Hindawi
Scientific Programming
Volume 2022, Article ID 9197606, 13 pages
https://doi.org/10.1155/2022/9197606

Research Article

Molecular Docking for Ligand-Receptor Binding Process Based on Heterogeneous Computing

Jianhua Li,[1] **Guanlong Liu**,[1] **Zhiyuan Zhen**,[1] **Zihao Shen**,[2] **Shiliang Li**,[2] **and Honglin Li**[2]

[1]*School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China*
[2]*State Key Laboratory of Bioreactor Engineering, Shanghai Key Laboratory of New Drug Design, School of Pharmacy, East China University of Science and Technology, Shanghai 200237, China*

Correspondence should be addressed to Jianhua Li; jhli@ecust.edu.cn

Received 24 October 2021; Revised 12 December 2021; Accepted 28 December 2021; Published 10 January 2022

Academic Editor: Basilio B. Fraguela

Molecular docking aims to predict possible drug candidates for many diseases, and it is computationally intensive. Particularly, in simulating the ligand-receptor binding process, the binding pocket of the receptor is divided into subcubes, and when the ligand is docked into all cubes, there are many molecular docking tasks, which are extremely time-consuming. In this study, we propose a heterogeneous parallel scheme of molecular docking for the binding process of ligand to receptor to accelerate simulating. The parallel scheme includes two layers of parallelism, a coarse-grained layer of parallelism implemented in the message-passing interface (MPI) and a fine-grained layer of parallelism focused on the graphics processing unit (GPU). At the coarse-grain layer of parallelism, a docking task inside one lattice is assigned to one unique MPI process, and a grouped master-slave mode is used to allocate and schedule the tasks. Meanwhile, at the fine-gained layer of parallelism, GPU accelerators undertake the computationally intensive computing of scoring functions and related conformation spatial transformations in a single docking task. The results of the experiments for the ligand-receptor binding process show that on a multicore server with GPUs the parallel program has achieved a speedup ratio as high as 45 times in flexible docking and as high as 54.5 times in semiflexible docking, and on a distributed memory system, the docking time for flexible docking and that for semiflexible docking gradually decrease as the number of nodes used in the parallel program gradually increases. The scalability of the parallel program is also verified in multiple nodes on a distributed memory system and is approximately linear.

1. Introduction

Ligand-receptor molecular docking is a significant computational method to study intermolecular interactions and predict the structure of complex based on the principle of shape complementarity and property complementarity between ligand and receptor. Molecular docking is mandatory for the discovery of new drugs [1] because a lot of emerging diseases demand prompt treatments, such as COVID-19 [2].

Molecular docking always requires massive computing consumption. In the docking process, computing costs come from several deciding factors, such as the flexibility of receptor [3] and the number of docking instances. According to the flexibility of the receptor, molecular docking can be classified into two groups, namely, rigid docking and flexible docking. Rigid molecular docking methods simply regard the receptor structure as a rigid configuration without the flexibility of the receptor and are not very accurate in predicting ligand-receptor binding modes. In contrast, flexible docking methods allow the dynamic conformational changes of the receptor during the binding process [4], which makes the docking simulation process closer to the real process. Flexible docking is available in many docking programs, such as AutoDock [5], DOCK [6], AutoDock Vina [7], GalaxyDock [8], and FITTED [9]. This kind of method is more accurate than the rigid method. With the increasing complexity of the receptor, the time of docking simulation will be significantly increased [10]. Meanwhile, different docking applications need variable docking instances. For example, in docking-based virtual screening [11], each ligand is docked with a particular region (active site) of a specific receptor with elucidated 3D structure, and

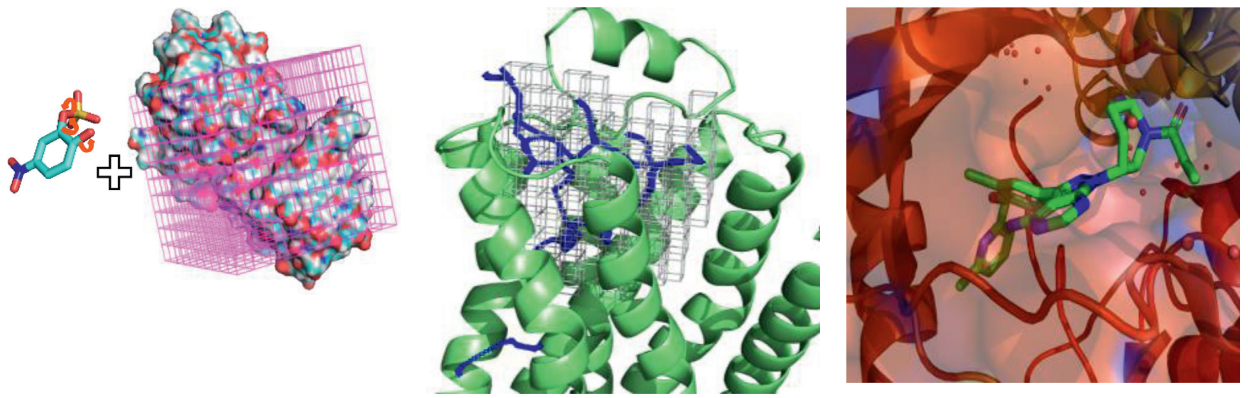
with the increase in docking instances, the time expenditure of the virtual screening is more than that of an active-site docking. However, the flexibility of the receptor and the increasing docking instances lead to a more computationally intensive and inefficient docking simulation, limiting the role of molecular docking in drug discovery.

To solve the time-consuming challenges of molecular docking, parallel computing is used to speed up molecular docking methods. The first way is to use a message-passing interface (MPI) to parallelize docking methods for clusters or multicore servers [12–14], e.g., VinaLC [15] and Dock6.MPI [16]. VinaLC is an AutoDock Vina parallel docking program developed with MPI and multithreaded technology and has been scaled up to more than 15 K CPUs. Using VinaLC, docking calculations of one million flexible compounds took only 1.4 hours to finish. Dock6.MPI, the parallel version of Dock6, is based on the traditional master-slave mode of MPI. In this mode, the master node performs I/O operations and docking task management, while the slave nodes simultaneously perform independent docking calculations. Dock6.MPI has been reported to run 27,000 docking tasks on Blue Gene using 4,096 and 8,192 processors with 8% and 12% overhead, respectively. Although MPI programming is suitable for the parallel execution of docking tasks in large clusters, it requires expensive computing resources. Utilizing graphics processing unit (GPU) processors is the second way to accelerate docking applications [17–22]. GeauxDock [23] is a docking program specially developed for accelerators. It achieves a speedup as high as x 3.5 compared to a serial program [24]. Altuntaș et al. offloaded the most computationally intensive part of any docking simulation, which is the genetic algorithm, to accelerators [25]. The GPU-accelerated system achieves a speedup of around ~ x 14 with respect to a single CPU core. Mengran Fan et al. proposed a generalized parallel strategy using GPU acceleration for the molecular docking program MedusaDock [26]. The overall performance of the GPU version has increased by about 4 times [27]. Currently, most GPU-based docking programs use a GPU processor so that their speedup is usually within 10, as compared to the serial program. The third way is to use a batch system with parallel job scheduling. DOVIS2.0 [28] is a docking program based on AutoDock4, which starts docking tasks in parallel through a batch queue. In a cluster system with multiple CPUs, DOVIS 2.0 has achieved a higher speedup. However, this method uses intermediate files to transfer data to ensure communication between nodes, which substantially increases the I/O consumption in the docking process, slows down the calculation, and limits the maximum number of CPUs that can be used simultaneously.

Undeniably, only very few scientists use the way of heterogeneous computing (i.e., MPI and GPU) to accelerate docking computation, because computing characteristics in most docking tasks do not fit heterogeneous computing. Usually, scientists directly use the way of MPI parallelism or GPU parallelism. However, MPI and GPU can be combined and used for large-scale computing tasks in many fields, and the MPI-GPU heterogeneous way has been widely used [29–31]. In computational fluid dynamics, Choi et al. used a floating-point compression algorithm to optimize the GPU memory capacity in the heterogeneous MPI-GPU implementation [32], and Lai et al. developed a heterogeneous parallel program combining MPI and CUDA for CFD applications on high-performance computing clusters to greatly improve computational efficiency [33]. Komatitsch et al. used MPI to model high-order finite-element seismic wave propagation on the large GPU cluster in physics and seismology [34]. In machine learning, the heterogeneous parallel sequence minimization optimization algorithm uses GPU to implement binary classifiers and MPI to solve multiclass on the "one-against-one" method [35]. In summary, the heterogeneous parallelism of MPI-GPU has been applied in many fields, and it is a potential way to accelerate molecular docking simulation.

In this study, we propose a heterogeneous parallel scheme of molecular docking for the ligand-receptor binding process. The ligand-receptor binding process, developed by Bai et al. [36], aims at a rigorous and precise binding free energy landscape. To simulate the ligand-receptor binding process, the binding pocket of the receptor is divided into subcubes (lattices), the ligand is docked into each lattice, and the binding configurations are obtained inside the lattice. The whole docking-based simulated ligand-receptor binding process is demonstrated in Figure 1. There are a large number of molecular docking tasks in the binding process, which is extremely time-consuming. Fortunately, this kind of computing can fit heterogeneous parallel computing. Based on MPI-GPU, the proposed heterogeneous parallel scheme includes two layers of parallelism, namely, a coarse-grain layer and a fine-grain layer of parallelism. At the coarse-grain layer of parallelism, a docking task for one lattice is assigned to one unique MPI process, and a grouped master-slave mode is used to allocate and schedule these tasks. Meanwhile, at the fine-gained layer of parallelism, GPU accelerators undertake the computationally intensive computing of scoring functions and related conformation spatial transformations in one single docking task, and the CPU takes the rest. Briefly, our main contribution to this study is to provide a heterogeneous parallel scheme, which combines both MPI and GPU to accelerate molecular docking for the ligand-receptor binding process. This is the first attempt for parallelization of molecular docking for ligand-receptor binding process based on heterogeneous computing, which includes both CPU-end parallelism and GPU-end parallelism, and the experimental results of the parallel scheme prove its feasibility from the docking acceleration and scalability.

The remainder of this study is organized as follows: Section 2 briefly describes the molecular docking for the ligand-receptor binding process, proposes the parallel scheme for the ligand-receptor binding process, and introduces the implementation details of the parallel scheme. Section 3 describes the experimental results obtained by using the parallel programs that are introduced. Finally, conclusions and future works are given in Section 4.



(a) (b) (c)

Figure 1: Docking-based ligand-receptor binding process. (a) A lattice model is constructed for the ***β*2AR** by dividing the large cubic box surrounding the protein into many small cubic boxes (lattices). The color of small lattices is pink. (b) Blue stick represents the possible ligand-binding pathway addressed by using the Voronoi mesh-based MOLE [37] algorithm. We pick out lattices that enclose the possible binding pockets and the ligands entering and leaving paths. Each lattice (gray lattice) represents a docking task. (c) The stick represents the ligand. Ligand is in the binding pocket of protein.

2. Materials and Methods

2.1. Molecular Docking for Ligand-Receptor Binding Process. The objective of molecular docking for the ligand-receptor binding process is to construct a rigorous and precise binding free energy landscape and further to estimate both the binding affinity and the binding kinetics. In molecular docking for ligand-receptor binding process, we divide the binding pocket of the receptor into subcubes (lattices), pick out the lattices that enclose the possible binding pockets and the ligands entering and leaving paths, and fix the ligand into each selected lattice using a unique docking program iFitDock.

iFitDock is used for binding configuration sampling in each selected lattice. To achieve a complete flexible docking, iFitDock provides two docking modules, namely, semiflexible docking and flexible docking, and one real docking consists of two docking steps, namely, a flexible docking and a following semiflexible docking. In the flexible docking of iFitDock, the flexibility of the receptor is considered, and it allows the conformational free change of both ligand and receptor. In the semiflexible docking, the flexibility of the ligand is considered and the receptor is regarded as a rigid structure. Here, iFitDock introduces the multiobjective genetic algorithm NSGA-II [38] for conformation searching, and it uses molecular mechanics generalized Born surface area (MM-GBSA) [39, 40] binding free energy for conformation evaluation. The energy function in iFitDock, as shown in equation (1), is a weighted sum of terms representing van der Waals, electrostatic, and desolvation contributions.

$$\Delta E = w_1 \Delta E_{v\,dw} + w_2 \Delta E_{es} + w_3 \Delta E_{gb} + w_4 \Delta E_{sa}. \quad (1)$$

Due to different flexibilities in the semiflexible docking module and flexible docking module, conformation evaluation in each module is different.

In a flexible docking module, the conformation evaluation contains conformation spatial transformations and three objective functions. In particular, conformation spatial transformations include ligand's entire rotation and translation, rotation of ligand's rotatable bond, and rotation of residue's rotatable bond. The three objective functions are shown as follows:

$$\begin{aligned} f_1(x) &= \Delta G_{L\_\text{intra}} + \Delta G_{fR\_\text{intra}} \\ &= \Delta E_{v\,dv\,\_L} + \Delta E_{es\_L} + \Delta E_{v\,dv\,\_fR} + \Delta E_{es\_fR}, \end{aligned} \quad (2)$$

$$\begin{aligned} f_2(x) &= \Delta G_{\text{inter}\_L-\text{fR}} + \Delta G_{\text{inter}\_L-\text{rR}} \\ &= \Delta E_{v\,dv\,\_L-fR} + \Delta E_{es\_L-fR} \\ &\quad + \Delta E_{v\,dv\,\_L-rR} + \Delta E_{es\_L-rR}, \end{aligned} \quad (3)$$

$$\begin{aligned} f_3(x) &= \Delta G_{\text{inter}\_rR-fR} \\ &= \Delta E_{v\,dv\,\_rR-fR} + \Delta E_{es\_rR-fR}. \end{aligned} \quad (4)$$

In equations (3) and (4), "L" represents the small molecule, "fR" and "rR" represent the flexible residues of the protein and the rigid parts of the protein, respectively. $f_1(x)$ represents the sum of the internal energies of small molecules and flexible residues. $f_2(x)$ represents the interaction energy between small molecule and protein (the flexible residues and the rigid part). $f_3(x)$ represents the interaction energy between the flexible residues and the rigid part.

In the semiflexible docking, the conformation evaluation contains similar conformation spatial transformations and two objective functions. Here, conformation spatial transformations only include the entire rotation and translation of the ligand and the rotation of rotatable bonds of the ligand. The two objective functions are shown as follows:

$$\begin{aligned} f_1(x) &= \Delta G_{L\_\text{intra}} \\ &= \Delta E_{v\,dv\,\_L} + \Delta E_{es\_L}, \end{aligned} \quad (5)$$

$$\begin{aligned} f_2(x) &= \Delta G_{\text{inter}\_L-R} \\ &= \Delta E_{v\,dv\,\_L-R} + \Delta E_{es\_L-R}. \end{aligned} \quad (6)$$

In equations (5) and (6), "L" and "R" represent the small molecule and the protein, respectively. $f_1(x)$ represents the internal energy of the small molecule, and $f_2(x)$ represents the energy of interaction between the small molecule and the protein.

The detailed flowchart of iFitDock is illustrated in Figure 2.

Taking the iFitDock into consideration, the detailed steps of molecular docking for the ligand-receptor binding process are described as follows:

(1) The receptor structure is analyzed, and the possible binding pockets or gorges for the ligand entering or leaving from the active site are addressed.

(2) A lattice model for the receptor protein is constructed by dividing the large cubic box surrounding the protein into many small cubic boxes (lattices), and N lattices are picked out, which enclose the possible binding pockets and the ligands entering and leaving paths.

(3) n candidate conformations of the protein are generated by performing the flexible docking module of iFitDock for all selected lattices.

(4) A corresponding lattice model is constructed for each candidate conformation by dividing the large cubic box surrounding the candidate conformation into many lattices, and M lattices are picked out that enclose the possible binding pockets and the ligands entering and leaving paths.

(5) $n * M$ ligand-receptor binding configurations are generated by performing the semiflexible docking module of iFitDock for the selected lattices obtained in step (4).

In step (3) and step (5) of molecular docking for the ligand-receptor binding process, the flexible docking module and the semiflexible docking module are called multiple times in different lattices. Although each of the docking instances is relatively independent, the entire molecular docking is extremely time-consuming.

2.2. Parallelism Scheme. To accelerate molecular docking for the ligand-receptor binding process, parallelization of multiple time-consuming docking instances is a regular and efficient way. Considering that there are many involved docking instances and the intensive computing consumption of one docking instance is expensive, a heterogeneous MPI-GPU parallel scheme is used to accelerate the simulation. On one hand, MPI can support both shared memory and distributed memory systems with better scalability, so we choose MPI to perform multiple docking instances. On the other hand, as for one docking instance, we try to map the most computationally intensive part of the serial program to the GPU accelerators. In one word, our MPI-GPU heterogeneous parallelism scheme is constructed in two layers of parallelism:

(a) At the coarse-grain layer of parallelism, a docking computation is taken, which is in one lattice as a parallel allocation unit, and each allocation unit is assigned to one process using a grouped master-slave scheduling strategy;

(b) At the fine-grain layer of parallelism, the computationally intensive parts are offloaded to GPU and the rest of the program is kept on CPU.

In the coarse-grain layer of parallelism, MPI computation is chosen as the implementation technology because it can be widely used in clusters and also can be used in a shared memory computer or even a multicore computer.

Master-slave parallel strategy is used for task assignment and data transfer in coarse-grain parallelization. In the traditional MPI master-slave mode, the master process performs control and the slave processes perform computation. The task of the master process is to initialize all processes and distribute the calculation tasks to the slave processes; the slave processes finish the calculation tasks according to the requirements of the master process and return the final obtained results to the master process. However, when there are a large number of slave processes, the master process suffers from the heavy communication cost among itself and the slaves, and it becomes a bottleneck. Hence, a grouped master-slave mode of MPI is taken and used to mainly alleviate the heavy communication overhead. First, all processes are divided into multiple groups by grouped parameters, and then, the simple master-slave mode is adopted within one group. To balance workloads for different groups, a uniform block partition policy is taken to allocate processes to groups. This kind of parallelization can reduce communication time overhead and greatly improve parallel efficiency.

At the fine-grain layer of parallelism, the computationally intensive part(s) of the docking task in one lattice is offloaded to GPU. To identify which parts are computationally intensive in a docking task, we first analyze the time expenditures for serial iFitDock. Here, two different experiments of iFitDock are investigated to get the actual time statistics. In a semiflexible docking experiment, the docking of the protein carboxypeptidase A (1CPS) is taken as an example, while in the flexible docking experiment, the docking of the estrogen receptor α (3DT3) is tested. Considering that the NSGA-II algorithm is a heuristic algorithm, the time consumptions have slight deviations among repeated experiments, and all the recorded values are the average of repeated 10 experiments. In this study, the parameters of NSGA-II for all apartments are set as follows: the population size is set to 3,000, the value of crossover probability is set to 0.9, the value of mutation probability is set to 0.3, and the evolution generation is set to 600.

According to the functions of the program, both the semiflexible docking module and flexible docking module of the iFitDock program can be divided into five submodules, and their time expenditures in semiflexible docking and those in flexible docking are, respectively, shown in Tables 1 and 2. As can be seen from Tables 1 and 2, both the most time-consuming parts are the objective functions in both docking modules and their time ratio reaches up to 75.58% and 90.6%, respectively, due to the complex computing process of free binding energy. Moreover, the time

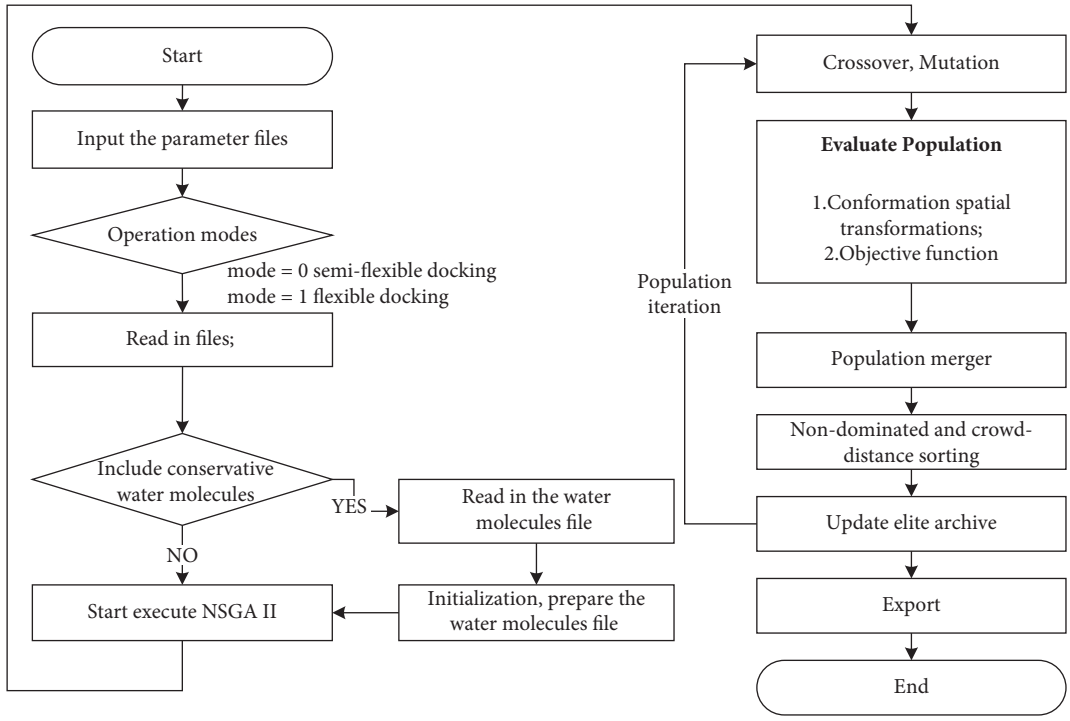


Figure 2: Docking flowchart of iFitDock.

Table 1: Time elapsed for different modules of the semiflexible experiment in iFitDock.

Submodule	Time (seconds)	Proportion of total time (%)
IO	4.2	0.16
Crossover, mutation, selection, and nondominated sorting	298	11.26
Rotation and translation of ligand	345	13
Calculation of objective functions	1999	75.58
Total time	2646.2	100

This table lists the elapsed time, and the most time-consuming part is the calculation of objective functions.

Table 2: Time elapsed for different modules of the flexible experiment in iFitDock.

Module	Time (seconds)	Proportion of total time (%)
IO	1462	3.5
Crossover, mutation, selection, and nondominated sorting	276	0.7
Rotation and translation of ligand, and rotation of residues	2169	5.2
Calculation of objective functions	37323	90.6
Total time	41230	100

This table lists the elapsed time, and the most time-consuming part is the calculation of objective functions.

expenditures of the rotation and translation in both dockings, which belong to the conformation searching, take the second palace in time expenditures.

According to the above statistics, it is promising to offload the computation of scoring functions and related conformation spatial transformation to GPU. Since objective functions and conformational transformation are computed in each individual of the population within NSGA-II and are not related to other individuals, it is practical to make them separately calculated in the GPU threads.

The final MPI-GPU parallel scheme is shown in Figure 3.

2.3. Computational Details. In this section, the specific computational details of the above parallel scheme are described. A carefully designed MPI + GPU heterogeneous parallel program has been developed based on the above parallel scheme.

In the coarse-grained layer of parallelization, there are two main issues in the implementation: (1) processes grouped and task assignment and (2) communication among the processes.

The grouped master-slave mode of MPI is applied to this work. Supposing the number of processes is n and the grouped parameter is g, all MPI processes should be divided

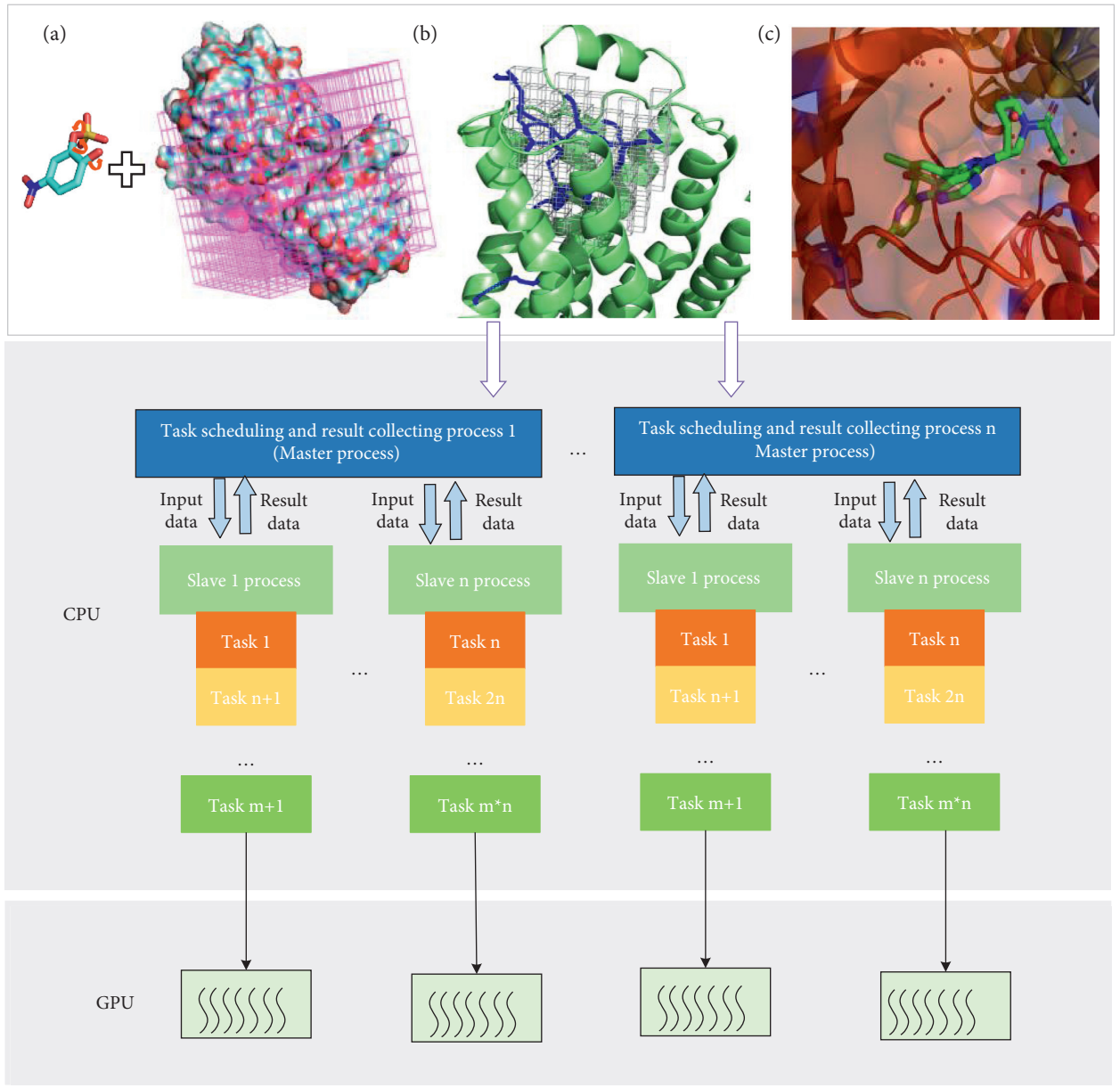


Figure 3: MPI-GPU parallel scheme. Here, the uppermost part is from Figure 1.

into g groups. The number of processes p g in a group is computed as follows:

$$pg = \lceil \frac{n}{g} \rceil. \tag{7}$$

In this policy, the master process is ranked from 0 to $g-1$, and the slave processes are ranked from g to $n-1$.

However, all docking tasks are assigned to a specific process group. A block partition policy is taken to allocate tasks to a process group. If the amount of all tasks is m, the number of tasks t g in a group is computed as follows:

$$tg = \lceil \frac{m}{g} \rceil. \tag{8}$$

In each group, the master takes charge of task assignments, and there is a lot of communication among the processes in the group. There are two statuses, namely, active phase and end phase, in a single group as illustrated in Figure 4. When all processes are started, the group steps into the active phases. In the active phase, free slave processes send the values of their ranks to the master process, and the master receives the messages from any free slave process by setting the data source parameter to MPI_ANY_SOURCE. When the master has received the messages and there are unassigned docking tasks, the master assigns an unassigned docking task to a free slave process by sending a task status "true" to the slave process. Later, the master process reads the file information required for docking in the form of a character stream and sends this file information to the slave process in the form of another MPI message. When the slave process has received the data transferred by the master process, it performs the docking task. After the docking is

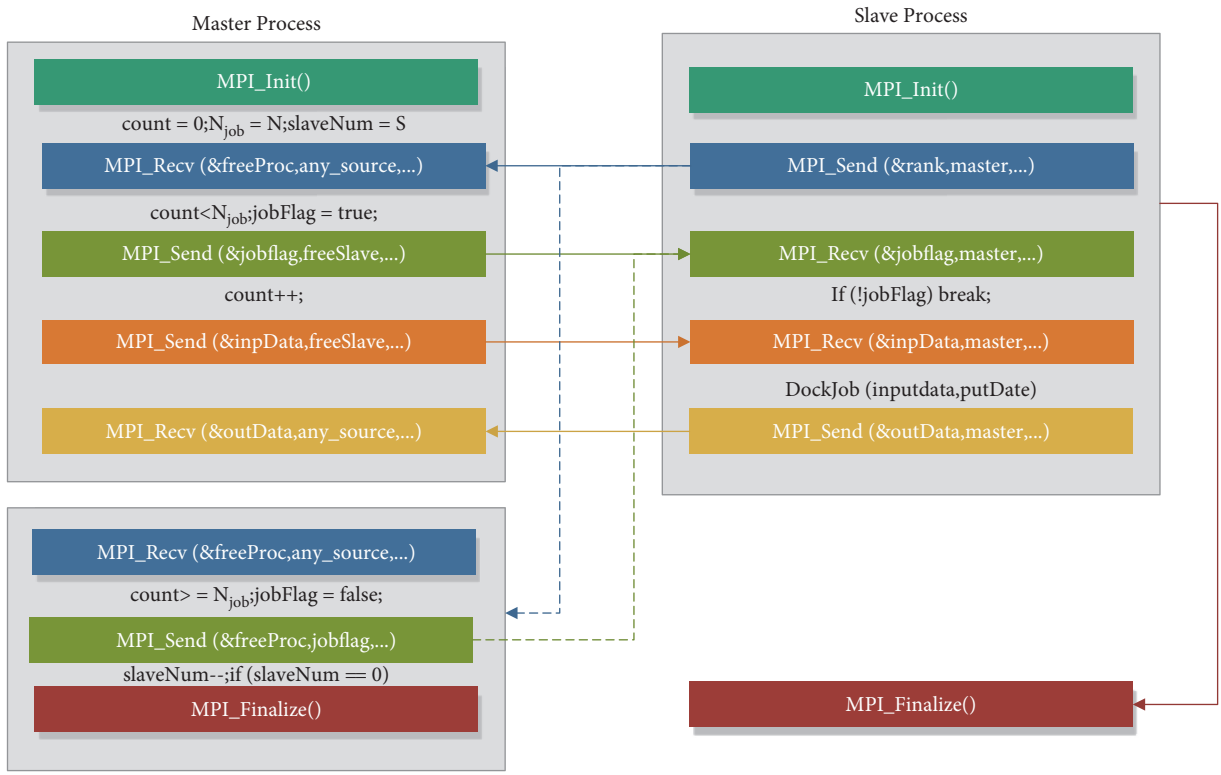


FIGURE 4: Implementation of master-slave mode in a single group. MPI calls are paired by boxes of the same color and connected by arrows. *count* is the number of docking tasks that have been completed. N_{job} is the number of docking tasks. *jobFlag* represents the task status, that is, whether there are undistributed tasks. *slaveNum* is the number of slave processes in a group. *slaveNum* equals p_g in equation (6).

completed, the slave process sends its output data and free state to the master process so that the master process can continue to assign tasks. If there are still unassigned tasks, the master process will continue to assign tasks to the free slave processes; otherwise, the master process will notify all free slave processes to enter the end phase by sending a task status "false," to gradually reduce the number of active processes. When the number of active slave processes becomes zero, the master process also enters the end phase. In the end phase, the master process or slave process will call MPI_Finalize() to release resources and end the MPI environment. In implementing such a master-slave MPI scheme in a single group, the master process is in charge of job dispatching and transferring of input files, while the slave processes are kept busy performing individual docking tasks until all the tasks are finished and return output files. The communications including file transferring and notification messages are all accomplished by a pair of simple MPI send/receive calls.

At the fine-grain layer of parallelism, the GPU parallel scheme is based on CUDA programming to perform docking calculations in each slave process. The computationally intensive part(s) of the docking task in one lattice are offloaded to GPU. In molecular docking for ligand-receptor binding process, although the semiflexible docking module and the flexible docking module deal with the receptor in different ways, they have similar computational processes.

Here, we take the CPU-GPU heterogeneous computing for flexible docking as an example to illustrate the computational process. Figure 5 shows CPU-GPU parallel implementation of one iteration process in flexible docking for one lattice. The CPU-GPU program consists of three parts, namely, the program in CPU, the program in GPU, and the exchanging of data, between CPU and GPU. The CPU program not only deals with input data and output data but also handles several steps of the NSGA-II including crossover, mutation, merging, sorting, and updating the elite archive file. In addition, the CPU program also handles the accumulation of objective functions. The exchange data between CPU and GPU mainly include the spatial coordinate values of all atoms, energy lattice values, the van der Waals energy, and electrostatic potential energy between atoms. The program in GPU includes two parts: conformation spatial transformations and three objective functions.

In an iteration of NSGA-II in flexible docking, the program in the CPU first performs the crossover and mutation of the genetic algorithm and transfers the data of the spatial coordinate values of all atoms and grid energy values to the GPU device through the PCIe bus.

After the data have been copied into GPU, the program in GPU computes the three-dimensional transformation of the entire molecule and three-dimensional transformation around rotatable bonds of atoms. The kernel function of these two rotation transformations defines a grid that contains (3 $*$ atom number $*$ population size) threads. Thus, each thread can compute 3 coordinate values of an atom in each individual. Subsequently, the coordinate values

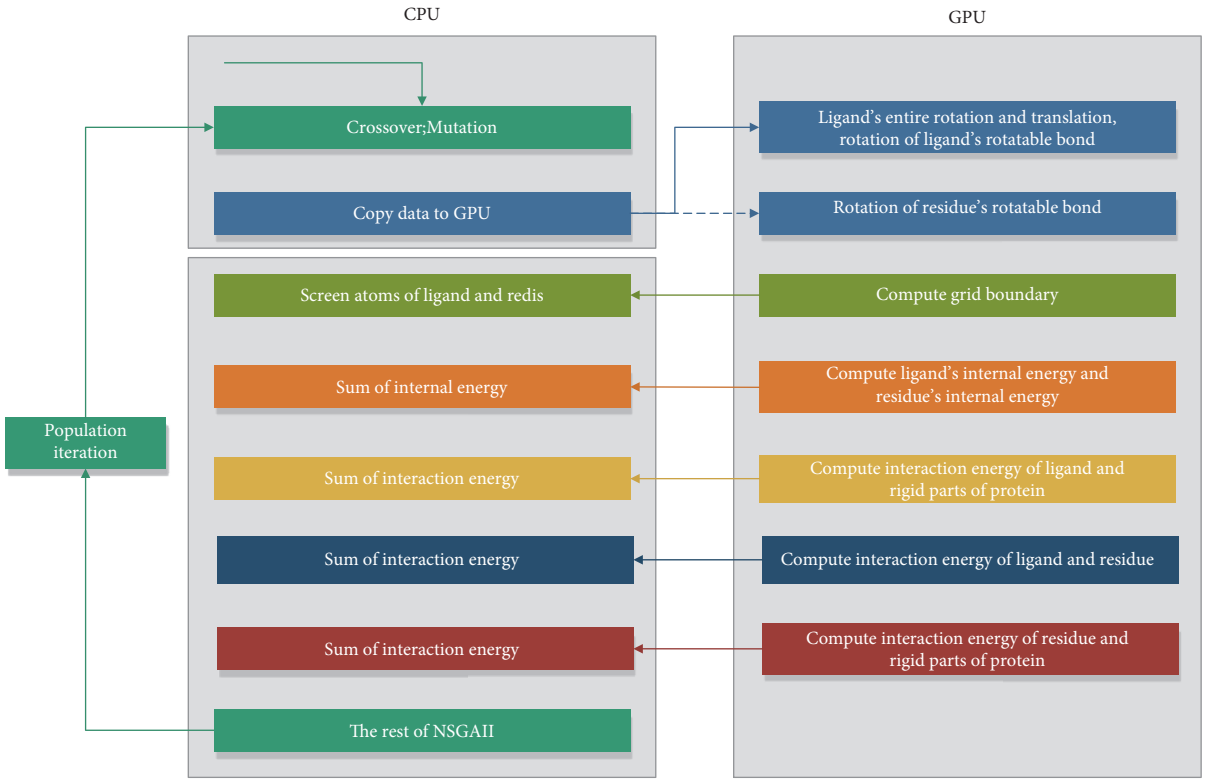


Figure 5: CPU-GPU parallel implementation of one iteration process in flexible docking for one lattice.

are backed up to the CPU side. The rotations of the ligand's rotational bonds and residue's rotational bonds are calculated in the same way.

Before calculating the objective function, the program in GPU determines the energy grid boundary. After the grid boundary is sent to the CPU, the program in the CPU filters out the atoms, which are out of the boundary or in the benzene ring and other rings. Then, the data of the atoms involved in the computation are copied into the GPU.

Afterward, the program steps into the computation of the three objective functions. These objective functions are $f_1(x)$ in equation (2), $f_2(x)$ in equation (3), and $f_3(x)$ in equation (2), respectively. The program in GPU similarly computes all energies, and the CPU program is in charge of the sum of the energies of the same kind.

Here, the interaction energy between the flexible residues and the ligand, i.e., $\Delta G_{\text{inter\_}L-fR}$ in equation (3), is used to illustrate its parallelization in GPU. In computing the interaction energy, because one atom of the ligand and one atom belonging to the residues must interact with each other, the two atoms are considered as an atom pair and the interaction energy is the summation of each interaction energy between one atom pair. The kernel function of GPU sets the number of threads to the number of atom pairs, so each thread can compute the energy of each individual's atom pair. All threads can be organized into a three-dimensional structure, with the number of threads in each block set to $(8 * 8 * 8)$ and the number of the grid set to ((population size $*$ ligand's atomic number $*$ residue's atomic number)/$(8 * 8 * 8)$). The CPU program calculates the cumulative sum of the interaction energies after all interaction energies are computed in the same way.

All objective functions are similarly computed.

Finally, the program in the CPU performs the remaining steps of NSGA-II. Then, the iteration is finished.

Similarly, the GPU-based parallel implementation of semiflexible docking is similar to flexible docking. Although our preliminary work [41] has given some details about GPU-based parallelization of semiflexible docking for iFitDock, the current implementation has already improved and optimized the semiflexible docking in terms of performance. Moreover, it has also parallelized the flexible docking of iFitDock.

3. Results and Discussion

The parallel program is developed in C++, CUDA, and MPI under Linux operating system.

Ideally, for an MPI + GPU program and the experiments, it is better to test the program on a multinode server cluster where each node is with a GPU or several GPUs. However, instead of the above server cluster, regular users can have a better chance to access two typical HPC environments: a multicore server with GPUs and a multinode supercomputing environment without GPU. Here, such two typical environments are chosen as experimental environments:

(Multicore processor & GPUs system 1) A Linux (Red Hat 5.8) system with an Intel Xeon E5-2650 CPU processor with 8 cores and 4 Nvidia Tesla M2090 graphic cards.

(Virtual distributed memory system 1) A Linux (Red Hat 4.8.5–44) system with an Intel Xeon(R) Gold 6266C CPU processor, 16 physical cores, and 32 logical cores. The system uses Docker virtualization technology, and each Docker container is considered as a node.

Multicore processor & GPUs system 1 has a processor with 8 cores and 4 GPUs, and testing experiments can evaluate the whole MPI + GPU program when each core handles one MPI process. However, *Virtual distributed memory system* 1 does not include any GPU, so the fine-grain layer of parallelism, *i.e.*, the GPU module of the program, does not work, and the testing experiments only evaluate the MPI module of the program.

Considering that conformational changes during the ligand-receptor binding process greatly affect the docking effect, 6 compounds were selected for flexible docking experiments and all compounds have larger conformational changes of residues during the binding process, while 10 compounds were taken for semiflexible docking experiments and all compounds have larger conformational changes of ligands during the binding process. All compound files of crystal structure for the experiments were downloaded from the RCSB database (https://www.rcsb.org).

File preparation of the flexible docking experiments and that of the semiflexible docking experiments is described in Supplementary Materials. In the Materials, Supplementary Table 1 illustrates the selection of flexible residues of 6 compounds and the number of atoms and rotatable bonds of each input (ligand, receptor, and flexible residues), the Supplementary Figure 1 illustrates the process of preparing files for the β2 adrenergic receptor (β2AR) in the flexible docking experiment, and the Supplementary Figure 2 illustrates the process of preparing files for the carboxypeptidase A (1CPS) in the semiflexible docking experiment.

3.1. Docking Speedup on the Multicore Server with GPUs. To obtain the speedup ratio of the parallel program, the parallel program and the serial program are tested on a *Multicore processor & GPUs system* 1. We have picked 50 lattices that enclose the possible binding pockets and the ligands entering and leaving paths for experiments. The running time of the parallel program and serial program, and the speedup ratios obtained by parallel programs in simulation experiments are shown in Figure 6.

In the flexible docking experiments, as shown in Figure 6(a), the compound with the best performance is **PPAR-β**, and the parallel program achieves a speedup ratio as high as 45 times. However, the compound with the worst performance ratio is **PPAR-α**, and the parallel program achieves a speedup ratio as high as 28 times. Here, the running time and the speedup ratio for a compound are mainly determined by the number of all atoms in the flexible residues of the protein. In different compounds, the number of all atoms in the flexible residues varies due to several factors such as the width and the position of the protein tunnel, and it determines the computing time of the objective functions and furthers the whole running time.

In semiflexible docking experiments, as shown in Figure 6(b), the compound with the best performance is **1DR1**, and the parallel program achieves a speedup ratio as high as 54.5 times. Meanwhile, the compound with the worst performance is **1DID**, and the parallel program achieves a speedup ratio as high as 17.5 times. Here, the running time and the speedup ratio for a compound are mainly determined by the number of all atoms and the number of all rotational bonds in the ligand. The ligand of **1DR1** has much more atoms and rotational bonds than the ligand of **1DID**, which results in a longer docking calculation time. In the semiflexible experiment, the number of atoms in the ligand becomes a more important factor that determines the computing time of the objective functions.

The results of the experiments for the ligand-receptor binding process on the *Multicore processor & GPUs system 1* demonstrate the speedup ratio of the parallel program, with achieved a speedup ratio as high as 45 times in flexible docking and as high as 54.5 times in semiflexible docking, and it indicates that the speedup effect of the parallel program is productive.

3.2. Docking Speedup on the Distributed Memory System. To obtain the speedup of the parallel program, the parallel program and the serial program are also tested on *Virtual distributed memory system 1*. We have picked 150 lattices that enclose the possible binding pockets and the ligand entering and leaving paths for experiments on the distributed memory system.

The docking times of the flexible experiment and those of the semiflexible docking experiments on the distributed system are shown in Figure 7. The docking time for flexible docking is much greater than the docking time for semiflexible docking.

According to the grouped strategy, when the parallel program is running on the distributed system (32 logical nodes), all nodes are organized into groups "1" and "2," respectively, to verify the validation of grouped master-slave mode. Here, two compounds, namely, **1 cps** and **1did,** are used for semiflexible experiments, and another two compounds, namely, **β2AR** and **PPAR-α,** are used for flexible experiments. As shown in Figure 8, the parallel program is running on 32 nodes, the docking time of all compounds with "2" groups is shorter than the "1" group on the distributed system. The reason is that 2 master processes alleviate communication blocking. In the "1" group, the only master process suffers from the heavy communication cost among itself and the slaves, and it becomes a bottleneck when the number of slave processes is 31 (32 minus the master). In "2" groups, two master nodes allocate docking tasks and receive results, although the number of slave processes is 30 (32 minus two masters).

The results of the experiments for the ligand-receptor binding process on a distributed memory system reflect the speedup trend of the parallel program, which indicates the docking time for flexible docking and that for semiflexible docking gradually decreases as the number of nodes used in the parallel program gradually increases. Furthermore, the result of grouped experiments demonstrates the effectiveness of the grouped strategy.

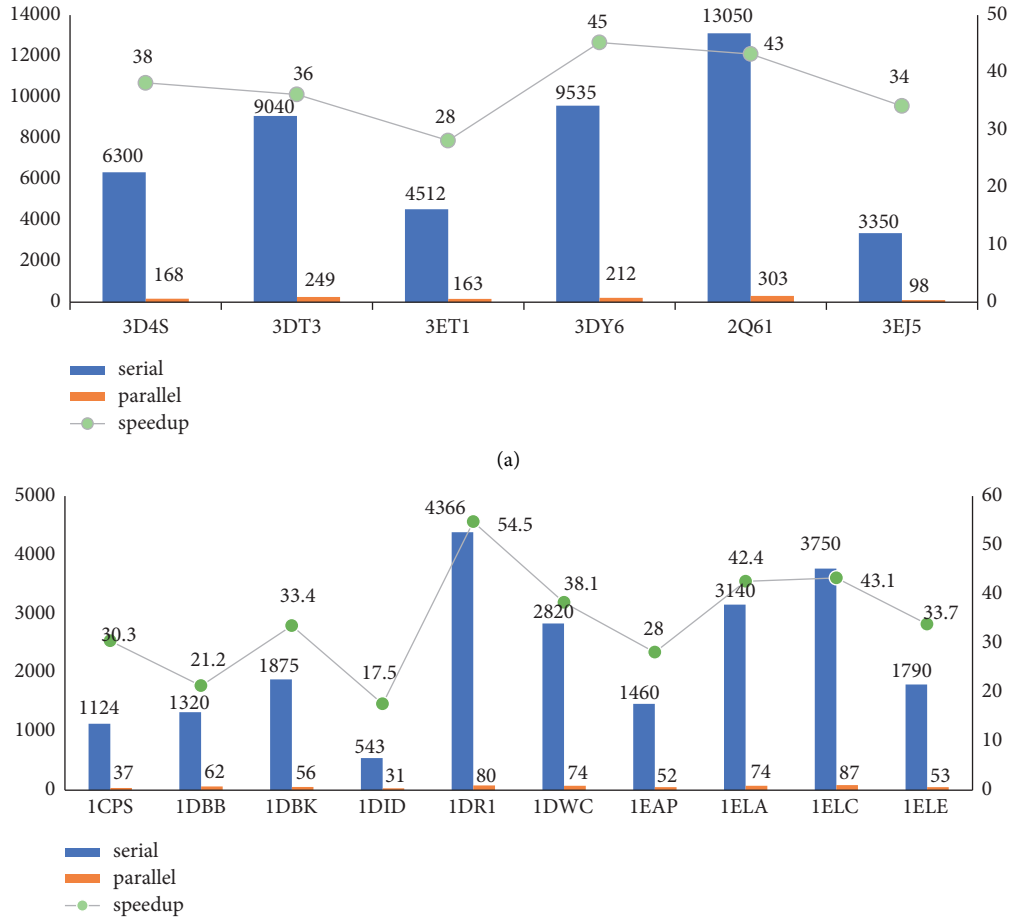


Figure 6: The running time of the parallel program and serial program (in minutes), and the speedup ratios obtained by parallel programs in simulation experiments. The bar chart indicates the running time of the serial program and parallel program, and the line graph represents the speedup ratios obtained by the parallel program in the simulation experiment. (a) Results of flexible docking experiments. (b) Results of semiflexible docking experiment.

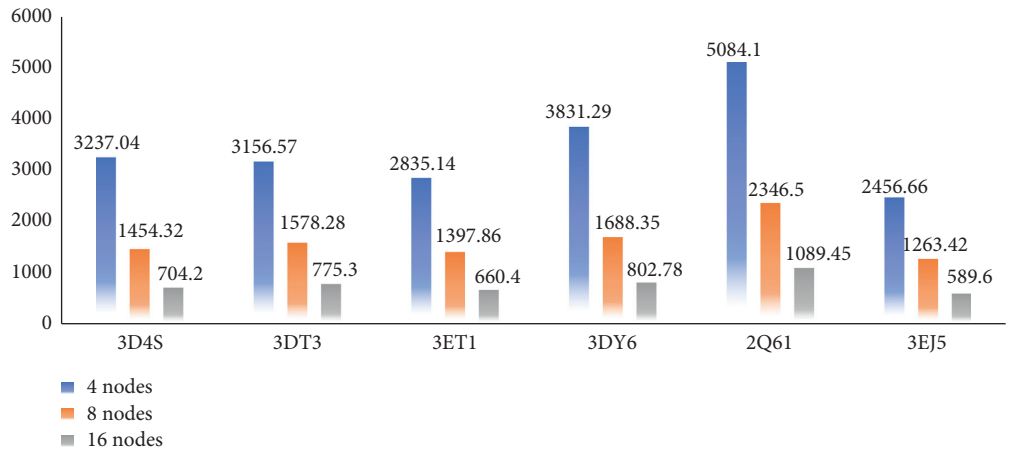
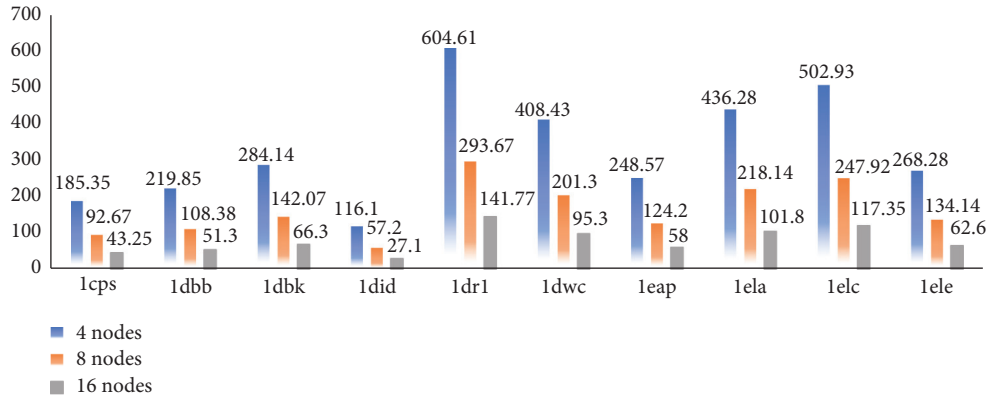


Figure 7: Continued.



(b)

Figure 7: The running time of the parallel program on 4, 8, and 16 nodes (in minutes). (a) The running time of the parallel program in flexible docking experiment. (b) The running time of the parallel program in semiflexible docking experiment.

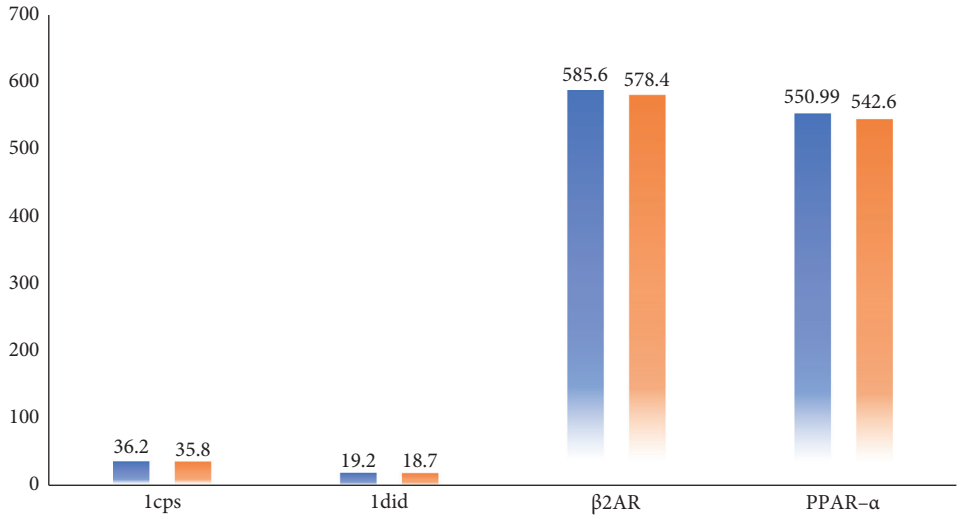


Figure 8: The running time of the parallel program on 32 nodes (in minutes).

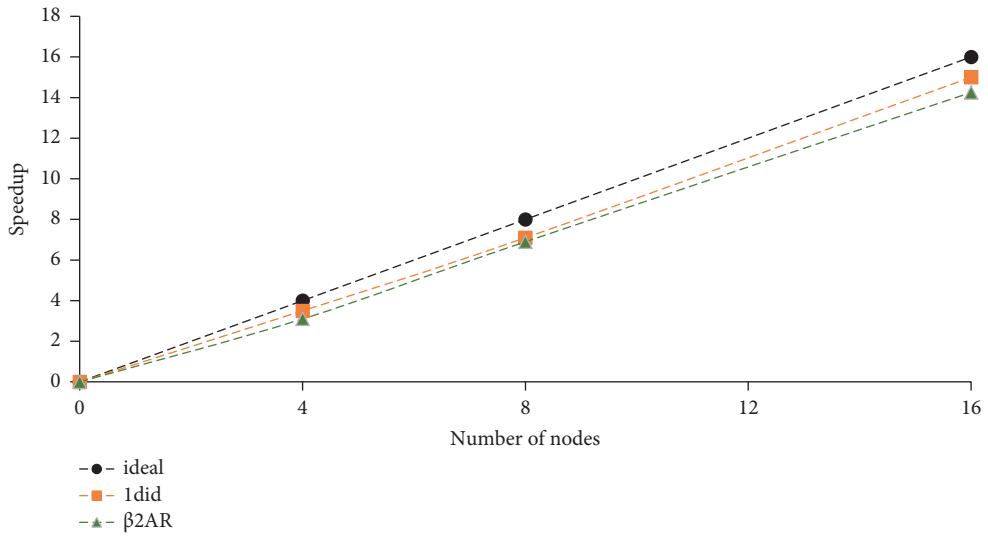


Figure 9: Scaling on the distributed memory system. The speedup ratios are obtained for the parallel program is running on 4, 8, and 16 nodes.

3.3. Scalability on the Distributed Memory System. To prove the scalability of the parallel program, scalability experiments are tested on a *Virtual distributed memory system 1*. Considering the number of the physical cores of the distributed system is 16, the parallel programs, respectively, run on 4, 8, and 16 nodes to get results. Here, **1did** is used for the semiflexible scalability experiment, and ***β*2AR** is for the flexible scalability experiment.

Considering that the docking calculations within each lattice are independent of each other, and the exchange of data between the MPI master and slave processes only takes place at the beginning and end of the docking, the scalability of the parallel program should be linear. When the parallel program is running on 4, 8, and 16 nodes, the speedup upper bounds should be 4, 8, and 16, respectively. The actual speedup ratios are shown in Figure 9, and all ratios are close to the corresponding speedup upper bounds. The results have proved that the scalability of the parallel program is approximately linear.

4. Conclusions

In this study, we have introduced a heterogeneous parallel scheme of molecular docking for the ligand-receptor binding process. By using MPI and GPU, we construct a two-layer parallelism framework and develop a corresponding parallel program. The results of the experiments for the ligand-receptor binding process show that on a multicore server with GPUs the parallel program has achieved a speedup ratio as high as 45 times in flexible docking and as high as 54.5 times in semiflexible docking, and on a distributed memory system the docking time for flexible docking and that for semiflexible docking gradually decreases as the number of nodes used in the parallel program gradually increases. The scalability of the parallel program is also verified in multiple nodes on a distributed memory system and is approximately linear. The parallel program provides strong support for the construction of a rigorous and accurate binding free energy landscape, and further estimation of binding affinity and binding kinetics.

Data Availability

The files of crystal structure for the experiments were downloaded from the RCSB database (https://www.rcsb.org).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Fundamental Research Funds for National Key R&D Program of China (under grant no. 2016YFA0502304) and the Important Drug Development Fund, the Ministry of Science and Technology of China (under grant no. 2018ZX09735002).

Supplementary Materials

Appendix A: selection of flexible residues and preparation for the flexible docking experiments. Appendix B: preparation for the semiflexible docking experiments. (*Supplementary Materials*)

References

[1] F. Leonardo, D. S. Ricardo, O. Glaucius, and A. Adriano, "Molecular docking and structure-based drug design strategies," *Molecules*, vol. 20, Article ID 13384, 2015.

[2] Y. Chen, Y. Guo, Y. Pan, and Z. J. Zhao, "Structure analysis of the receptor binding of 2019-nCoV," *Biochemical and Biophysical Research Communications*, vol. 525, no. 1, pp. 135–140, 2020.

[3] D. B. Kokh, R. C. Wade, and W. Wenzel, "Receptor flexibility in small-molecule Docking calculations," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 1, pp. 298–314, 2011.

[4] K. W. Lexa and H. A. Carlson, "Protein flexibility in docking and surface mapping," *Quarterly Reviews of Biophysics*, vol. 45, no. 3, pp. 301–343, 2012.

[5] G. M. Morris, R. Huey, W. Lindstrom et al., "Automated docking with selective receptor flexibility," *Journal of Computational Chemistry*, vol. 30, 2009.

[6] W. J. Allen, T. E. Balius, S. Mukherjee et al., "Dock 6: impact of new features and current docking performance," *Journal of Computational Chemistry*, vol. 36, no. 15, pp. 1132–1156, 2015.

[7] O. Trott and A. J. Olson, "Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *Journal of Computational Chemistry*, vol. 31, 2009.

[8] W. H. Shin and C. Seok, "GalaxyDock: protein-ligand docking with flexible protein side-Chains," *Journal of Chemical Information and Modeling*, vol. 52, no. 12, pp. 3225–3232, 2012.

[9] C. R. Corbeil and N. Moitessier, "Docking ligands into flexible and solvated macromolecules. 3. Impact of input ligand conformation, protein flexibility, and water molecules on the accuracy of docking programs," *Journal of Chemical Information and Modeling*, vol. 49, no. 4, pp. 997–1009, 2009.

[10] M. Totrov and R. Abagyan, "Flexible ligand docking to multiple receptor conformations: a practical alternative," *Current Opinion in Structural Biology*, vol. 18, no. 2, pp. 178–184, 2008.

[11] C. McInnes, "Virtual screening strategies in drug discovery," *Current Opinion in Chemical Biology*, vol. 11, no. 5, pp. 494–502, 2007.

[12] S. R. Ellingson, J. C. Smith, and J. Baudry, "VinaMPI: facilitating multiple receptor high-throughput virtual docking on high-performance computers," *Journal of Computational Chemistry*, vol. 34, no. 25, pp. 2212–2221, 2013.

[13] B. Collignon, R. Schulz, J. C. Smith, and J. Baudry, "Task-parallel Message Passing Interface Implementation of Autodock4 for Docking of Very Large Databases of Compounds Using High-Performance Super-computers," in *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, USA, August 2011.

[14] A. P. Norgan, P. K. Coffman, J. P. Kocher, D. J. Katzmann, and C. P. Sosa, "Multilevel parallelization of AutoDock 4.2," *Journal Of Cheminformatics*, vol. 3, 2011.

[15] X. Zhang, S. E. Wong, and F. C. Lightstone, "Message passing interface and multithreading hybrid for parallel molecular docking of large databases on petascale high performance

computing machines," *Journal of Computational Chemistry*, vol. 34, pp. 915–927, 2013.

[16] A. Peters, M. E. Lundberg, P. Therese Lang, and C. P. Sosa, "High throughput computing validation for drug discovery using the DOCK program on a massively parallel system," in *Proceedings of the 1st Annual Midwest Symposium on Computational Biology and Bioinformatics*, Washington, DC, USA, September 2007.

[17] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten, "GPU-accelerated molecular modeling coming of age," *Journal of Molecular Graphics and Modelling*, vol. 29, pp. 116–125, 2011.

[18] I. Baldomero, S. Antonio, B. C. Andrés, L. A. José, P. S. Horacio, and M. C. José, "Metadock 2: a high-throughput parallel metaheuristic scheme for molecular docking," *Bioinformatics*, vol. 37, 2020.

[19] B. L. Imbernon and A. Lozano, J. M. Cutillas-Lozano and D. Giménez, HYPERDOCK: Improving virtual screening through parallel hyperheuristics," *International Journal of High Performance Computing Applications*, vol. 34, 2020.

[20] D. Santos-Martins, L. Solis-Vasquez, A. F. Tillack, M. F. Sanner, A. Koch, and S. Forli, "Accelerating AutoDock4 with GPUs and gradient-based local search," *Journal of Chemical Theory and Computation*, vol. 17, no. 2, pp. 1060–1073, 2021.

[21] O. Korb, T. Stützle, and T. E. Exner, "Accelerating molecular docking calculations using graphics processing units," *Journal of Chemical Information and Modeling*, vol. 51, no. 4, pp. 865–876, 2011.

[22] X. Ding, Y. Wu, Y. Wang, J. Z. Vilseck, and C. L. Brooks, "Accelerated CDOCKER with GPUs, parallel simulated annealing, and fast fourier transforms," *Journal of Chemical Theory and Computation*, vol. 16, no. 6, pp. 3910–3919, 2020.

[23] D. Yun, F. Ye, P. F. Wei, J. Ramanujam, and M. Jarrell, "GeauxDock: A. Novel, Approach for mixed-resolution ligand docking using a descriptor-based force field," *Journal Of Computational Chemistry*, vol. 36, 2015.

[24] Y. Fang, Y. Ding, W. P. Feinstein et al., "GeauxDock: accelerating structure-based virtual screening with heterogeneous computing," *PLoS One*, vol. 11, 2016.

[25] S. Altunta, Z. Bozkus, and B. B. Fraguela, "GPU accelerated molecular docking simulation with genetic algorithms," in *Proceedings of the European Conference On the Applications Of Evolutionary Computation*, Porto, Portugal, April 2016.

[26] J. Wang and N. V. Dokholyan, "MedusaDock 2.0: efficient and accurate protein-ligand docking with constraints," *Journal of Chemical Information and Modeling*, vol. 59, no. 6, pp. 2509–2515, 2019.

[27] M. Fan, J. Wang, H. Jiang et al., "GPU-accelerated flexible molecular docking," *The Journal of Physical Chemistry B*, vol. 125, no. 4, pp. 1049–1060, 2021.

[28] X. Jiang, K. Kumar, X. Hu, A. Wallqvist, and J. Reifman, "Dovis 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0," *Chemistry Central Journal*, vol. 2, no. 1, 2008.

[29] I. Faraji and A. Afsahi, "Design Considerations for GPU-Aware Collective Communications in MPI," *Concurrency And Computation: Practice and Experience*, vol. 30, 2018.

[30] J. M. Hashmi, C. H. Chu, S. Chakraborty et al., "Zero-copy MPI derived datatype processing on modern CPU and GPU Architectures," *Journal of Parallel and Distributed Computing*, vol. 144, 2020.

[31] C. H. J. Eckert, E. Zenker, M. Bussmann, and D. Albach, "HASEonGPU-An adaptive, load-balanced MPI/GPU-code for calculating the amplified spontaneous emission in high power laser media," *Computer Physics Communications*, vol. 207, pp. 362–374, 2016.

[32] J. Choi, Y. Kim, and H. Y. Yeom, "Overcoming GPU Memory Capacity Limitations in Hybrid MPI Implementations of CFD," in *Proceedings of the International Conference on Internet and Distributed Computing Systems*, pp. 100–111, Naples, Italy, October 2019.

[33] J. Lai, H. Yu, Z. Tian, and H. Li, "Hybrid MPI and CUDA parallelization for CFD applications on multi-GPU HPC clusters," *Scientific Programming*, vol. 2020, Article ID 8862123, 15 pages, 2020.

[34] D. Komatitsch, G. Erlebacher, D. Göddeke, and D. Michéa, "High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster," *Journal of Computational Physics*, vol. 229, no. 20, pp. 7692–7714, 2010.

[35] R. Elgohary, H. Khaled, H. Faheem, and M. Elkott, "Multi-class support vector achine training and classification based on MPI-GPU hybrid parallel architecture," in *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics*, pp. 179–188, Cairo, Egypt, September 2018.

[36] F. Bai, Y. Xu, J. Chen et al., "Free energy landscape for the binding process of Huperzine A to acetylcholinesterase," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, 2013.

[37] M. Petřek, P. Košinová, J. Koča, and M. Otyepka, "MOLE: A Voronoi diagram-based explorer of molecular channels, pores, and tunnels," *Structure*, vol. 15, pp. 1357–1363, 2007.

[38] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[39] P. A. Greenidge, C. Kramer, J.-C. Mozziconacci, and R. M. Wolf, "MM/GBSA binding energy prediction on the PDBbind data set: successes, failures, and directions for further improvement," *Journal of Chemical Information and Modeling*, vol. 53, no. 1, pp. 201–209, 2013.

[40] T. Hou, J. Wang, Y. Li, and W. Wang, "Assessing the performance of the MM/PBSA and MM/GBSA methods. 1. The accuracy of binding free energy calculations based on molecular dynamics simulations," *Journal of Chemical Information and Modeling*, vol. 51, no. 1, pp. 69–82, 2011.

[41] J. Xu, J. Li, and Y. Cai, "Molecular Docking Simulation Based on CPU-GPU Heterogeneous Computing," in *Proceedings of the International Workshop On Advanced Parallel Processing Technologies 2017*, Cham, Vietnam, September 2017.