Vina-FPGA: A Hardware-Accelerated Molecular Docking Tool With Fixed-Point Quantization and Low-Level Parallelism

Ming Ling, *Member, IEEE*, Qingde Lin, Ruiqi Chen, *Member, IEEE*, Haimeng Qi, Mengru Lin, *Student Member, IEEE*, Yanxiang Zhu, *Member, IEEE*, and Jiansheng Wu

***Abstract*— Molecular docking (MD) is one of the core steps in the expensive and time-consuming process of drug design, which is basically an optimization problem based on scoring functions. AutoDock series MD software is widely accepted by academia and industry, among which AutoDock Vina (Vina) is the latest and most popular version due to its accuracy and relatively high speed. However, contrast to its prior version, i.e., AutoDock4, hardware acceleration approaches of Vina are rarely reported. In this article, we propose Vina-field-programmable gate array (FPGA), a hardware-accelerated Vina implementation with FPGA that exploits the low-level parallelism. First, the fixed-point quantization is analyzed and realized to accelerate the MD algorithm with a better energy efficiency in hardware. To boost the performance of the module-level computation, multiple in-module hardware pipelines have been designed and implemented. Besides, a strategy for fast accessing to block RAM (BRAM) is implemented by utilizing the layout of data, which brings four times memory access speed to the intermolecular and intramolecular energy computing modules. Under the same 140 ligand–receptor benchmarks, Vina-FPGA performs up to 6.9× (average 3.7×) faster than a state-of-the-art CPU does while consuming only 2.5% energy with similar docking accuracies. Compared to the GPU-accelerated implementation or Vina-GPU, the average energy consumption of Vina-FPGA is merely 45%.**

***Index Terms*— AutoDock Vina, field-programmable gate array (FPGA), hardware accelerator, optimization algorithm.**

I. Introduction

DRUG design has always been the forefront issue in pharmaceutical industries, which normally takes more than ten years of research and costs over 10 billion dollars [1]. Molecular docking (MD) is a core step in the current drug design process for fast screening candidate drug molecules with computer algorithms, whose essence is an optimization problem. The scoring functions are used to quantitatively evaluate the docking accuracies [2], while a specific search algorithm is leveraged to speed up the optimal results searching process in the huge potential solution space. Different MD software are equipped with different search algorithms and scoring functions.

AutoDock Vina (Vina) [3] is an outstanding MD tool that obtains the highest score on the latest test set CASF-2016 and outperforms all the other docking tools [4]. Despite the similarity of their names, AutoDock4 [5] and AutoDock Vina are different software with distinguishing scoring functions and searching algorithms. Vina is much more preeminent than AutoDock4 in terms of docking speed and docking accuracy [3]. Nevertheless, the computation process of Vina is still painfully time-consuming due to the massive combinatorial possibilities of multidimensional data. The searching algorithm used by Vina is the Iterated Local Search Global Optimizer (ILSGO), which completes the global search (GS) with a variant of multi-initial states simulated annealing (SA) algorithm [6], [7], while a quasi-Newton method is used for a gradient descent local search (LS) (in Vina's case, Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) [3]). Although it is possible to leverage the parallelism by a multicore platform in Vina's multi-initial states SA GS, which, in fact, is a built-in feature of Vina via the Boost Library [8], hardware accelerations of the SA and BFGS iterations are significantly difficult due to the highly serialized and irregular processes in them.

Some researchers focused on the Vina acceleration problem via algorithmic improvements, such as the scoring function optimizations [9], [10] and computing process optimizations [11], [12], [13]. However, the acceleration effects of such approaches are so far unsatisfactory. Apart from that, VirtualFlow [14], an open-source drug design platform that equips Vina as the virtual screening tool, greatly reduced huge-scale screening time by 160 000 CPUs. However, this resource is almost unaffordable for most researchers, not to mention its huge power consumption. Besides, VirtualFlow only focuses on the multicore parallelism rather than promoting the single-core operation speed, which means that the speedup ratio only comes from abundant CPU resources. Some prior studies have tried the feasibilities of hardware-based Vina accelerations. For example, Viking is a pioneer exploration in GPU acceleration for Vina [15]. Unfortunately, due to the iterative nature of the searching algorithm, Viking even takes longer computing time on GPUs than the original CPU version.

In this article, we propose Vina-field-programmable gate array (FPGA), a hardware-accelerated Vina implementation with FPGA. The parallelism of Vina is analyzed and

Manuscript received 4 May 2022; revised 31 July 2022 and 21 September 2022; accepted 20 October 2022. Date of publication 4 November 2022; date of current version 22 March 2023. This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 61974024. *(Corresponding authors: Ming Ling; Ruiqi Chen.)*

Ming Ling, Qingde Lin, and Haimeng Qi are with the National ASIC System Engineering Technology Research Center, Southeast University, Nanjing 210096, China (e-mail: trio@seu.edu.cn; lqd@seu.edu.cn; 727692147@qq.com).

Ruiqi Chen is with the Zhangjiang Fudan International Innovation Center, Fudan University, Shanghai 200433, China (e-mail: ruiqichen@ieee.org).

Mengru Lin and Yanxiang Zhu are with the VeriMake Innovation Laboratory, Nanjing Renmian Integrated Circuit Company Ltd., Nanjing 210088, China (e-mail: linmengru@verimake.com; yanxiangzhu@verimake.com).

Jiansheng Wu is with the School of Geographic and Biological Information, Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: jansen@njupt.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3217275.

Digital Object Identifier 10.1109/TVLSI.2022.3217275

categorized into three levels, namely, low-level parallelism (LLP), mid-level parallelism (MLP), and high-level parallelism (HLP). As we analyzed in Section II.B, Vina is equipped with a built-in mechanism for the HLP, while the MLP is significantly hard for the hardware acceleration due to the iterative and irregular characteristics of Vina algorithm. Therefore, a hardware-accelerated architecture, or Vina-FPGA, in the LLP is proposed and realized on Xilinx XCKU060. The key contributions of this work are given as follows.

1) Both the ILSGO and BFGS algorithms in Vina possess the characteristics of strong dependencies and irregularities, which means that it is difficult to parallelize the iterations of BFGS and ILSGO. Fortunately, we were able to find parallelism at a lower level, such as the basic computing modules in BFGS. Therefore, hardware pipelines are widely adopted in those modules in Vina-FPGA. To the best of our knowledge, Vina-FPGA is the first reported FPGA accelerator with a significant speedup for AutoDock Vina. Moreover, compared to the CPU and GPU implementations under 140 ligand–receptor benchmarks, Vina-FPGA averagely consumes only 2.5% and 45% energy, respectively.
2) The intermolecular and intramolecular energy computing modules are the core steps of scoring function calculation, which are based on a group of serial and time-consuming tables looking up. Thus, this article realizes the parallel calculation of intermolecular energy and intramolecular energy with their internal pipelines to improve the system performance.
3) To lower the computation complexity, the intermolecular energy calculation is implemented by trilinear interpolations based on the precalculated grid data from the host CPU. Besides, each trilinear interpolation calculation requires eight precalculated grid data, which are stored in BRAM. However, limited by the number of BRAM ports (up to two ports), it takes at least four cycles to obtain these eight values in a normal design. This article utilizes the characteristics of data organization and maps the values of the vertex points in the grid to four separate BRAM blocks. Thus, the vertex data used in the trilinear interpolations will be evenly distributed in the four BRAM blocks such that the data needed to complete a trilinear interpolation can be accessed in one cycle.

The rest of this article is structured as follows. Section II introduces the background and motivation of this article. Section III provides the proposed architecture and design of Vina-FPGA. Section IV gives the details of the experimental methodology for the acceleration and accuracy evaluations of Vina-FPGA and discusses the results. Section V reports the related work, while Section VI provides the conclusion and looks out to the future work.

II. Backgrounds

A. Searching Algorithm of Vina

The nature of an MD algorithm is an optimization problem, which searches for the best ligand conformation (Con) in the preset searching space or a docking box, that is, the ligand, or the candidate drug molecule, in this configuration has the lowest energy when it attaches to the given receptor (normally, a target protein). A docking box is the possible binding area of the given receptor molecule, which is normally

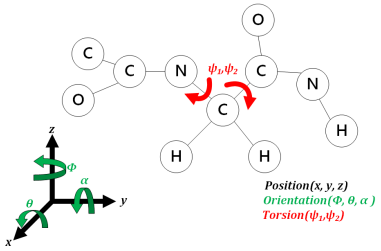


Fig. 1. POT of a ligand.

Algorithm 1 Iterated Local Search Global Optimizer

Input: $Con \in [Position(x,y,z), Orientation(\emptyset,\theta,\alpha), Torsion(\psi_T)]$
Input: $N_{atomA} \in (A = \{1,2,\ldots,108\}), N_{TorsionT} \in (T = \{1,2,\ldots,32\})$
Output: $Coords_n$ for n = 1 to N $(1 \le N \le 20)$
Output: f

```
1: for exhaustiveness do
2:     Random Con
3:     for i ≤ (105 * N_atom + 1050 * (N_torsion + 6) + 5250) do
4:         i++
5:         mutate(Con)
6:         Con, COORD, f_j = BFGS(Con)
7:         if MetropolisAccept(f_j, f) then
8:             Con, COORD, f_k = BFGS(Con)
9:             //For the accepted Con, a finer BFGS search is conducted
10:            Coords_n = Insert(COORD)
11:            f = f_k
12:        else
13:            Continue
```

determined by a pharmaceutical professional and stored in the initialization file of Vina. It is easy to understand that Vina evaluates the energy among the ligand–receptor complex as the objective, or the scoring function, which is the sum of intramolecular energy (the energy among the atoms inside the ligand molecule) and intermolecular energy (the energy between the ligand molecule and the receptor). To simplify and speed up the energy evaluation process, Vina looks up the tables that store precalculated energy data to calculate both intramolecular and intermolecular energy, which will be discussed in Section III.D.

The Con of the ligand is determined by three factors, including position, orientation, and torsion, as shown in Fig. 1, which represent the spatial coordinates (three degrees of freedom, x, y, and z in Fig. 1), the Euler angle transformations (three degrees of freedom, α, θ, and φ in Fig. 1), and the torsion angles (degrees of freedom depending on the number of rotatable branches in the ligand molecule, i.e., one rotatable branch adding one extra degree of freedom, e.g., $\psi 1$ and $\psi 2$ in Fig. 1) of the rotatable branches of the ligand, respectively. Obviously, a huge combination of Con exists for the optimal ligand Con searching, which requires huge computational effort.

Vina uses a method called ILSGO to complete this mission, as shown in Algorithm 1. First, random generates an initial Con in the search space randomly [16]. The optimal Con is then investigated by the iterative search (from line 3 to line 13), the search depth (iteration times) of which is empirically determined by the number of atoms (Natom) and the number of rotatable branches (Ntorsion) in the ligand, as shown in line 3. It is worth noting that the constants (105, 1050, 6, and 5250) in line 3 are empirically preset by the Vina algorithm, which are not discussed in this article. As in a classical SA algorithm, a mutate of current Con will be carried out,

Algorithm 2 BFGS

Input: $Con \in [Position(x,y,z), Orientation(\emptyset,\theta,\alpha), Torsion(\psi_T)]$
Output: $Con \in [Position(x,y,z), Orientation(\emptyset,\theta,\alpha), Torsion(\psi_T)]$
Output: COORD $\in [\vec{v}_0, \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{Natom-1}]$, $\vec{v}_j \in [x_j, y_j, z_j]$
Output: f
1: **initial** H_0
2: $Con_0 = Con$
3: Con_0 **Convert to** $COORD_0$, XK_0
4: $energy_{Inter} =$ **Inter-molecular**(XK_0)
5: $energy_{Intra} =$ **Intra-molecular**($COORD_0$)
6: $f_0 = energy_{Intra} + energy_{Inter}$
7: $g_0 =$ **Derivation**($COORD_0$)
8: **for** j $\leq$ ***uint***((25+N_{atom}) / 3) **do**
9: j++
10: $d_j = -H_j \times g_j$
11: $\alpha, f_{j+1}, g_{j+1}, Con_{j+1} =$ **Armijo-Goldstein**(Con_j, d_j, f_j, g_j)
12: **if** $||g_{j+1}||_2 < 10^{-5}$ **then**
13: **Break**
14: $H_{j+1} =$ **H-update**($\alpha, H_j, g_{j+1}, g_j$)
15: $Con, f, COORD = Con_{j+1}, f_{j+1}, COORD_{j+1}$

Algorithm 3 Armijo–Goldstein

Input: $Con \in [Position(x,y,z), Orientation(\emptyset,\theta,\alpha), Torsion(\psi_T)]$
Input: d, f, g
Output: $Con \in [Position(x,y,z), Orientation(\emptyset,\theta,\alpha), Torsion(\psi_T)]$
Output: α, f, g, Con
1: $Con_0 = Con$
2: $f_0 = f$
3: **for** j = 0; j $\leq$ 10; j++ **do**
4: $Con_{j+1} = Con_j + \alpha \times d_j$
5: Con_{j+1} **Convert to** $COORD_{j+1}$, XK_{j+1}
6: $energy_{Inter} =$ **Inter-molecular**(XK_{j+1})
7: $energy_{Intra} =$ **Intra-molecular**($COORD_{j+1}$)
8: $f_{j+1} = energy_{Intra} + energy_{Inter}$
9: $g_{j+1} =$ **Derivation**($COORD_{j+1}$)
10: **if** ((f_{j+1} - f_j) < 0.0001 $\times \alpha \times d \times g_j$ **then**
11: **Break**
12: **else**
13: $\alpha = \alpha / 2$
14: $\alpha, f, g, Con = \alpha, f_{j+1}, g_{j+1}, Con_{j+1}$

i.e., the values in either POT will be randomly changed at the beginning of each iteration. The LS of ILSGO is realized by a gradient descent method carried out by BFGS (line 6). At the end of the BFGS iteration, comparisons between the values of pre and post scoring function fs are performed by MetropolisAccept. If the result is accepted, a new BFGS iteration is executed for a finer search with a more precise step stride (line 8); otherwise, a new ILSGO iteration will be started. After the second iteration of BFGS, the result energy and Cons are compared with the members of the output queue with a maximum length of 20 that defined by Vina. The Cons with 20 lowest energy are inserted into the queue (line 10).

The BFGS method is shown in Algorithm 2, which is divided into two modules, namely, the linear search module Armijo–Goldstein (AG) and Hessian matrix updating module. The search direction d could be obtained by Hessian matrix updating (line 14), while the search step α could be acquired by the linear search (line 11). The searching loop can be early stopped when the gradient stops declining (line 12 and line 13). To calculate the initial value of the objective function f (lines 4–6), which is the sum of the intermolecular energy and the intramolecular energy, and its derivative g (line 7), Vina uses a new data structure, COORD, to record all the absolute coordinates of each atom in the ligand. It is easy to understand that COORD can be derived from Con of a given ligand (line 3). Furthermore, another data structure is also constructed, XK, to record all the coordinates of each atom in the docking box. Since the different ligands own different docking boxes on the receptor, XK could be acquired by performing coordinate transformations on COORD.

The linear search module AG is used for finding the search step, shown in Algorithm 3. The AG method minimizes the search interval by iteratively evaluating the objective function (energyinter + energyintra) until the optimal step size is found in the descending direction.

B. Parallelism Levels of Vina Algorithm

As we discussed in the previous section, the Vina algorithm is implemented by four-layer loops shown in Fig. 2. The first layer (exhaustiveness level) generates multiple initial points independently (line 1 of Algorithm 1), while the second-layer (global searching level or GS in short) loop (line 3–line 13 of Algorithm 1) performs global optimization search based on the given initial point assigned by line 2 in Algorithm 1. Besides, the third-layer (local searching level or LS in short) loop realizes a local optimization search based on a specific ligand Con by BFGS (Algorithm 2), and the fourth-layer (AG level) loop performs a linear search for the searching step determination (Algorithm 3). The built-in mechanism of Vina supports the parallelization of the first layer (exhaustiveness level) by the Boost Library [8]. Due to the independency of the search progress of each initial Con for a given ligand molecule, multiple search threads with corresponding initial Cons could be assigned to different cores in a multicore architecture. Unfortunately, the iterative and irregular nature of the inner three-layer loops, namely, the GS level, LS level, and AG level, make them hard, if not impossible, to be executed concurrently. As we can conclude, the parallelisms of Vina can be divided into three levels, similar to the analysis in [17].

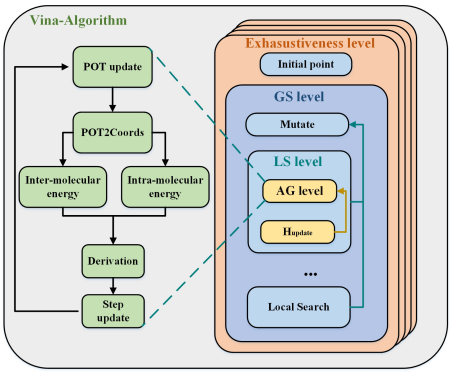


Fig. 2. Four-layer loops structure of Vina.

1) *High-Level Parallelism*: Parallelizing the initial point evaluations on different computing units, which is the exhaustiveness level in Fig. 2.
2) *Mid-Level Parallelism*: Parallelizing Algorithms 1–3, which includes GS, LS, and AG levels in Fig. 2.
3) *Low-Level Parallelism*: Parallelizing the basic modular operations in Armijo–Goldstein (AG) level and LS level of Fig. 2, which includes the four modules in AG level, POT2Coords module, two energy calculation modules (inter and intra), derivation module, and the Hessian matrix updating module (H_{update}) in the LS level.

Although HLP is a built-in feature of Vina, while MLP is hard to be implemented in a hardware architecture, LLP still could be exploited by a hardware-based accelerator.

As we can find in Fig. 2, modules of POT2Coords (line 5 in Algorithm 3), intermolecular and intramolecular energy calculations (lines 6 and 7 in Algorithm 3), derivation (line 9 in Algorithm 3), and H_{update} (line 14 in Algorithm 2) could be implemented in a pipelined manner to accelerate the computing throughput. Moreover, intermolecular and intramolecular energy calculations are independent of each other; therefore, these two modules could also be calculated in parallel.

C. AutoDock Vina and AutoDock4

The key difference between AutoDock 4 (AD4) and AutoDock Vina (Vina) is the searching algorithm. AD4 uses the Lamarckian Genetic Algorithm (LGA) as the global method to generate new entities (i.e., different Cons of the ligand) and selects the stronger ones from the entire population that survive through generations. In addition, LGA is also an LS method that performs an adaptive-iterative process to minimize the energy of randomly chosen entities [29]. Due to the independencies in this process, it is possible for the accelerator developers to exploit this inherent parallelism to speed up the searching. For example, in the FPGA implementation of [28], a three-stage pipeline architecture was proposed to execute the genetic algorithm, ligand position calculation, and energy evaluations concurrently, while in the newer study [29], the authors proposed a more aggressive parallel scheme that adopts nine independent LS hardware modules to accelerate the searching process.

Vina, on the other hand, utilizes a totally different searching method, in which the global searching process is performed by a variant of SA algorithm, while a quasi-Newton method (i.e., BFGS) is used for local searching. Vina names this combined global and local searching method as ILSGO. Because of the refined score function and the ILSGO searching method, Vina significantly outperforms AD4 in both docking speed (up to 62× in a single thread scenario) and docking accuracy [3]. However, compared to the LGA in AD4, ILSGO is an irregular and iterative process. Instead of local searching the intramolecular and intermolecular energy of the subset (because each entity, i.e., a Con of the ligand, in the subset is independent of each other, it is possible to perform local searches of these entities in parallel) of a genetic population, an ILSGO iteration only starts from a ligand Con mutation of the last LS result. This means that it is almost impossible to execute ILSGO (including GS, LS, and AG levels in Fig. 2) in parallel. Fortunately, after careful analysis, we found that there is still fine-grained, or low-level, parallelism existing inside BFGS and AG modules. By exploiting this LLP and an optimized memory accessing layout, we successfully realized an FPGA-accelerated Vina in this article.

III. Proposed Architecture

A. Fixed-Point Quantization

To seek a suitable bit width with an acceptable range of accuracies, the core process (Algorithms 1–3) of the native Vina algorithm is modeled by MATLAB for the evaluations of fixed-point quantization on the platform with Intel i7-12 700 K at 3.61 GHz. To demonstrate the validity of the fixed-point quantization, we select two small ligand molecules (“1uwc” with 15 atoms and “1xm6” with 20 atoms) and two medium ligand molecules (“1a30” with 31 atoms and “1r1h” with 33 atoms) from a mainstream docking dataset [22] for the accuracy assessment. To obtain the mean error, we perform

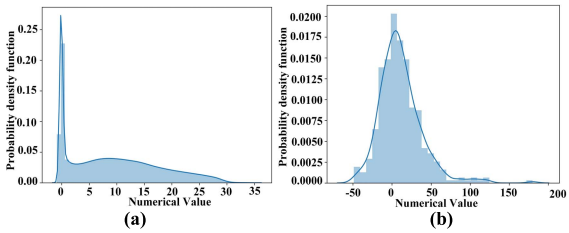


Fig. 3. (a) Probability density functions of the Vina input data and grid values. (b) Precalculated data stored in BRAM and probability density function of the grid values.

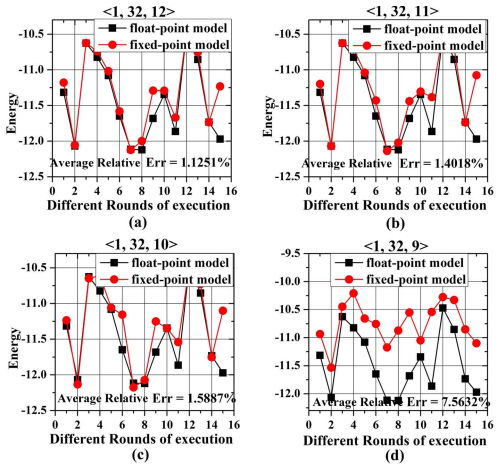


Fig. 4. Error evaluation under different bit widths, in which ⟨1, 32, 10⟩ means that the total width is 32 with 1 bit for sign and 10 bits for the fraction. (a) Error evaluation with fixed-point of ⟨1, 32, 12⟩. (b) Error evaluation with fixed-point of ⟨1, 32, 11⟩. (c) Error evaluation with fixed-point of ⟨1, 32, 10⟩. (d) Error evaluation with fixed-point of ⟨1, 32, 9⟩.

ten times of full Vina Algorithm on each of these four ligands. Due to the interpretation nature, even for the simplest 1uwc, it takes MATLAB about 40 h to run the ten experiments. Thus, it is impractical and unnecessary to run more experiments to evaluate the quantization accuracy. Moreover, the final experimental results shown in Figs. 16 and 17 demonstrate the reliability of our quantization choice.

Note that this article uses $\langle A, B, C\rangle$ to represent the sign bit width, the total bit widths, and the fraction bit widths, respectively. To find the optimal calculation bit widths, this article adopts a stepwise approach, i.e., constantly adjusting the bit width of computing units and memory. As shown in Fig. 3(a), the values of Vina input data are distributed in the range from -50 to 200 and the size of each docking box never exceeds 128. The value distribution indicates that a 7-bit integer width (a range of ± 128) is good enough to represent all the values used in Vina-FPGA. Any data larger than 128 will be saturated to 128. Once the integer bit width determined, we can evaluate the quantization errors with different fractional bit widths as long as we keep the integer bit width larger than 7. Thus, to make the programming of our MATLAB model easier, we keep the total bit width as a constant 32 in our experiment. It can be found from Fig. 4 that the error rises rapidly when the bit width mapping changed from ⟨1, 32, 10⟩ to ⟨1, 32, 9⟩. Note that the energy in Fig. 4 means the free energy after a ligand–receptor docking. Lower docking energy normally indicates a better, i.e., more stable and docking result. Thus, the energy differences between the corresponding (with the same round number in the x-axis)

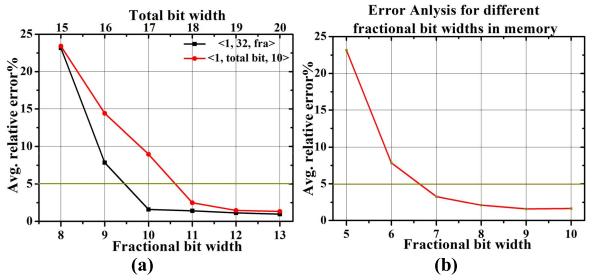


Fig. 5. Relative error evaluation under different bit widths of memory and computational units. (a) Relative error analysis with uniform bit width of memory and computational. (b) Relative error analysis with memory while the computing units are fixed with ⟨1, 18, 10⟩.

docking results, i.e., one result from the fixed-point model (the red dot) and the other from the float-point model (the black dot), are the absolute errors of the fixed-point model. The absolute error or error in short can be denoted by $|\text{Energy}_{\text{fixed}-\text{point}} - \text{Energy}_{\text{float}-\text{point}}|$, while the relative error means the value of $|(\text{Energy}_{\text{fixed}-\text{point}} - \text{Energy}_{\text{float}-\text{point}})/\text{Energy}_{\text{float}-\text{point}}|$. As we can find in Fig. 4(d), bigger energy errors are caused by a 9-bit fraction representation (also the average relative error is higher than 7.5%). Thus, the optimal fractional bit width should be determined as 10 and the total bit width mapping is ⟨1, 18, 10⟩ because we have already determined that the optimal integer width is 7. This choice can also be verified by the results in Fig. 5(a), in which although integer bit widths bigger than 7 obtain lower errors, the profit margins are almost neglectable.

The above quantization process is performed by using a uniform bit width for the calculation unit and memory. However, memory is mainly used for the energy calculation, which means that its precision can be further reduced to save the precious on-FPGA memory capacity. As shown in Fig. 3(b), the absolute values in the energy lookup tables never exceed 64. Thus, we can evaluate the quantization errors with different fractional bit widths while keeping the integer bit width as 6. It can be found that the error rises rapidly when the bit width changes from ⟨1, 14, 7⟩ to ⟨1, 13, 6⟩ in Fig. 5(b). Thus, the optimal bit width mapping for memory should be ⟨1, 14, 7⟩. Therefore, these two fixed-point formats (the computing units with ⟨1, 18, 10⟩ and the memory units with ⟨1, 14, 7⟩) are accepted in Vina-FPGA implementation. What is more, all calculation modules use the unified fixed-point format of ⟨1, 18, 10⟩, including input, output, temporary, and intermediate data. The fixed-point format of ⟨1, 14, 7⟩ is only used to store the constant values of grid energy tables in BRAM. The grid energy tables are precalculated in the initialization stage by the CPU and are downloaded to BRAM through the peripheral component interconnect express (PCIe) interface. Once these tables are downloaded to BRAM, data are only read out (and automatically extended to the ⟨1, 18, 10⟩ format) by other modules.

B. Main Architecture of Vina-FPGA

The architecture of Vina-FPGA with the above described fixed-point format is presented in Fig. 6(a). The host CPU in a PC performs initializing tasks before the docking search, such as files reading, preprocessing, random number generating, and the initial ligand Con generating. Then, CPU sends these data to the block RAM (BRAM) in the FPGA via the PCIe interface. The FPGA carries out the global search iteration (ILSGO

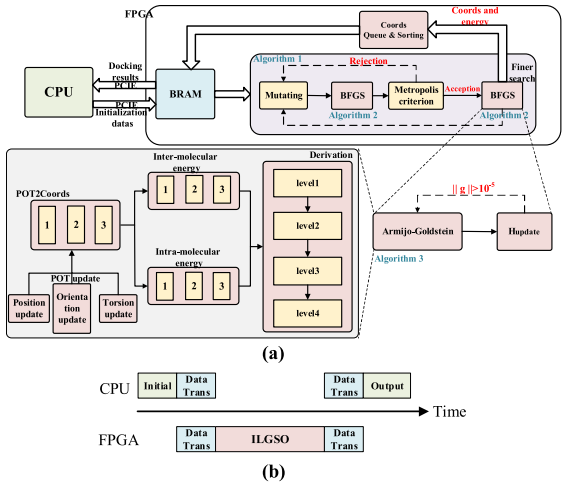


Fig. 6. Architecture and data flow of Vina-FPGA. (a) Architecture of Vina-FPGA. (b) Data flow between CPU and FPGA.

or Algorithm 1), which mainly includes BFGS, Mutating, and Metropolis criterion modules. As described in Algorithm 1, the BFGS module is used to find the local optimal solutions, while the Mutating module generates the initial ligand Con for the next iterative loop of the global search, by adding a random fluctuation to the local optimal solution found in the previous global search loop. As we can find in Algorithm 2 (BFGS) and Algorithm 3 (AG), these two algorithms share some basic modules, such as POT2Coords, intermolecular and intramolecular energy computing, and derivation. Besides, BFGS also has the Hessein matrix updating module. To boost the hardware throughput and the timing, pipelines are widely adopted in these basic modules in Vina-FPGA, which will be discussed in detail in Sections III-C–III-F. What is more, two energy calculation modules are parallelized to improve performance. Note that there is no data sent back to CPU during the ILSGO iteration. Vina-FPGA only returns the final docking result in the queue of a complete ILSGO to the host CPU via the PCIe interface. As we can see from Fig. 6(b), the host CPU sends all data to FPGA after the initial stage and switches to the IDLE state until one complete ILGSO search is done by Vina-FPGA.

C. Coordinates Generation

The Cons of ligand molecules in Vina are represented by position, orientation, and torsion (POT in short), which record the spatial coordinates (of the molecular root), the Euler angle transformations, and the torsion angles of the rotatable keys of the ligand, respectively, as shown in Fig. 1. To convert POT to atomic coordinates, Vina uses a quaternion to represent the spatial rotation of the root node, which records the spatial rotation of the node by the unit vector and the rotation angle around its axis. Note that a node is a group of atoms that form a rigid body (i.e., the relative positions of these atoms in this node are fixed). A ligand could have multiple nodes, while the relative positions among these nodes could be changed during the docking process. To simplify our discussion, however, we use only one circle to represent a node in Fig. 7. For example, θ_1 and vector $\boldsymbol{v}_1$ in Fig. 7 denote the spatial rotation of the root node "$C1$." Thus, the quaternion of "$C1$" could be represented as $(\cos(\theta/2), \sin(\theta/2)\vec{v})$ [18]. Therefore, the relative position of any two connected nodes, such as "$C1$"

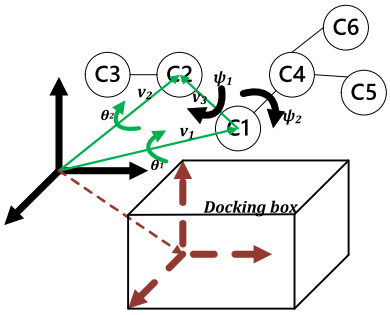


Fig. 7. Quaternions and the docking box. The quaternion of “C1” to the origin point is determined by $\boldsymbol{v}_1$ and θ_1. The docking box is utilized to calculate the intermolecular energy.

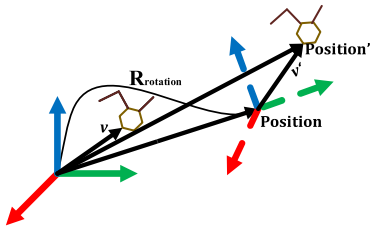


Fig. 8. Rotation matrix and its transformation. Vector of Position' equals (Position + v' = Position + $v \times R_{\text{rotation}}$).

and “$C2$” in Fig. 7, can be obtained during the initialization of Vina. The rotation angle around the axis is the torsion ψ_1. Thus, the quaternion of “$C2$” node can be obtained by quaternion operations [19] according to the quaternion of “$C1$” (determined by $\boldsymbol{v}_1$ and θ_1 in Fig. 7) and the quaternion of “$C1$”–“$C2$” (determined by $\boldsymbol{v}_3$ and ψ_1 in Fig. 7). Similarly, the quaternion of all nodes could be calculated. Furthermore, Vina converts the

$$\text{Rotation} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-xwy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix} \tag{1}$$

quaternion, i.e., (w, x, y, z), to a rotation matrix by (1), where w is $\cos(\theta/2)$ and (x, y, z) represents the vector $\sin(\theta/2)\vec{v}$ [18]. As shown in Fig. 8, a vector ($\boldsymbol{v}$) from the origin point is multiplied by the rotation matrix ($\mathbf{R}_{\text{rotation}}$) to get a new coordinate vector ($\boldsymbol{v}'$) with the same vector size. Then, a new node position (Position′) could be acquired by adding the vector Position to the vector $\boldsymbol{v}''$. Therefore, each rigid body has its own rotation matrix and starting coordinates such as $\mathbf{R}_{\text{rotation}}$ and Position′ in Fig. 8. However, each rigid body is a group of atoms and the relative positions of these atoms in this rigid body are fixed, which is recorded in a data structure named atom_coords. According to $\mathbf{R}_{\text{rotation}}$ and origin of each rigid body and the position between each atom (atom_coords) in a rigid body, the coordinates of each atom could be obtained by (2). Thus, the atomic coordinates in each node could be obtained as long as the rotation matrix ($\mathbf{R}_{\text{rotation}}$) and the position of this node (origin) are acquired. What is more, because different docking boxes are used for different receptors, Vina performs the coordinate transformation to obtain XK for intermolecular energy module

Based on the above discussion, the process of coordinates generation consists of three parts, which include $\mathbf{R}_{\text{rotation}}$ and origin generation of each rigid body, coordinates calculation

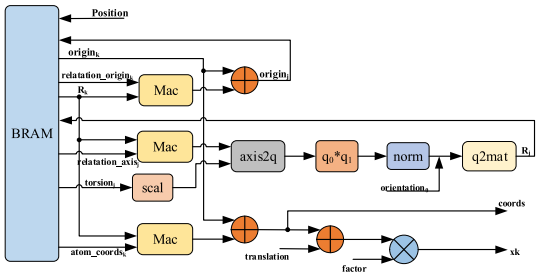


Fig. 9. Pipeline architecture diagram of POT2Coords.

based (2), and XK calculation by coordinate transformation.

$$\text{coord} = \text{origin} + \text{Rotation} * \text{atom\_coords}. \tag{2}$$

Thus, the POT2coords module is divided into a three-stage module-level pipeline, as shown in Fig. 9. The stage of origin and $\mathbf{R}_{\text{rotation}}$ calculation generates the origin and $\mathbf{R}_{\text{rotation}}$ in parallel. The origin is calculated by a multiply–accumulate module and an adder, while a quaternion is acquired by the “axis2q” module according to the formula of $(\cos(\theta/2), \sin(\theta/2)\vec{v})$. Note that θ is scaled within $-\pi$ to π by the “scal” module and $\vec{v}$ is acquired by the coordinate rotation based on $\mathbf{R}_{\text{rotation}}$. Then, quaternion multiplication is designed to calculate the quaternion of the new node [19]. Since quaternion is a normalized vector, a normalization module is employed before the $\mathbf{R}_{\text{rotation}}$ calculation. The “q2mat” module is designed to obtain $\mathbf{R}_{\text{rotation}}$ according to (1). Then, $\mathbf{R}_{\text{rotation}}$ and origin of each node will be stored in BRAM with the size of 1 kB. The stage of coordinate generation is realized according to (2), while the stage of the coordinate transformation is realized by an adder and a multiplier.

D. Energy Calculations

Energy calculation modules perform the calculation of intermolecular energy and intramolecular energy. The intermolecular energy is the algebraic sum of energy between each atom in the ligand molecule and the receptor protein, which depends on the coordinates (XK) of the ligand molecule in the docking box. Similarly, the intramolecular energy is the energy between each atom pair inside the ligand molecule, which is related to the relative coordinates of each atom in *Coords*.

1) Intramolecular Energy: The intramolecular energy is the sum of the energies produced by pairs of atoms that have interacting forces in the ligand molecule. The force only occurs between certain atomic types and the force is ignored once the surface distance between these two atoms is greater than 8 Å (Å is a unit of distance between atoms, which equals 10^{-10} m). Based on the atom type and distance (actually, Vina uses the square of the distance, or r^2, to index the table entry) of different atom pairs, Vina obtains the intramolecular energy from lookup tables. The energy of each atomic pair at different distances is calculated in the preprocessing and stored in a BRAM table, as shown in Fig. 10. There are 153 different pairs of atoms that have interactive forces, whose values are determined by the 2051-level discretized surface distance between these two atoms. To lower the quantization error induced by the distance remainder cut off, a compensation term, rem $\times$ ($m\_\text{data}_{\text{type,int}} - m\_\text{data}_{\text{type,int+1}}$), is added to the table looking up result, shown in (3). Note that

$$f_{\text{intra}_i} = m_{\text{data}_{\text{type,int}}} + \text{rem} * \left(m\_\text{data}_{\text{type,int}} - m\_\text{data}_{\text{type,int+1}}\right) \tag{3}$$

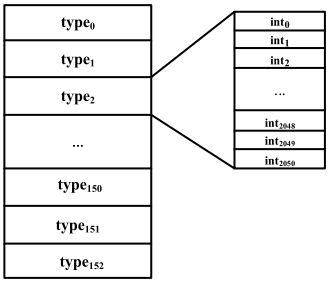


Fig. 10. Organization of the intramolecular energy and derivative lookup table.

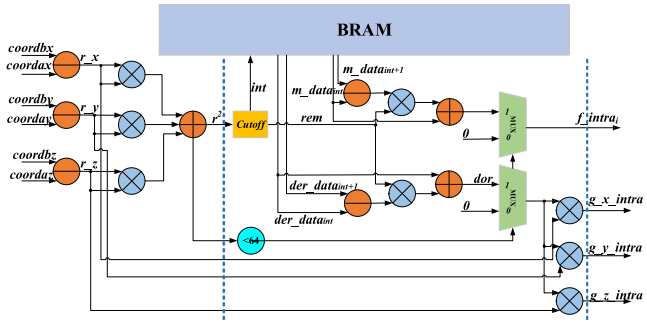


Fig. 11. Hardware implementation of intramolecular energy.

int is the discrete value, while rem is the discrete error of the surface distance, which means that the sum of int and rem is $2051 \times r^2$. Similarly, the energy gradient between each atom can

$$g_{\text{intra}_i} = \left(\text{der}_{\text{data}_{\text{type,int}}} + \text{rem} * \left(\text{der\_data}_{\text{type,int}} - \text{der\_data}_{\text{type,int}+1}\right)\right) * \vec{r} \quad (4)$$

also be obtained by a similar method and the only difference is that the gradient calculation has its own table, which is shown in (4). Since the gradient owns direction, Vina multiplies the gradient by the coordinate vector between the two atoms.

This article implements the hardware of intramolecular energy computing, as shown in Fig. 11. First, the Euclidean distance r^2 between the two atoms (coorda and coordb) is obtained. Then, the discrete error rem can be gained by cutoff, while int can be gotten by discretization. Based on these parameters, the energy and derivative between the two atoms are obtained according to (3) and (4). Meanwhile, we construct a three-staged pipeline to improve the system throughput, which includes the Euclidean distance r^2 calculation, the energy and gradient calculation, and the result output.

2) BRAM Remapping Strategy: The intermolecular energy is the sum of the energies between all ligand atoms and receptor atoms. However, the huge number of atoms in the receptor molecule, normally a protein, makes it impossible to calculate the energy with a reasonable time overhead

To lower the computation complexity, the intermolecular energy calculation in Vina is implemented by trilinear interpolations based on the precalculated grid data from the host CPU [20]. Note that Vina needs different energy tables for different types of atoms, for example, a table for the carbon atoms (C) and another table for the nitrogen atoms (N). As shown in Fig. 12(a), Vina divides the searching space into multiple adjacent cubes or grids, in which the intermolecular energy between an imagined atom on each vertex point

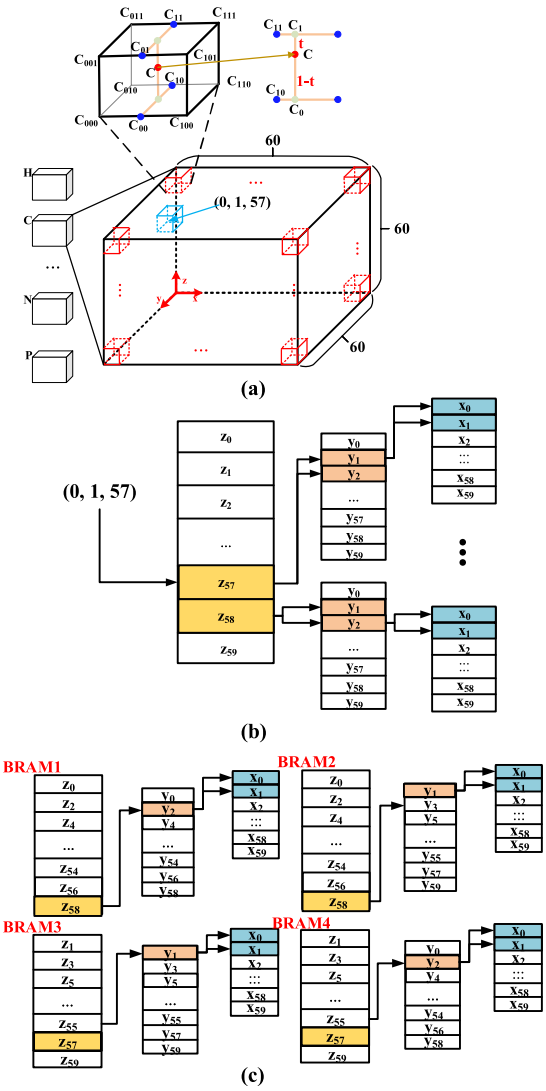


Fig. 12. Trilinear interpolation and data remapping strategy. (a) Diagram of trilinear interpolations for intermolecular energy calculation. (b) Intermolecular energy data arrangement of origin Vina. (c) Intermolecular energy data arrangement of Vina-FPGA. Each BRAM owns independent parity index of y and z. For example, the data in which index of z is even and index of y is even will be mapped into BRAM1.

(i.e., C_{000}–C_{111}) and the receptor is precalculated and stored in an energy table corresponding to the atom type.

Therefore, for a given atom in the ligand, point C in Fig. 12(a), the energy of point C ($E(C)$) could be computed according to $E(C) = E(C1) + t \times [E(C0) - E(C1)]$ by a linear interpolation in which t is the normalized distance between points C and C_1. The energy at C_0 and C_1 could be calculated by two other linear interpolations from C_{10} and C_{00} and C_{11} and C_{01}, respectively. Similarly, the energy at C_{10}, C_{00}, C_{11}, and C_{01} could be derived from the precalculated energy of the eight vertex points, C_{000}–C_{111}, which could be obtained from the corresponding energy table with the same atom type to the ligand atom, i.e., a carbon atom in Fig. 12(a).

The values of points C_{000}–C_{111} could be accessed by the address according to (5) if the docking box size is divided into

$$\text{Addr} = \text{TableBase} + x + 60 * (y + 60 * z) \quad (5)$$

60 cubes × 60 cubes × 60 cubes (i.e., grids) and the discrete data are organized by a 3-D index table, which is shown in Fig. 11(b). Note that the table contains a total of 60 × 60 × 60 data, while for each index of z, there are 60 × 60 data for different x and y values, and for each index of y, there are 60 data for different x values. For example, to obtain the vertex values of the grid with coordinate ($x = 0$, $y = 1$, and $z = 57$), shown as the blue cube in Fig. 11(a), eight data stored at (0, 1, 57), (1, 1, 57), (0, 2, 57), (1, 2, 57), (0, 1, 58), (1, 1, 58), (0, 2, 58), and (1, 2, 58) will be accessed, as shown in Fig. 12(b). Due to the irregular changes of ligand molecules and limited by the two read ports of each BRAM block, it takes at least four cycles to obtain these eight vertex values, should all these values be stored in only one BRAM block. To solve this problem, we utilize the characteristics of data arrangement and map the values of the vertex points in the grid to four separate BRAM blocks, noted as $\text{BRAM}_{1,2,3,4}$, which are shown in Fig. 12(c). It is worth mentioning that each BRAM owns independent parity of the index of y and z. For example, BRAM1 will be accessed if the index of z and y is even, while BRAM2 will be accessed if the index of z is even and the index of y is odd. Therefore, the eight vertex data used in the trilinear interpolations will be evenly distributed in the four BRAM1, 2, 3, and 4 blocks to support one-cycle accessing

3) Intermolecular Energy: Based on the above discussion, the trilinear interpolation formula is shown in (6). Note that $f000$–$f111$ represent the grid

$$\begin{aligned} f_{\text{inter}_j} = {} & (1-x_d)(1-y_d)(1-z_d)f_{000} + x_d(1-y_d)(1-z_d)f_{100} \\ & + (1-x_d)(1-y_d)z_d f_{001} + x_d(1-y_d)z_d f_{101} \\ & + (1-x_d)y_d(1-z_d)f_{010} + x_d y_d(1-z_d)f_{110} \\ & + (1-x_d)y_d z_d f_{011} + x_d y_d z_d f_{111} \end{aligned} \tag{6}$$

vertex values, while the coordinate (x_d, y_d, z_d) represents the normalized positions of the target ligand atom located in the grid. As we can find, 24 multipliers and seven adders are required to complete a trilinear interpolation calculation, which leads to longer critical timing paths and degrades the system performance. Thus, we convert (6) to (7) by extracting common factors,

$$\begin{aligned} f_{\text{inter}_j} = {} & [f_{000}(1-x_d) + f_{100}x_d](1-y_d)(1-z_d) \\ & + [f_{001}(1-x_d) + f_{101}x_d](1-y_d)z_d \\ & + [f_{010}(1-x_d) + f_{110}x_d]y_d(1-z_d) \\ & + [f_{011}(1-x_d) + f_{111}x_d]y_d z_d \end{aligned} \tag{7}$$

which needs 16 multipliers and seven adders to calculate f_inter_j without changing the calculation result.

The implementation of intermolecular energy module is shown in Fig. 13. Similarly, we utilize a three-stage pipeline to improve the system throughput, which includes the grid position determination, the energy and gradient calculation, and the result output.

E. Derivation Calculation

The intermolecular and intramolecular energy calculation modules not only output the energy inside the ligand and energy between the ligand and the receptor but also obtain the energy gradient of each ligand atom in directions of 3-D coordinates x, y, and z. The derivation calculation module converts the energy gradient in the direction of x, y, and z back

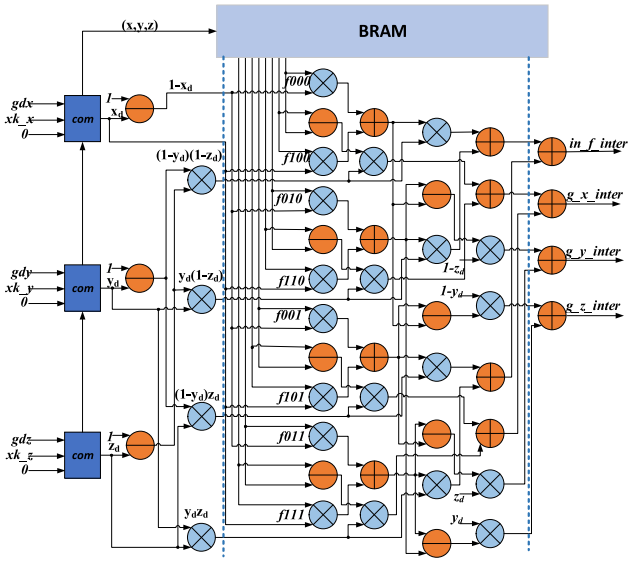


Fig. 13. Hardware implementation of intermolecular energy.

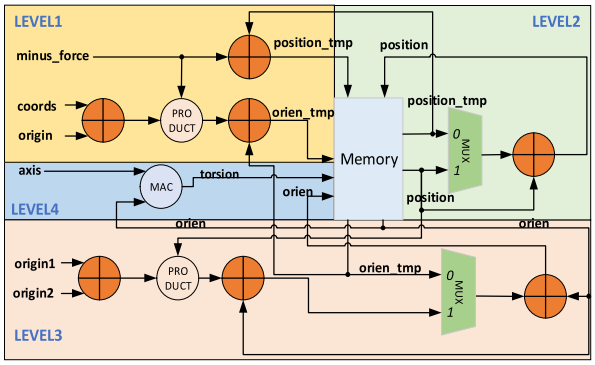


Fig. 14. Hardware implementation of derivation calculation.

to the gradient in POT, which means that it is a reverse process of POT2Coords module and requires some parameters from this module. Unfortunately, the derivation module obtains the gradient through the chain derivation rule, which brings huge challenges to the pipeline implementation. Hence, we divide the derivation calculation module into four levels (derivation calculation of each node based on the derivation of each atom inside the node, POT derivative calculations), which are executed in serial due to the data dependencies. However, we implement pipelines inside each level, which is similar to the Coords generation module and shown in Fig. 14. Due to each node owning multiple atoms as discussed in Section III-C, the derivation of each node will be calculated and stored in the memory (position_tmp and orien_tmp) with the size of 256 B in the first level. What is more, levels of 2–4 are the process of dimensionality reduction, which calculates the derivation of POT based on the derivations of each node in level 1. Unfortunately, the derivation of orientation relies on the derivation of position, while the derivation of torsion depends on the derivation of orientation, which means that we can only calculate derivations of POT in serial. Besides, the input of coords, origin, origin 1, origin 2, and axis is from the POT2Coords module, while minus_force is the gradient of each atom in the ligand, which is from the energy calculation modules. In the first level, a product module is introduced to calculate the cross product of vector with derivation of each

atom (minus_force) and the relative coordinate vector (coords–origin) which requires three-stage pipelines in level 1. Besides, adder and access to memory require two additional pipeline stages, which means that level 1 is a five-stage pipeline. Similarly, levels 2 and 4 include read memory, calculation (adder or multiply–accumulate), and write memory, which requires a three-level pipeline. What is more, level 3 owns one more serial adder than level 1 so that it requires a six-stage pipeline to improve timing.

F. Hessian Matrix Update

The Hessian matrix is used to quantify the local curvature of the objective function, which is the energy between the ligand and the receptor in our work. Thus, the Hessian matrix needs to be updated after each iteration according to (8). Note that k

$$\boldsymbol{h_{k+1}} = \begin{cases} h_k + \dfrac{\left(\frac{y_k^T h_k y_k}{d_k^T y_k} + \alpha\right) * d_k d_k^T - h_k y_k d_k^T - d_k y_k^T h_k}{d_k^T y_k}, \\ \qquad d_k^T y_k \rangle 0 \\ h_k, \\ \qquad d_k^T y_k \le 0 \end{cases} \tag{8}$$

$$\boldsymbol{h_0^*} = \frac{\alpha d_0^T y_0}{y_0^T y_0} * \boldsymbol{h_0} \tag{9}$$

is the BFGS iteration number and the initial value is 0. The N-dimensional (N depends on the dimensions of POT, i.e., position, orientation, and torsion) Hessian matrix is represented as h, while d represents the search direction, which is acquired by $-(h_k * g_k^T)^T$. α is the search step, while y is the gradient change rate, which is acquired by $(g_{k+1} - g_k/(k+1) - k)$ (g is the gradient from the derivation calculation module). Specifically, the initial Hessian matrix ($\mathbf{h_0^*}$) is corrected in Vina [3] by (9)

$$\begin{aligned} \boldsymbol{h_{k+1}} = \boldsymbol{h_k} \quad * & \begin{cases} \dfrac{\alpha d_k^T}{y_k^T y_k}, & k = 0 \\ 1, & k \ne 0 \end{cases} \\ + \boldsymbol{d_k d_k^T} * & \begin{cases} \dfrac{\alpha + \begin{cases} \dfrac{\alpha y_k^T h_k y_k}{y_k^T y_k}, & k = 0 \\ \dfrac{y_k^T h_k y_k}{d_k^T y_k}, & k \rangle 0 \end{cases}}{d_k^T y_k}, & d_k^T y_k \rangle 0 \\ 0, & d_k^T y_k \le 0 \end{cases} \\ - \left(\boldsymbol{h_k y_k d_k^T} + \boldsymbol{d_k y_k^T h_k}\right) \quad * & \begin{cases} \dfrac{\alpha}{y_k^T y_k}, & d_k^T y_k \rangle 0 \\ \dfrac{1}{d_k^T y_k}, & d_k^T y_k \le 0 \end{cases} \end{aligned} \tag{10}$$

and $\mathbf{h}_0$ is the identity matrix, which could quantify the local curvature of the objective function at the initial point to improve the accuracy. Thus, h_{k+1} consists of three matrices $[h_k, d_k^* d_k^T, (h_k^* y_k^* d_k^T + d_k^* y_k^{T*} h_k)]$ and their coefficients, which is shown in (10) based on (8) and (9).

Based on the above formula, we implemented the design of Hessian matrix update and the input consists of vectors (d and y), Hessian matrix (h), search step α, and the first update signal ($k = 0$), which is shown in Fig. 15. Two serial multiply–accumulate modules are designed to obtain the

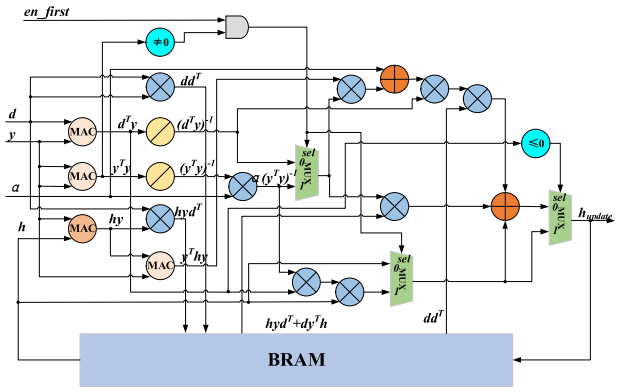


Fig. 15. Hardware implementation of Hessian matrix updating.

denominator of all factors ($d^T y$ and $y^T y$) in (10). Besides, the coefficient of the term $\boldsymbol{d_k d_k^T}$ is very complex. Thus, a parallel multiply–accumulate module is designed to get $h_k y_k$ and a serial multiply–accumulate module is designed to get $y_k^T h_k y_k$. Meanwhile, the matrix of $\boldsymbol{h_k y_k d_k^T}$ will be computed based on the result of $h_k y_k$ and the matrix of $\boldsymbol{d_k d_k^T}$ will be acquired with a multiplier. However, since the symmetry of the Hessian matrix, only $\boldsymbol{h^* y^* d^T}$ will be computed and stored in the local memory to save memory resources [$(\boldsymbol{hyd^T})^{\mathbf{T}}$ equals $\boldsymbol{dy^T h^T}$ and $\boldsymbol{dy^T h^T}$ equals $\boldsymbol{dy^T h}$]. Thus, ($\boldsymbol{hyd^T + dy^T h}$) will be accomplished in the local memory by transpose and addition of matrix according to $(\boldsymbol{h^* y^* d^T})^T$. Ultimately, h_{k+1} will be generated by three Muxs based on the above results.

IV. Experimental and Discussion

A. Resource Utilization

The proposed design is coded in Verilog HDL and synthesized by Vivado on the Ultrascale FPGA (xcku060-ffva1156-2-i) board with the clock frequency of 150 MHz. Kintex Ultrascale FPGA has enough BRAM capacity to store all the grid energy data used for the energy computing, which guarantees that the eight vertex values used for the intermolecular energy computing could be obtained in just one cycle. Also, its PCIe interface can be easily used for the data exchange between the CPU and the accelerator. Moreover, the 20-nm FinFET technology used in Kintex Ultrascale makes the power consumption of the FPGA much lower. Table I shows the resource utilization of each module in the system. The PCIe module is generated by Vivado IP Tool for communications with the host CPU. Due to the lookup tables needed in the calculation, intermolecular and intramolecular energy calculations consume significant capacity of BRAM. The Coords Queue & Sorting module stores and sorts the calculation results from each BFGS and only the final results are sent back to the host CPU when ILGSO is completed. Although a hardware-based sorting mechanism consumes plenty of resources, communications between the FPGA and the host CPU can be significantly reduced by this module. Moreover, this module could be shared by multiple computing lanes for high-level parallelism in the future.

High resource utilization results in huge challenges in routing, not to mention using almost 100% BRAM runs out of wiring resources under the default strategy, which makes it difficult to increase the system frequency. Therefore, we adopt the Vivado's built-in Floorplanning tool, making the energy calculation module close to the BRAM blocks to optimize

TABLE I
RESOURCE UTILIZATION OF VINA-FPGA

	Mutate	Metropolis accept	H_{update}	Conformation update	Coords generation	Inter-molecular energy	Intra-molecular energy	Derivation	PCIe	Coords Queue&Sorting	Other	Total
LUT	6005(1.8%)	4255(1.2%)	20945(6.3%)	11663(3.5%)	8956(2.7%)	3111(0.93%)	1667(0.50%)	14275(4.3%)	17254(5.2%)	127101(38.3%)	44354(13.3%)	259586(78.2%)
FF	4792(0.7%)	472(0.07%)	17593(2.7%)	8276(1.2%)	13000(1.9%)	3174(0.48%)	1800(0.27%)	10912(1.6%)	18532(22.8%)	156642(23.6%)	46129(6.9%)	281322(42.4%)
DSP	36(1.3%)	0	87(3.2%)	68(2.5%)	68(2.5%)	36(1.3%)	14(0.5%)	15(0.5%)	0	63(2.3%)	3(0.1%)	390(14.1%)
BRAM	0	0	68(2.5%)	0	0	760(70.4%)	251(23.2%)	0	45(4.1%)	0	0	1080(100%)

the critical path. In addition, we change the implementation strategy to Congestion_SpreadLogic_high, and it relieves the congestion between the energy calculation and related modules.

B. Validation

The output energy between the ligand and the receptor as well as the root-mean-square deviation (RMSD) between the output ligand Con and the X-ray measurements (the ground truth) are usually used to verify the docking results [21]. We validate Vina-FPGA using a mainstream MD dataset consisting of 140 molecular pairs (ligands and receptors) [22]. Due to the usage of fixed point, the calculation process of the original Vina and Vina-FPGA cannot be guaranteed to be completely consistent even when they use the same random seeds. Similarly, due to the stochastic nature of ILSGO methods, each ILSGO run can have different endpoint results for either software or hardware implementations. To account for this variation, we run Vina and Vina-FPGA ten times and compare the RMSDs and energy distributions [17]. Besides, the parameter of exhaustiveness (the number of initial values of the SA) in ILSGO (line 1 in Algorithm1) has a great influence on the output energy and RMSD, which has a default value of 8 in Vina. As Trott et al. [3] had claimed that users should increase the exhaustiveness value for more stable results, we conduct our experiments with exhaustiveness of 16 and 32. Thus, the total number of global searches in our research is 10 runs $\times$ 32 (or 16, depending on the exhaustiveness selections) $\times$ 140 pairs $=$ 44 800. Compared to the studies in [28] and [29], which only conducted 100 runs $\times$ 60 complexes $=$ 6000 [28] and 100 runs $\times$ 5 complexes $=$ 500 [29] global searches, our investigations are more general and convincing. Fig. 16(a) shows the output energy differences between two rounds of Vina docking with different random seeds under the exhaustiveness of 16. Each red dot in Fig. 16(a) indicates the energy output for a given ligand and receptor complex, while the x and y coordinates of the dot represent the output values from these two-round executions. Ideally, the output energy from these two different rounds of Vina should be the same and located on the diagonal. However, it can be found that although most of the energy outputs are closely located at the diagonal, there are still some points that relatively deviated from the diagonal. This is because of the stochastic nature of Vina algorithm, as we explained in Section II. Fig. 16(b) presents the corresponding variations between Vina and the proposed Vina-FPGA. As we can notice, the error and variance are close between Fig. 16(a) and (b). To further reduce the dispersion, we performed the same

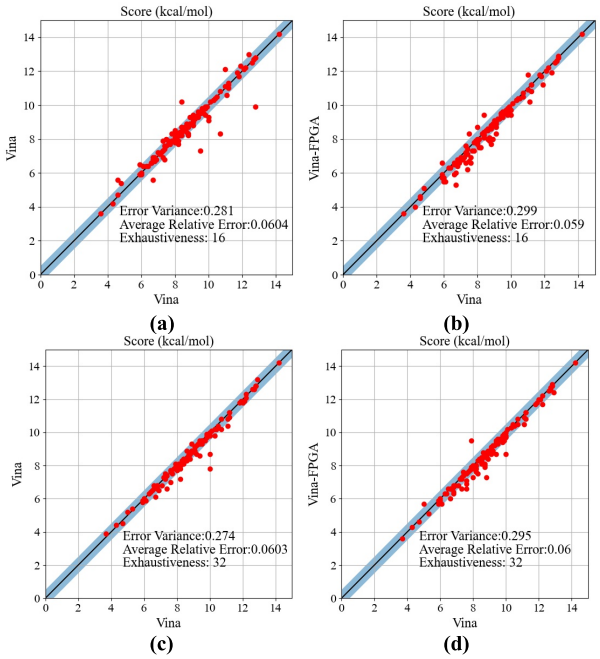


Fig. 16. Output docking energy comparisons of Vina-FPGA and Vina. There are considerable variations even between two rounds of executions of Vina due to the stochastic nature of the algorithm, shown in (a) and (c) with exhaustiveness of 16 and 32. Similar variations exist between Vina-FPGA and Vina with different exhaustiveness. However, the error variances and average relative errors are close to those of different Vina executions, shown in (b) and (d).

experiment with the exhaustiveness of 32. Fig. 16(c) shows the output energy differences between two rounds of Vina docking with different random seeds under the exhaustiveness of 32, while Fig. 16(d) shows the corresponding variations between Vina and proposed Vina-FPGA. Compared to the results from exhaustiveness of 16, the dispersion of Fig. 16(c) and (d) is narrowed and the deviation of most values is within ± 0.5, which is the blue area in Fig. 16.

The RMSD distributions (RMSD describes the similarity between the docking result and the X-ray measurement of the ligand Con) of the final Vina output results of the 1400 ligand–receptor complexes (140 $\times$ 10 rounds) with exhaustiveness of 16 is shown in Fig. 17(a). Similarly, Fig. 17(b) shows the RMSD distributions of the final output from Vina-FPGA under exhaustiveness of 16 whose u and σ have a slight deviation from Fig. 17(a) due to the quantization and stochastics error. What is more, the proportions of RMSD less than 2 Å are 53.6% and 46.7% in all docking results of

TABLE II
PERFORMANCE AND ENERGY COMPARISON OF VINA, VINA-GPU, AND VINA-FPGA

Evaluation metric of performance		CPU	Vina- GPU [30]	Vina-FPGA	Improvement over CPU	Improvement over Vina-GPU
Avg. timing-cost	Initial	1.734s	-	1.903s	-	-
	PCIe	-	-	0.156s	-	-
	ILGSO	180.547s	7.27s	46.343s	3.9×	0.15×
	Total	182.281s	9.20s	48.402s	3.7×	0.19×
Power		Core: 47.34W Board: N/A	Core: 67.2W Board: 203W	Core: 4.70W Board: 12.84W	10.1×	14.3×
Energy Consumption (Core Power * Total Execution Time)		8547.09J	488.54J	217.81J	39. 3×	2.2×

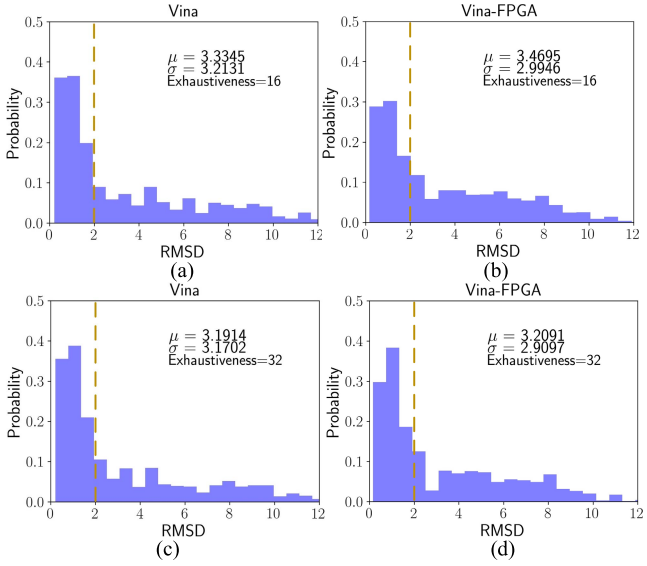


Fig. 17. Comparisons of RMSD distributions from Vina and Vina-FPGA. Due to the stochastic nature of the algorithm, we perform the same experiment ten rounds and there are a slight deviation of u and σ with the exhaustiveness of 16, shown in (a) and (b). To further reduce the stochastic error, we repeated the same experiment when the exhaustiveness is 32. It can be found that u and σ are declined compared to the exhaustiveness of 16 and u and σ are very close between Vina and Vina-FPGA, which is shown in (c) and (d). (a) RMSD with Ex16 of Vina. (b) RMSD with Ex16 of Vina-FPGA. (c) RMSD with Ex32 of Vina. (d) RMSD with Ex32 of Vina-FPGA.

Vina and Vina-FPGA, respectively. Normally, RMSD less than 2 Å is a reliable criterion for docking results. Similarly, we reconstructed the above experiment under the condition of exhaustiveness being 32 to further reduce the stochastic error, which is shown in Fig. 17(c) and (d). It can be found that u and σ have a slight decrease compared with the results with exhaustiveness of 16. Besides, u and σ of Vina and Vina-FPGA are very close when exhaustiveness is 32. The proportions of RMSD less than 2 Å, in this case, are 55.2% and 52.2% in all docking results of Vina and Vina-FPGA, respectively.

C. Performance and Energy Consumption

Performance is evaluated by comparing the total MD computing time of the original Vina running on the Intel i7-12 700 K at 3.61 GHz, Vina-GPU [30] on a NVIDIA RTX 3090 GPU [32], and Vina-FPGA at 150 MHz. The results are shown in Table II. The time of Vina is mainly composed of initialization and ILGSO. Initialization of Vina includes ligand file reading and data preprocessing. The time used by Vina-FPGA consists of initialization, PCIe communication, and ILGSO. The data conversion time of double-precision floating-point data to fixed-point data is counted into the initialization of Vina-FPGA, which is to prepare the necessary data of ILGSO for PCIe communication. Besides, the PCIe communication time of Vina-FPGA is shown in Table II and we only send data to BRAM once for a given ligand–receptor pair. Note that the time is the average docking time per ligand in our test bench. The average CPU time for a ligand docking (total time) is 182.281 s, while that of Vina-FPGA is merely 48.402 s with the exhaustiveness of 32. The average speedup of Vina-FPGA is 3.7× and the maximum speedup is 6.9× when N_{atom} is 31 and N_{torsion} is 4, which is shown in Fig. 18. Although Vina-GPU achieves an average 20× speedup than CPU and the average GPU time for a ligand docking (total time) is 9.2 s, it benefits from the reduction of the iterative loops in each global ILSGO search. By dramatically reducing the search depths, or iteration times, of each individual ILSGO loop to only 22, Vina-GPU executes 1000 independent such depth-reduced ILSGO loops (threads) simultaneously. In this case, the total number of ligand Cons being globally searched is only 22 × 1000. According to the observation from [30], Vina-GPU can achieve a similar docking accuracy while significantly reducing the docking time due to the shortened search depth. However, limited by the hardware resources, it is impossible for the CPU version or Vina-FPGA to execute such

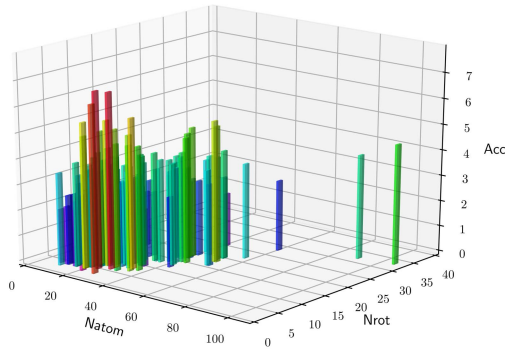


Fig. 18. Acceleration ratio of Vina-FPGA for different ligands.

huge amount individual ILSGO searching threads in parallel. In contrast, either the CPU version or Vina-FPGA needs to execute each ILSGO iteration for 22 365 times to guarantee the searching accuracy, under limited ILSGO iterations (or exhaustiveness, 32 in our case). Obviously, the concurrently executed depth-reduced ILSGO loops in Vina-GPU could be much faster than the execution of a much deeper ILSGO loop.

The power (static and dynamic power of the FPGA chip) of Vina-FPGA, evaluated by Xilinx XPower Analyzer [34], is reduced by 90% compared to CPU, which is measured by Intel Power Gadget [23]. What is more, we also measured the power consumption of the entire FPGA board with an electricity meter (model number DL333501C [31]), which shows that the overall power of the FPGA board is 12.84 W when Vina-FPGA is running. To have a fair comparison, we only list the total energy used to complete ILGSO (core power $\times$ ILSGO time), which is also the most time-consuming part, in CPU, Vina-GPU [30], and Vina-FPGA. The total energy consumption of Vina-FPGA is reduced by 39$\times$ compared to the CPU version. To compare the performance and energy consumption with Vina-GPU [30], we measured the average docking time and power used by Vina-GPU on a NVIDIA RTX 3090 GPU [32]. The power of the GPU Core and the board (including the core) are measured by the tool of GPU-Z [33]. As we can find in Table II, although Vina-FPGA is much slower than Vina-GPU, the energy consumed by Vina-GPU is still 2.2$\times$ of Vina-FPGA. Fortunately, MD software is not a real-time application. Even a look-like trivial speedup of the docking process is still meaningful, considering the huge workload in virtual screening. The lower power consumption of Vina-FPGA could be a potential advantage in a data center scenario with the large scale of drug virtual screening. Moreover, by deploying Vina-FPGA (an enhanced version) on an FPGA cluster that contains multiple FPGA chips, it is highly possible to obtain a similar or even faster accelerating ratio compared to Vina-GPU.

D. Comparison of Vina-FPGA and FPGA-Accelerated AD4

Although it is unfair to compare the accelerators under different computation process, we still compare our work to two FPGA accelerators for AD4 [28], [29], which is shown in Table III.

As we discussed earlier, the key difference between AD4 and Vina is the searching algorithm. AD4 uses the LGA as the global search method and the LS method [29]. Due to the independencies in this process, it is possible for the accelerator developers to exploit this inherent parallelism to speed up the searching, for example, the three-stage pipeline architecture in [28] and the nine independent LS hardware modules in [29].

However, ILSGO used by Vina is an irregular and iterative process. An ILSGO iteration only starts from a ligand Con mutation of the last LS result. This means that it is almost impossible to execute ILSGO (including GS, LS, and AG levels in Fig. 2) in parallel. Thus, Vina-FPGA exploits the LLP inside BFGS and AG modules.

The main innovations of our work on FPGA implementations of AD4 are listed as follows.

1) *Vina-FPGA Further Accelerates the AD4:* To the best of our knowledge, Vina-FPGA is the first reported FPGA accelerator with an effective speedup (average 3.7$\times$ for 140 ligands) for Vina. This acceleration ratio is higher than the average 2.7$\times$ speedup (for only five ligands) of the OpenCL-based FPGA implementation of AD4 [29], which even uses an aggressive parallel architecture. Compared to the average 23$\times$ AD4 acceleration (for 60 ligands) in [28], it seems that the speedup of Vina-FPGA is trivial. However, if we consider the fact that Vina achieves averagely 62$\times$ faster than AD4 with higher accuracy on a single CPU, the absolute docking speed of our work is still almost 10$\times$ faster than [28].
2) *Architecture That Exploits Low-Level Parallelism*: Because of the dependencies among different stages in the ILSGO loop, the architecture of Vina-FPGA must look for parallelism in the lower level. To improve the throughput of the lower level computing modules, different pipeline structures for the modules are proposed. Moreover, because the two energy evaluations (intra and inter) are independent to each other, Vina-FPGA executes these two modules in parallel.
3) *More Efficient Memory Usage*: The AD4 accelerator [28] consumes 40-MB off-FPGA SRAM to store the initialization data, the grid energy tables used for energy evaluations, and docking results. Vina-FPGA only consumes 32-Mb BRAM that benefited from its 14-bit quantization. Also, a new method that remaps the layout of the energy table data is proposed to guarantee one-cycle access to the data used in intermolecular energy computing.
4) *Extension Possibilities*: The works of [28] and [29] allowed data exchange with the SRAMs via simple interfaces. In our design, CPU and FPGA are connected by PCIe, and both of them have their own memory space. This architecture is also suitable for a future cluster design.

V. Related Work

The accelerator of MD is a hot issue in academia and industry, mainly focusing on three levels: CPU, GPU, and FPGA.

CPU: The CPU-based acceleration of MD focuses on improvements to the algorithms. Handoko et al. [11] proposed QuickVina, which reduced the number of iterative searches by adding specific constraints to improve the operation speed. Alhossary et al. [12] presented a scheme to further reduce the number of iterative searches. Hassan et al. [13] proposed a method to eliminate similar molecular Cons to reduce the number of searches. Apart from that, as the original Vina can run on multiple CPUs, VirtualFlow [14] greatly reduced computing time by 160 000 CPUs. However, this resource is almost unaffordable for most researchers, not to mention its huge power consumption.

GPU: Imre et al. [24] optimized AutoDock4 by graphics processing unit (GPU). Solis-Vasquez et al. [25] utilized OpenCL to implement GPU deployment of AutoDock4 on GTX260 and Titan [26]. Solis-Vasquez et al. [27] proposed the ADADELTA algorithm based on the search algorithm of AutoDock4 and deployed it on GPU using OpenCL. Viking is a previous study in Vina GPU acceleration [15]. Unfortunately, due to the uniqueness of Vina in terms of iteration, Viking takes more calculation time on GPU. Tang et al. [30] optimized AutoDock Vina by reducing search depth and increasing the number of initial search points, which achieves significant acceleration. However, the acceleration effect only comes from sufficient hardware resources rather than achieving acceleration under a single computing path.

TABLE III
COMPARING AD4 FPGA IMPLEMENTATIONS WITH OUR WORK

	[28]	[29]	Our work
FPGA	**Virtex-4 LX200**	**Arria 10 GX 1150**	**Ultrascale XCKU060**
Docking Tool	**AutoDock 4**	**AutoDock 4**	**AutoDock Vina**
Clock (MHz)	**100**	**186**	**150**
Precision	**Fixed-point**	**32bit fixed-point for Local Search Floating-point for Energy computing**	**18bit fixed-point (14bit for BRAM)**
Logic[a]	**130K (65%)**	**222K (52%)**	**259K (78.2%)**
Number of Tested Ligands	**60**	**5**	**140**
Speedup to Single Core CPU	**23.3×**	**2.18×**	**3.7×**

a: Xilinx FPGAs in LUTs and Intel FPGAs in ALMs

FPGA: Imre et al. [28] implemented a hardware accelerator for AutoDock4 based on a field-programmable gate array (FPGA) through pipeline design. Solis-Vasquez et al. [29] employed OpenCL to implement the FPGA deployment of AutoDock4.

Due to the parallel nature of the AutoDock4 algorithm, the acceleration for the AutoDock series of MD software is mainly concentrated on AutoDock4. However, because of the irregularity and long iteration of the algorithm, the acceleration research on Vina is still mainly at the algorithm level. This article implements the hardware accelerator of Vina for the first time, and the proposed architecture of our system performs up to 6.9× faster than a state-of-the-art CPU does and has an average 37× higher energy efficiency with the same MD benchmarks.

VI. CONCLUSION

This article presents the first FPGA-based AutoDock Vina implementation called Vina-FPGA. It is the first to accelerate Vina by FPGA with fixed-point quantization. To accomplish this task, we first analyzed the parallel levels of the Vina algorithm, i.e., LLP, MLP, and HLP. Although Vina supports HLP by the Boost Library, MLP is hard to be exploited due to the irregularity and iterations in the algorithm. Thus, we realize a hardware-accelerated AutoDock Vina or Vina-FPGA, by utilizing the LLP of Vina in this article. Various module-level pipelines are adopted in Vina-FPGA to improve the system throughput. To improve memory access speed, a novel data mapping strategy is proposed to achieve equivalent BRAM multiport access. The implementation of Vina-FPGA achieves up to 6.9× (average 3.7×) speedup and similar docking accuracies compared to a single-core implementation of a state-of-the-art CPU. The average energy consumed by Vina-FPGA for a ligand–receptor docking is only 2.5% of the CPU implementation and 45% of Vina-GPU. In the further work, we would like to extend our work into a cluster design, in which multiple lanes of Vina-FPGA will be deployed onto multiple FPGA boards that are connected by a high-speed Ethernet switch.

REFERENCES

[1] M. Mirza and N. Ikram, "Integrated computational approach for virtual hit identification against Ebola viral proteins VP35 and VP40," *Int. J. Mol. Sci.*, vol. 17, no. 11, p. 1748, Oct. 2016.

[2] V. Salmaso and S. Moro, "Bridging molecular docking to molecular dynamics in exploring ligand-protein recognition process: An overview," *Frontiers Pharmacol.*, vol. 9, p. 923, Aug. 2018.

[3] O. Trott and A. J. Olson, "AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *J. Comput. Chem.* vol. 31, no. 2, pp. 455–461, 2010.

[4] T. Gaillard, "Evaluation of AutoDock and AutoDock Vina on the CASF-2013 benchmark," *J. Chem. Inf. Model.*, vol. 58, no. 8, pp. 1697–1706, Aug. 2018.

[5] G. M. Morris, "AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility," *J. Comput. Chem.*, vol. 30, no. 16, p. 2009, pp. 2785–2791.

[6] S.-Y. Lee and K. G. Lee, "Synchronous and asynchronous parallel simulated annealing with multiple Markov chains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 10, pp. 993–1008, 1996.

[7] D. S. Goodsell, G. M. Morris, and A. J. Olson, "Automated docking of flexible ligands: Applications of AutoDock," *J. Mol. Recognit.*, vol. 9, no. 1, pp. 1–5, Jan. 1996.

[8] *Boost C++ Libraries*. Accessed: Oct. 10, 2021. [Online]. Available: https://www.boost.org

[9] V. Y. Tanchuk, V. O. Tanin, A. I. Vovk, and G. Poda, "A new, improved hybrid scoring function for molecular docking and scoring based on AutoDock and AutoDock Vina," *Chem. Biol. Drug Des.*, vol. 87, no. 4, pp. 618–625, Apr. 2016.

[10] R. Quiroga and M. A. Villarreal, "Vinardo: A scoring function based on AutoDock Vina improves scoring, docking, and virtual screening," *PLoS ONE*, vol. 11, no. 5, May 2016, Art. no. e0155183.

[11] S. D. Handoko, X. Ouyang, C. T. T. Su, C. K. Kwoh, and Y. S. Ong, "QuickVina: Accelerating AutoDock Vina using gradient-based heuristics for global optimization," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 9, no. 5, pp. 1266–1272, Sep. 2012.

[12] A. Alhossary, S. D. Handoko, Y. Mu, and C.-K. Kwoh, "Fast, accurate, and reliable molecular docking with QuickVina 2," *Bioinformatics*, vol. 31, no. 13, pp. 2214–2216, Jul. 2015.

[13] N. M. Hassan, A. A. Alhossary, Y. Mu, and C.-K. Kwoh, "Protein-ligand blind docking using QuickVina-W with inter-process spatio-temporal integration," *Sci. Rep.*, vol. 7, no. 1, pp. 1–13, Dec. 2017.

[14] C. Gorgulla, "An open-source drug discovery platform enables ultra-large virtual screens," *Nature*, vol. 580, no. 7805, pp. 663–668, 2020.

[15] J. H. Shin, J. Shin, J. Chae, and Y. S. Jeong, "GPU-accelerated AutoDock Vina: Viking," presented at the ACS Spring Nat. Meeting Expo, Philadelphia, PA, USA, Mar. 2020.

[16] S. Forli, "Computational protein-ligand docking and virtual drug screening with the AutoDock suite," *Nature Protocols*, vol. 11, no. 5, pp. 905–919, 2016.

[17] N. A. Murugan, A. Podobas, D. Gadioli, E. Vitali, G. Palermo, and S. Markidis, "A review on parallel virtual screening softwares for high-performance computers," *Pharmaceuticals*, vol. 15, no. 1, p. 63, Jan. 2022.

[18] C. Hu, M. Meng, M. Mandal, and P. Liu, "Robot rotation decomposition using quaternions," in *Proc. Int. Conf. Mechatronics Autom.*, Jun. 2006, pp. 1158–1163.

[19] J. Baek, H. Jeon, G. Kim, and S. Han, "Visualizing quaternion multiplication," *IEEE Access*, vol. 5, pp. 8948–8955, 2017.

[20] M. M. Jaghoori, B. Bleijlevens, and S. D. Olabarriaga, "1001 ways to run AutoDock Vina for virtual screening," *J. Comput.-Aided Mol. Des.*, vol. 30, no. 3, pp. 237–249, Mar. 2016.

[21] Z. Bikadi and E. Hazai, “Application of the PM6 semi-empirical method to modeling proteins enhances docking accuracy of AutoDock,” *J. Cheminform.*, vol. 1, no. 1, pp. 1–16, Dec. 2009.
[22] 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,” *J. Chem. Theory Comput.*, vol. 17, no. 2, pp. 1060–1073, Feb. 2021.
[23] J. Olivas, T. Kleimola, and M. Price. *Intel® Power Gadget*. Intel. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/tool/powergadget.html
[24] I. Pechan and B. Feher, “Molecular docking on FPGA and GPU platforms,” in *Proc. 21st Int. Conf. Field Program. Log. Appl.*, Sep. 2011, pp. 474–477.
[25] L. Solis-Vasquez and A. Koch, “A performance and energy evaluation of OpenCL-accelerated molecular docking,” in *Proc. 5th Int. Workshop OpenCL*, May 2017, pp. 1–11.
[26] L. Solis-Vasquez, D. Santos-Martins, A. F. Tillack, A. Koch, J. Eberhardt, and S. Forli, “Parallelizing irregular computations for molecular docking,” in *Proc. IEEE/ACM 10th Workshop Irregular Appl., Archit. Algorithms (IA3)*, Nov. 2020, pp. 12–21.
[27] L. Solis-Vasquez, D. Santos-Martins, A. Koch, and S. Forli, “Evaluating the energy efficiency of OpenCL-accelerated AutoDock molecular docking,” in *Proc. 28th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2020, pp. 162–166.
[28] I. Pechan, B. Fehér, and A. Bérces, “FPGA-based acceleration of the AutoDock molecular docking software,” in *Proc. 6th Conf. Ph.D. Res. Microelectron. Electron.*, Jul. 2010, pp. 1–4.
[29] L. Solis-Vasquez and A. Koch, “A case study in using OpenCL on FPGAs: Creating an open-source accelerator of the AutoDock molecular docking software,” in *Proc. 5th Int. Workshop FPGAs Softw. Programmers*, Aug. 2018, pp. 1–143.
[30] S. Tang, “Accelerating AutoDock Vina with GPUs,” *Molecules*, vol. 27, no. 9, p. 3041 May 2022.
[31] Deli. *DL333501C*. Accessed: Mar. 16, 2021. [Online]. Available: http://www.deli-tools.com/productinfo/index/3867.html
[32] Nvidia. *RTX 3090*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.nvidia.cn/geforce/graphics-cards/30-series/rtx-3090-3090ti
[33] TechPowerup. *GPU Z*. Accessed: Aug. 28, 2021. [Online]. Available: https://www.techpowerup.com/gpuz/
[34] Xilinx. *Xilinx Power Estimator (XPE)*. Accessed: May 7, 2021. [Online]. Available: https://www.xilinx.com/products/technology.po-wer/xpe.html

Ming Ling (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1994, 2001, and 2011, respectively.

He is currently an Associate Professor with the National ASIC System Engineering Technology Research Center, Southeast University. His current research interests include memory subsystem of system-on-chip (SoC), processor architecture, and domain-specific architecture.

Qingde Lin was born in 1997. He received the B.S. degree from Zhejiang Gongshang University, Hangzhou, China, in 2020. He is currently working toward the M.S. degree at the School of Microelectronics, Southeast University, Nanjing, China.

His current research interests mainly include the architecture of CPU and the parallel computing.

Ruiqi Chen (Member, IEEE) received the B.S. degree from Southeast University Chengxian College, Nanjing, China, in 2017, and the M.S. degree in integrated circuit engineering from Fuzhou University, Fuzhou, China, in 2020.

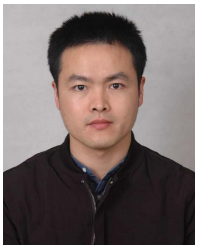
He is currently a Research Assistant with Fudan University, Shanghai, China. His current research interests include domain-specific architecture and field-programmable gate array (FPGA)-based accelerators.

Haimeng Qi was born in 1997. He received the B.S. degree from the Qingdao University of Science and Technology, Qingdao, China, in 2020. He is currently working toward the M.S. degree at the School of Microelectronics, Southeast University, Nanjing, China.

His current research interests include the specific accelerator design.

Mengru Lin (Student Member, IEEE) received the B.S. degree from Jimei University, Xiamen, China, in 2019, and the M.S. degree in integrated circuit engineering from Fuzhou University, Fuzhou, China, in 2022.

She is an Engineer with Verimake, Nanjing Renmian Integrated Circuit Company Ltd. Her current research interests include very large-scale integration (VLSI) and field-programmable gate array (FPGA)-based accelerators.



Yanxiang Zhu (Member, IEEE) received the B.S. and M.S. degrees from Southeast University, Nanjing, China, in 2004 and 2007, respectively.

He is the founder of VeriMake, Nanjing Renmian Integrated Circuit Company Ltd., Nanjing. His current research interests include human–computer interaction and domain-specific architecture

Jiansheng Wu received the B.S. degree in bioengineering from Nanchang University, Nanchang, China, in 2000, the M.S. degree in ecology from East China Normal University, Shanghai, China, in 2004, and the Ph.D. degree in biomedical engineering from Southeast University, Nanjing, China, in 2009.

He is currently a Full Professor with the School of Geography and Biological Information, Nanjing University of Posts and Telecommunications, Nanjing. His current research interests include artificial intelligence for drug discovery and machine learning.