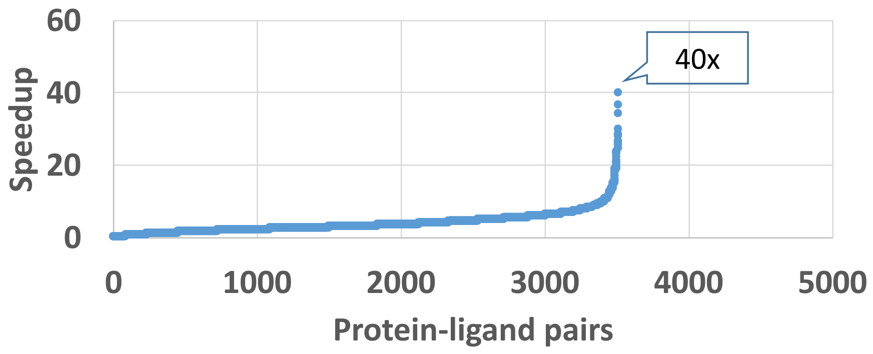


Figure 6.3. Performance improvement of the GPU kernel

If only comparing the GPU kernel runtime with the corresponding original CPU part, the performance improvement could be very huge. As the figure 6.3 shows, the GPU kernel can get at most 40x speedup. Around 26% of the protein-ligand pairs can get at least around 5x speedup. This data also verify the GPU architecture works efficiently on the docking algorithm acceleration.

6.2 Accuracy evaluation

We verify the correctness of the program in two parts, the first part is to evaluate the model energy, and the second part is to verify root-mean-square deviation (RMSD) value.

Our GPU version's calculation is based on complex energy. Therefore, as far as we can verify that for the PDBbind datasets our GPU version can get no lower energy binding poses than the CPU version, then we can verify the correctness. From the figure 6.4, we can see for nearly 3900 protein-ligand pairs datasets CPU version can find 287 cases, which has lower energy binding pose comparing with GPU version. While the GPU version can find 523 cases, which has lower energy binding pose comparing with the CPU version. All the rest 3065 cases have similar energy binding poses. It shows that if the energy difference is smaller than 10% of the MAX (CPU version, GPU version),

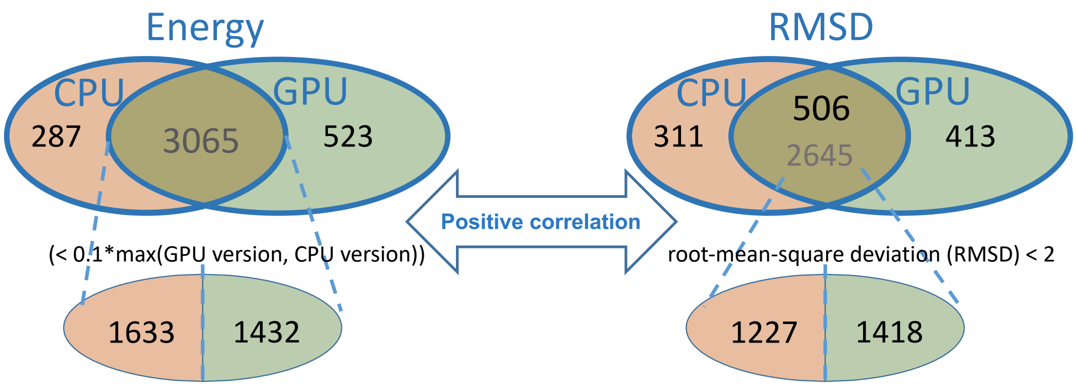


Figure 6.4. Venn diagram of Energy and RMSD value

their energy binding pose can be recognized as similar. Around 79% percentage of the cases are similar. To guarantee the fairness inside the similar result, we also compare all the similar result. There are 1633 cases CPU version performing better and 1432 cases GPU version performing better. Therefore, we conclude that our GPU version has similar accuracy compared to the CPU version.

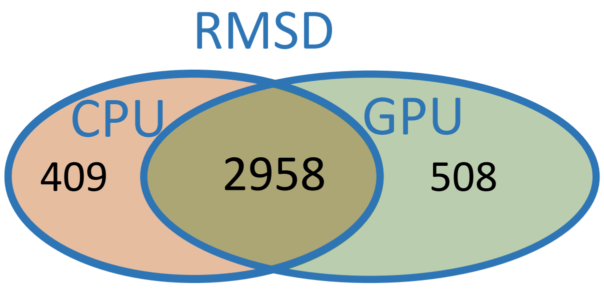


Figure 6.5. Venn diagram of RMSD value

Since the original MedusaDock uses RMSD value to verify the correctness. We also verify this parameter to guarantee the GPU version's correctness. The RMSD verification is separated into two parts: RMSD < 2, RMSD > 2. When RMSD < 2 the output pose can be treated as a successfully docked pose, otherwise the proposed pose is not fully docked. As figure 6.4 RMSD part shows, the CPU version could find 311 docked cases that GPU version can not find, while the GPU version can find 413 docked cases that CPU version can not find. There are 506 docked cases can be found by both CPU and GPU version. To guarantee the fairness of the RMSD > 2 cases, we also compare 2645 not fully dock cases, The CPU version has 1227 better cases and the GPU version has

1418 better cases. However, this comparison method does not take the similarity of the pose into consideration. In general, if RMSD value between two different poses is smaller than 2 , we can treat these two poses as similar pose. Therefore, we show another Venn diagram of RMSD value with similarity under consideration. The figure 6.5 shows that 2958 protein-ligand pairs can get similar binding pose, while the GPU version can find 508 significant better binding poses versus CPU version can only find 409.

In conclusion, the GPU version performs a little better than the CPU version. The reason why the CPU version could find lower energy/better conformation in some pairs which GPU version could not, is because of a perfect random seed for the CPU version. Since the current GPU kernel only accelerate the coarse docking step, and this step only tries to roughly find out the best pose candidate for fine docking step. For these failed cases fine docking step is the critical step to find out the better binding pose, its main step consists of enrichment and minimization, these steps closely associate with the random seed, a perfect random seed in this step can lead to a better binding pose. If we want to get a stable converged result we need to run MedusaDock hundreds of thousands of times with different random seed as input, in that situation most of the pairs (around 90%, based on 100 test cases) could find the docked conformation, which means a binding pose with its RMSD value smaller than 2 can be founded. Therefore, the accuracy of the GPU version is a little bit better than the CPU version, it means our GPU version can get converged more quickly, this also could be treated as performance improvement.

Chapter 7 Conclusion and Future Work

In this thesis, we propose a general computational molecular docking parallel strategy. We maintain fault-tolerant and loading balance inside the MPI version. And the GPU version carefully optimizes the main time-consuming parts targeting GPU architecture, including reduction part, local energy calculation, and memory access part. To evaluate the effectiveness of the proposed parallel strategy, we select MedusaDock as an example to show the performance and verify the correctness. The result demonstrates that our MPI version MedusaDock largely accelerates the virtual screen process, while the GPU version MedusaDock improves both the performance and accuracy of the MedusaDock.

With applying the parallel strategy on MedusaDock, our experiment result shows that both MPI and GPU version MedusaDock achieve superior performances than the original MedusaDock. The MPI version MedusaDock running on 9 nodes with 20 processes is 83.9x times faster than the original MedusaDock. If running on multiple nodes on a cluster the total time consuming can corresponding reduced depends on the node number. The GPU version achieves around 1.8x times improvement on overall performance.

For future work, on the one hand, the MedusaDock still has space to be optimized. After parallel processing, the coarse docking part on GPU, software bottleneck changes from coarse docking to fine docking. Although fine docking is highly nested with protein repacking side chain. It still can be moved onto GPU architecture. Repacking the protein side-chain is also the bottleneck in the protein complex case in coarse docking. The data structure convert and data transfer between CPU host and GPU device can also be optimized, by overlapping with some computational processes. We also need to combine the GPU version with the MPI version, so that we can fully use the resource of the heterogeneous computer Cluster. Currently, our GPU version only tested on one GPU device, if there are multiple GPU devices available, we can also parallel on ligand rotamers level across multiple GPU devices.

On the other hand, the traditional computational molecular docking method can combine with deep learning techniques. Currently, deep learning has dramatically promoted the development of various domains such as computer vision, natural language processing, and data mining. We also can consider leverage deep learning techniques on the identification of protein-ligand binding sites, virtual screening process and proposing docked pose. Before using deep learning techniques to directly solve the whole docking problem, we can consider combining the traditional molecular docking software with deep learning techniques by leveraging deep learning techniques on partial docking processes, like predicting the complex energy or picking out docked poses from all the conformation candidates. Currently, the score function which is used for calculating the complex energy works poorly on predicting the affinity of the complex, deep learning algorithm may solve this problem by training on large sums of known affinity information.

Bibliography

[1] WANG, J. and N. V. DOKHOLYAN (2019) "MedusaDock 2.0: Efficient and Accurate Protein–Ligand Docking With Constraints," *Journal of Chemical Information and Modeling*, **59**(6), pp. 2509–2515.

[2] BAJORATH, J. (2002) "Integration of virtual and high-throughput screening," *Nature Reviews Drug Discovery*, **1**(11), pp. 882–894.

[3] WALTERS, W. P., M. T. STAHL, and M. A. MURCKO (1998) "Virtual screening—an overview," *Drug discovery today*, **3**(4), pp. 160–178.

[4] SMITH, M. T. and J. L. RUBINSTEIN (2014) "Beyond blob-ology," *Science*, **345**(6197), pp. 617–619.

[5] BAI, X.-C., G. MCMULLAN, and S. H. SCHERES (2015) "How cryo-EM is revolutionizing structural biology," *Trends in biochemical sciences*, **40**(1), pp. 49–57.

[6] MENG, X.-Y., H.-X. ZHANG, M. MEZEI, and M. CUI (2011) "Molecular docking: a powerful approach for structure-based drug discovery," *Current computer-aided drug design*, **7**(2), pp. 146–157.

[7] JORGENSEN, W. L. (2004) "The many roles of computation in drug discovery," *Science*, **303**(5665), pp. 1813–1818.

[8] KITCHEN, D. B., H. DECORNEZ, J. R. FURR, and J. BAJORATH (2004) "Docking and scoring in virtual screening for drug discovery: methods and applications," *Nature reviews Drug discovery*, **3**(11), pp. 935–949.

[9] JONES, G., P. WILLETT, and R. C. GLEN (1995) "Molecular recognition of receptor sites using a genetic algorithm with a description of desolvation," *Journal of molecular biology*, **245**(1), pp. 43–53.

[10] TAYLOR, J. S. and R. M. BURNETT (2000) "DARWIN: a program for docking flexible molecules," *Proteins: Structure, Function, and Bioinformatics*, **41**(2), pp. 173–191.

[11] LIU, M. and S. WANG (1999) "MCDOCK: a Monte Carlo simulation approach to the molecular docking problem," *Journal of computer-aided molecular design*, **13**(5), pp. 435–451.

[12] Ding, F., S. Yin, and N. V. Dokholyan (2010) "Rapid flexible docking using a stochastic rotamer library of ligands," *Journal of chemical information and modeling*, **50**(9), pp. 1623–1632.

[13] Ding, F. and N. V. Dokholyan (2013) "Incorporating backbone flexibility in MedusaDock improves ligand-binding pose prediction in the CSAR2011 docking benchmark," *Journal of chemical information and modeling*, **53**(8), pp. 1871–1879.

[14] Trott, O. and A. J. Olson (2010) "AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *Journal of computational chemistry*, **31**(2), pp. 455–461.

[15] Rarey, M., B. Kramer, T. Lengauer, and G. Klebe (1996) "A fast flexible docking method using an incremental construction algorithm," *Journal of molecular biology*, **261**(3), pp. 470–489.

[16] Ewing, T. J. and I. D. Kuntz (1997) "Critical evaluation of search algorithms for automated molecular docking and database screening," *Journal of computational chemistry*, **18**(9), pp. 1175–1189.

[17] Trosset, J.-Y. and H. A. Scheraga (1999) "PRODOCK: software package for protein modeling and docking," *Journal of computational chemistry*, **20**(4), pp. 412–427.

[18] Brooks, B. R., R. E. Bruccoleri, B. D. Olafson, D. J. States, S. a. Swaminathan, and M. Karplus (1983) "CHARMM: a program for macromolecular energy, minimization, and dynamics calculations," *Journal of computational chemistry*, **4**(2), pp. 187–217.

[19] Cornell, W. D., P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman (1996) "A second generation force field for the simulation of proteins, nucleic acids, and organic molecules J. Am. Chem. Soc. 1995, 117, 5179- 5197," *Journal of the American Chemical Society*, **118**(9), pp. 2309–2309.

[20] Eldridge, M. D., C. W. Murray, T. R. Auton, G. V. Paolini, and R. P. Mee (1997) "Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes," *Journal of computer-aided molecular design*, **11**(5), pp. 425–445.

[21] Friesner, R. A., J. L. Banks, R. B. Murphy, T. A. Halgren, J. J. Klicic, D. T. Mainz, M. P. Repasky, E. H. Knoll, M. Shelley, J. K. Perry, et al. (2004) "Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy," *Journal of medicinal chemistry*, **47**(7), pp. 1739–1749.

[22] Morris, G. M., D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson (1998) "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function," *Journal of computational chemistry*, **19**(14), pp. 1639–1662.

[23] Muegge, I. and Y. C. Martin (1999) "A general and fast scoring function for protein- ligand interactions: a simplified potential approach," *Journal of medicinal chemistry*, **42**(5), pp. 791–804.

[24] Mitchell, J. B., R. A. Laskowski, A. Alex, and J. M. Thornton (1999) "BLEEP—potential of mean force describing protein–ligand interactions: I. Generating potential," *Journal of Computational Chemistry*, **20**(11), pp. 1165–1176.

[25] Velec, H. F., H. Gohlke, and G. Klebe (2005) "DrugScoreCSD knowledge-based scoring function derived from small molecule crystal data with superior recognition rate of near-native ligand poses and better affinity prediction," *Journal of medicinal chemistry*, **48**(20), pp. 6296–6303.

[26] Norgan, A. P., P. K. Coffman, J.-P. A. Kocher, D. J. Katzmann, and C. P. Sosa (2011) "Multilevel parallelization of AutoDock 4.2," *Journal of cheminformatics*, **3**(1), p. 12.

[27] Micevski, D. (2009) "Optimizing Autodock with CUDA," *Victorian Partnership For Advanced Computing Ltd*.

[28] Nowotny, T. (2010) "Parallel implementation of a spiking neuronal network model of unsupervised olfactory learning on NVidia® CUDA™," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1–8.

[29] Altuntaş, S., Z. Bozkus, and B. B. Fraguela (2016) "GPU accelerated molecular docking simulation with genetic algorithms," in *European Conference on the Applications of Evolutionary Computation*, Springer, pp. 134–146.

[30] Prakhov, N. D., A. L. Chernorudskiy, and M. R. Gainullin (2010) "VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters," *Bioinformatics*, **26**(10), pp. 1374–1375.

[31] Landaverde, R. and M. C. Herbordt (2014) "GPU optimizations for a production molecular docking code," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, pp. 1–6.

[32] Sukhwani, B. and M. C. Herbordt (2008) "Acceleration of a production rigid molecule docking code," in *2008 International Conference on Field Programmable Logic and Applications*, IEEE, pp. 341–346.

[33] ——— (2010) "FPGA acceleration of rigid-molecule docking codes," *IET computers & digital techniques*, **4**(3), pp. 184–195.

[34] Allusse, Y., P. Horain, A. Agarwal, and C. Saipriyadarshan (2008) “Gpucv: an opensource gpu-accelerated framework forimage processing and computer vision,” in *Proceedings of the 16th ACM international conference on Multimedia*, pp. 1089–1092.

[35] Heymann, S., K. Müller, A. Smolic, B. Froehlich, and T. Wiegand (2007) “SIFT implementation and optimization for general-purpose GPU,” .

[36] Fan, M., H. Jia, Y. Zhang, X. An, and T. Cao (2015) “Optimizing Image Sharpening Algorithm on GPU,” in *2015 44th International Conference on Parallel Processing*, IEEE, pp. 230–239.

[37] Chetlur, S., C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer (2014) “cudnn: Efficient primitives for deep learning,” *arXiv preprint arXiv:1410.0759*.

[38] Sukhwani, B. and M. C. Herbordt (2009) “GPU acceleration of a production molecular docking code,” in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pp. 19–27.

[39] Karimi, K., N. G. Dickson, and F. Hamze (2010) “A performance comparison of CUDA and OpenCL,” *arXiv preprint arXiv:1005.2581*.

[40] Stone, J. E., D. Gohara, and G. Shi (2010) “OpenCL: A parallel programming standard for heterogeneous computing systems,” *Computing in science & engineering*, **12**(3), pp. 66–73.

[41] Kuhlman, B., G. Dantas, G. C. Ireton, G. Varani, B. L. Stoddard, and D. Baker (2003) “Design of a novel globular protein fold with atomic-level accuracy,” *science*, **302**(5649), pp. 1364–1368.

[42] Ren, N., J. Liang, X. Qu, J. Li, B. Lu, and J. Tian (2010) “GPU-based Monte Carlo simulation for light propagation in complex heterogeneous tissues,” *Optics express*, **18**(7), pp. 6811–6823.

[43] Anderson, J. A., E. Jankowski, T. L. Grubb, M. Engel, and S. C. Glotzer (2013) “Massively parallel Monte Carlo for many-particle simulations on GPUs,” *Journal of Computational Physics*, **254**, pp. 27–38.

[44] Preis, T., P. Virnau, W. Paul, and J. J. Schneider (2009) “GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model,” *Journal of Computational Physics*, **228**(12), pp. 4468–4477.