Vina-FPGA-Cluster: Multi-FPGA Based Molecular Docking Tool with High-Accuracy and Multi-Level Parallelism

Ming Ling, *Member, IEEE,* Zhihao Feng, Ruiqi Chen, *Member, IEEE,* Yi Shao, Shidi Tang, and Yanxiang Zhu, *Member, IEEE*

***Abstract*—AutoDock Vina (Vina) stands out among numerous molecular docking tools due to its precision and comparatively high speed, playing a key role in the drug discovery process. Hardware acceleration of Vina on FPGA platforms offers a high energy-efficiency approach to speed up the docking process. However, previous FPGA-based Vina accelerators exhibit several shortcomings: 1) Simple uniform quantization results in inevitable accuracy drop; 2) Due to Vina's complex computing process, the evaluation and optimization phase for hardware design becomes extended; 3) The iterative computations in Vina constrain the potential for further parallelization. 4) The system's scalability is limited by its unwieldy architecture. To address the above challenges, we propose Vina-FPGA-cluster, a multi-FPGA-based molecular docking tool enabling high-accuracy and multi-level parallel Vina acceleration. Standing upon the shoulders of Vina-FPGA, we first adapt hybrid fixed-point quantization to minimize accuracy loss. We then propose a SystemC-based model, accelerating the hardware accelerator architecture design evaluation. Next, we propose a novel bidirectional AG module for data-level parallelism. Finally, we optimize the system architecture for scalable deployment on multiple Xilinx ZCU104 boards, achieving task-level parallelism. Vina-FPGA-cluster is tested on three representative molecular docking datasets. The experiment results indicate that in the context of RMSD (for successful docking outcomes with metrics below 2Å), Vina-FPGA-cluster shows a mere 0.2% lose. Relative to CPU and Vina-FPGA, Vina-FPGA-cluster achieves 27.33× and 7.26× speedup, respectively. Notably, Vina-FPGA-cluster is able to deliver the 1.38× speedup as GPU implementation (Vina-GPU), with just the 28.99% power consumption.**

***Index Terms*—AutoDock Vina (Vina), hardware accelerator, field-programmable gate array (FPGA) cluster, software/hardware (SW/HW) co-design.**

I. Introduction

MOLECULAR docking (MD) plays a key role in the current drug discovery process for fast screening candidate drug molecules with computer algorithms [1], [2]. AutoDock Vina (Vina) [3] stands out among numerous MD tools due to its precision and comparatively high speed [4], [5]. This can be attributed to significant upgrades over its predecessor, AutoDock4 (AD4) [6], Vina introduces great advancements in its scoring function and search algorithm, contributing to a remarkable accuracy. Furthermore, Vina's computational framework is optimized to harness the parallelism offered by multi-core CPUs, which reduces the computational time cost for MD.

However, Vina still requires significant computational time. This is due to Vina exhibiting irregular behaviors in the form of nested loops with changing upper bounds and differing control flows [7]. Hence, Some researchers focus on the Vina acceleration problem. On the CPU side, QuickVina2 [8] and QuickVina-W [9] accelerate the computation through algorithm optimization. Compared to hardware acceleration, the performance improvement brought by algorithms is still limited. VirtualFlow [10] stands as an open-source platform dedicated to drug design, incorporating Vina for virtual screening. Through the utilization of 160,000 CPUs, it considerably truncates the time required for extensive screening. However, the massive server resources and cost overhead are unaffordable for general molecular docking groups. On the GPU side, Vina-GPU [11] represents the first GPU implementation of Vina, boasting an impressive average speed-up of 21×. Leveraging the GPU's multi-core capabilities, it facilitates parallel computations over various initial states, consequently minimizing iteration counts for individual operations and resulting in this acceleration [12]. Vina-GPU2 [13] is an advanced GPU implementation that incorporates the enhanced Vina algorithms, QuickVina2 [14] and QuickVina-W [9], aiming for futher GPU acceleration in virtual screening applications.

Although achieving gains in Vina's parallel acceleration on mainstream computing platforms such as CPU/GPU, they meanwhile introduce significant energy consumption [15]. The FPGA-based accelerator is considered one of the most promising directions, since FPGAs provide low-power and high-energy efficiency and can be reprogrammed to accelerate different applications [16]. Moreover, FPGAs are potential solutions to accelerate MD process, which has been proven in previous MD tools [17], [18]. Motivated by such advantages, we have proposed Vina-FPGA [19], which is the first FPGA-based hardware acceleration of Vina and achieves average 3.7× speedup. Vina-FPGA leverages the uniform quantization to reduce the computational load and employs parallel processing strategies, such as in-module pipeline designs (intra-module level pipeline parallelism) and parallelized intra and inter-molecule energy computing (inter-module level parallelism), Which effectively balance precision and performance.

Ming Ling, Zhihao Feng, Ruiqi Chen, Yi Shao, and Shidi Tang are with the National ASIC System Engineering Technology Research Center, Southeast University, Nanjing 210096, China (e-mail: trio@seu.edu.cn; feng_zh@seu.edu.cn; ruiqichen@ieee.org; 220236527@seu.edu.cn; shidi@seu.edu.cn). Yanxiang Zhu is with the VeriMake Innovation Laboratory, Nanjing Renmian Integrated Circuit Company Ltd., Nanjing 210088, China (e-mail: yanxiangzhu@verimake.com).

Corresponding authors: Ming Ling and Ruiqi Chen.

However, Vina-FPGA still has room for improvements: (1) The sensitivity of Vina to data bit-width alterations compromises its accuracy when using previous uniform quantization methods; (2) The Vina workflow encompasses diverse complex computations, which hinders rapid evaluation and optimization for hardware design; (3) The multi-level iterative computations in Vina constrain the potential for further parallelism design; (4) Vina-FPGA's extensive on-chip memory resource usage and the resource consumption intensive sorting module limit its scalability.

To address these challenges and achieve higher performance with energy efficiency, we propose the Vina-FPGA-cluster, an FPGA-cluster-based molecular docking tool for Vina. First, we use a hybrid quantization based on the data distribution of Vina's computational modules, effectively minimizing the impact of fixed-point arithmetic on accuracy. Then, we develop a SystemC-based hardware model, which, in conjunction with QEMU, facilitates a software/hardware (SW/HW) co-design approach to expedite the hardware accelerator design process. In hardware design, by analyzing Vina's bottlenecks, we propose a novel architecture that facilitates the parallelism of Vina's innermost Armijo-Goldstein Line Search (AG) algorithm to achieve data-level parallelism. Finally, we deploy a cluster architecture across multiple FPGAs to achieve task-level parallel computation for Vina. Our main contributions are as follows:

- **Hybrid Fixed-point Quantization**: A novel approach to Vina's computations, this method involves mixed quantization of various variables within the computation process. When compared to the original Vina computation (32bit floating-point), there's a mere 0.2% reduction in RMSD success ratio.
- **A SystemC-based SW/HW Co-Design Model**: Designed specifically for the hardware acceleration of the Vina algorithm, this SystemC-based model enables swift theoretical performance analysis and facilitates faster design iterations and deployment processes.
- **A Novel Parallel AG Module Design**: Proposing a new architecture for the iterative Armijo-Goldstein Line Search algorithm, namely Bidirectional-AG, which parallelizes the processes. The design obtains 3.90× performance enhancement compared to Vina-FPGA, while only demanding less than twice the hardware resources.
- **FPGA-cluster Based Vina Acceleration**: By leveraging parallel computation across 4 FPGAs, Vina-FPGA-cluster achieves speedups of 27.33× and 7.26× when compared to CPU and Vina-FPGA, respectively. Notably, Vina-FPGA-cluster is able to deliver the 1.38× performance as GPU, with just the 28.99% power consumption. Moreover, the design of Vina-FPGA-cluster can be expanded to systems comprising over 64 or even more FPGA units. To the best of our knowledge, Vina-FPGA-cluster is the first time to represent the multi-FPGA acceleration specifically tailored for Vina.

The rest of the paper is organized as follows: Section II introduces the background and motivations; Section III describes the design methods of Vina-FPGA-cluster; Section IV describes the architecture of Vina-FPGA-cluster; Section V describes the experiments include implementation details and results; Section VI concludes the paper.

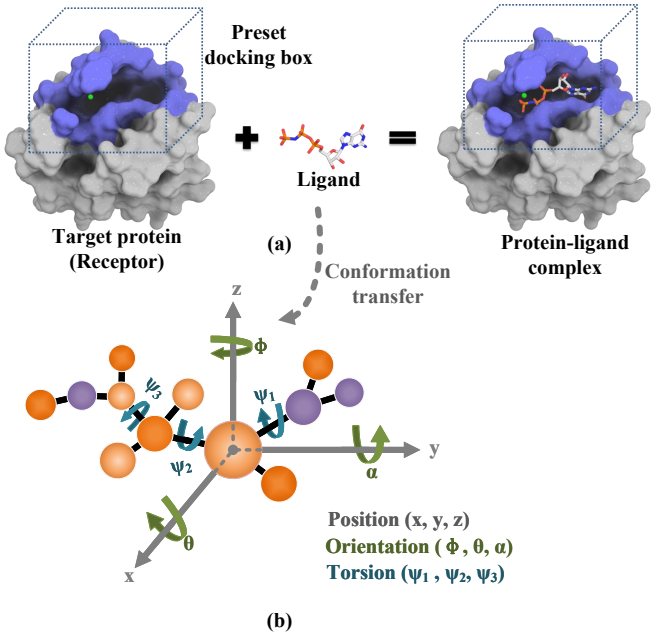


Fig. 1: (a) Schematic illustration of docking a small molecule ligand to a protein target forming a protein-ligand complex [21]. (b) The ligand's conformation variables include Position (Pos), Orientation (Ori), and Torsion (Tor).

II. Background and Motivations

A. Molecular Docking

Molecular docking (MD) is an essential process in scientific drug discovery to predict the binding mode and affinity of a small molecule (ligand) to a target protein [20]. This method can help to identify potential drug candidates by virtually screening large chemical libraries and selecting compounds that are most likely to bind to the target protein with high affinity. As shown in Fig. 1(a), a simple MD process is depicted, where the ligand within a preset docking box area undergoes continuous conformational changes. The variables for the ligand include Position (Pos), Orientation (Ori), and Torsion (Tor), as depicted in Fig. 1(b). During the continuous transformation of the ligand's conformation, its energy in relation to the receptor also changes dynamically. A docking is generally considered successful when the energy reaches a relatively low value. The MD tool uses the energy variations between the receptor and ligand as the output of a scoring function, with Pos, Ori, and Tor as inputs. This represents a typical optimization problem, often involving extensive computational processes.

B. AutoDock Vina

AutoDock [22], [23], [24], [6], [3] is a suite of tools developed by Scripps Research that are available for academic and industrial use at no cost. These tools stand out as some of the only widely-used docking programs to be released

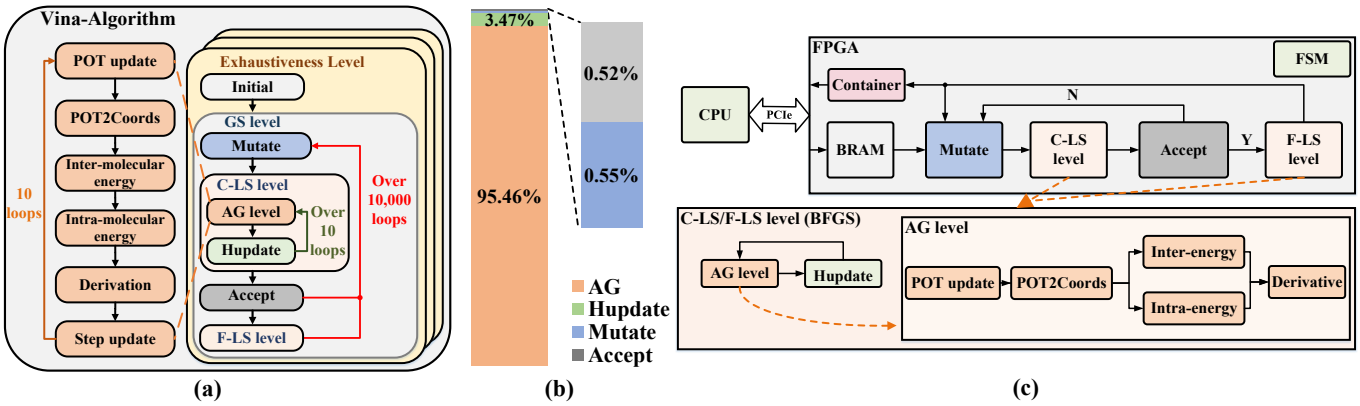


Fig. 2: (a) The algorithm flow of Vina. (b) The profiling of GS. GS, LS and AG stand for Global Search, Local Search and Armijo-Goldstein Line Search, respectively. (c) System architecture of Vina-FPGA [19].

TABLE I: Comparing the methodologies of AD4 and Vina

	AutoDock4 [6]	AutoDock Vina [3]
Scoring Function Parameters	• van der Waals • electrostatic • hydrogen bond • torsional penalty • desolvation	• hydrophobic interaction • hydrogen bond • torsional penalty
Serach Algorithm	• stochastic local search • genetic algorithm	• gradient-based local search • iterated local search • global optimizer

under open-source licenses (GNU General Public License and Apache Open Source License). Among the AutoDock tools, Vina distinguishes itself with superior performance. Compared to its predecessor, AD4, Vina has been demonstrated by its developers to offer higher accuracy in docking and to compute at greater speeds than AD4 [3]. The key differences between the two lie in the parameter settings of the scoring function and the search algorithm, as shown in Table I.

Although Vina takes more time consumption compared to recent commonly used MD tools (such as rDock [25] or LeDock [26]), it still has certain advantages in docking accuracy. This further indicates that accelerating Vina holds significant value. The challenge in accelerating Vina lies in it exhibiting irregular behaviors in the form of nested loops with changing upper bounds and differing control flows.

Vina's core algorithm flow, namely Iterated Local Search Global Optimizer (ILSGO), consists of four nested loops: the Exhaustiveness Level, the Global Search (GS) Level, the Local Search (LS) Level, and the Armijo-Goldstein Line Search (AG) Level, as shown in Fig. 2(a) and Algorithm 1. The Exhaustiveness Level generates multiple independent initial ligand conformations and searches for the global optimum in the search space based on these conformations, where each conformation is represented by multiple dimensions of data: Pos, Ori, and Tor. The GS, LS, and AG levels are interdependent. The algorithm proceeds as follows: At the GS level, initial ligand conformations are randomly perturbed by the Mutate module, altering any dimension of Pos, Ori, or Tor, which then proceeds to the LS level. The computation process at the LS level is a gradient descent search based on the BFGS method [3]. It is divided into coarse-grained LS (C-LS) and fine-grained LS (F-LS), both of which have identical computational procedures, differing only in step length. At the LS level, the linear search algorithm (mainly AG) determines the step length for the conformation search and the corresponding conformation. The Hessian matrix update module (Hupdate) calculates the new direction for ligand conformation search. At the AG-level, it starts with updating the combination coordinates of Pos, Ori, and Tor (POT) in the search direction, converting the updated POT into $Coords$ format data (POT2Coords). $Coords$ is a type of internal coordinate defined by Vina. Converting POT into $Coords$ enables the reduction of originally high-dimensional inputs. Moreover, it simplifies the computation process. Based on $Coords$ format, it completes the energy calculation for the conformation itself (Intra-molecular energy) and the binding energy between the ligand and receptor molecules (Inter-molecular energy). Then, through the derivative module, the derivative value of the energy is calculated. If the updated ligand energy does not get lower (the lower the energy, the more stable the docking conformation [27]), a step length update is performed, causing the conformation to change again. The AG level process will end after executing whole ten iterations, or it stops earlier when a conformation corresponding with lower energy is obtained. The number of iterations of the outer LS level loop is determined by the number of Tor of the ligand (defined as Equation 1):

$$Num_{LS\_loops} = (25 + Num_{Tor})/3, \quad (1)$$

and the number of GS Level loops is determined by the number of atoms in the ligand and the number of Tor (defined as Algorithm 1 line 3):

$$Num_{GS\_loops} = (Num_{atom} + 110 + 10 \cdot Num_{Tor}) \times 105. \quad (2)$$

To gain insight into the computing characteristics of Vina, we analyze the major computational process of Vina including AG, Hupdate, Mutate, Accept, whose execution time ratios are shown in Fig. 2(b). AG constitutes a significant portion of Vina's computational time, with over one million evaluations typically executed. Given its iterative characteristic, it emerges as the primary bottleneck for acceleration. Algorithm 2 depicts the specific process of AG, which comprises five distinct modules. These include the POT_update module (line 5), the

Algorithm 1: Iterated Local Search Global Optimizer

Input: $Con \in [Pos(x,y,z), Ori(\delta,\theta,\Phi), Tor(\psi_T)]$
Input: $N_{\text{atomA}} \in A = \{1,2,\ldots,108\}$
Input: $N_{TorsionT} \in (T = \{1,2,\ldots,32\})$
Output: $Coords_n$ for n=1 to N ($1 \leqslant N \leqslant 20$)
Output: f

1: **for** **exhaustiveness** **do**
/* The **Random** function is initial process. */
2: **Random**(Con)
3: **for** $i \leqslant (N_{atom} + 110 + 10 \cdot N_{Torsion}) \times 105$ **do**
4: $i++$
5: **Mutate**(Con)
/* The **C-LS** and **F-LS** functions are based on the BFGS method. Their difference lies in the step length, with **C-LS** is larger . */
6: $Con, COORD, f_i =$ **C-LS**(Con)
/* The **Accept** function is the Metropolis acceptance criterion. */
7: **if** **Accept**(f_i, f)
8: $Con, COORD, f_i =$ **F-LS**(Con)
9: $Coords_n =$ **Insert**$(COORD)$
10: $f = f_i$
11: **else**
12: **Continue**
13: **end**
14: **end**

Algorithm 2: Armijo-Goldstein (AG) line search Algorithm

Input: $Con \in [Pos(x,y,z), Ori(\delta,\theta,\Phi), Tor(\psi_T)]$
Input: d, f, g
Output: $Con \in [Pos(x,y,z), Ori(\delta,\theta,\Phi), Tor(\psi_T)]$
Output: α, f, g

1: $Con_0 = Con$
2: $f_0 = f$
3: $\alpha = 2^0$
4: **for** $i = 1; i \leqslant 10; i++$ **do**
5: $Con_i = Con_{i-1} + \alpha \times d$
6: Con_i **Convert to** $Coords_i$,XK_i
7: $energy_{inter} =$ **Inter-molecular**(XK_i)
8: $energy_{intra} =$ **Intra-molecular**$(Coords_i)$
9: $f_i = energy_{inter} + energy_{intra}$
10: $g_i =$ **Derivation**$(Coords_i)$
11: **if**$(f_i - f_{i-1}) < 0.0001 \times \alpha \times d \times g_i$
12: **break**
13: **else** $\alpha = \alpha/2$
14: **update** $\{\alpha, f_i, Con_i, g_i\}$ **to** $\{\alpha, f, g, Con\}$
15: **end**

POT2Coords module (line 6), two parallel energy computing modules (lines 7-8), and the Derivation module (line 10). Each iteration of the search necessitates the updating of the conformation Con_i (line 5) based on the last conformation Con_{i-1}, the step size α and the search direction d. The algorithm terminates when the energy value of the latest search outcome is lower than that of the previous iteration at a certain threshold (line 11-12). Otherwise, the next round of iteration is executed (line 5-10).

C. Vina-FPGA

Vina-FPGA stands as the first research to design FPGA-based accelerator for the Vina. To facilitate the implement of the Vina algorithm onto an FPGA, Vina-FPGA commences with a fixed-point quantization of data into two categories: data for storage units and data for computational units. The storage unit data, which comprises static parameters such as the energy values between different atom pairs, is quantized to 14 bits, including a 1-bit sign and a 7-bit fractional part. The computational unit data, representing the receptor and ligand details for the current docking process, is quantized to an 18-bit total width, with a 1-bit sign and a 10-bit fractional part. Subsequently, Vina-FPGA executes the entire computational process within the GS Level, with the system architecture shown in Fig. 2(c). The CPU completes the initial processing of the Exhaustiveness Level, it obtains the computational data, which is transferred to the FPGA via PCIe. The storage unit data is located in the BRAM, where it is combined with the computational unit data sent from the CPU for processing. Data transfer and operation between modules are controlled by Finite State Machines (FSM). The results of each iterative computation are stored and sorted by the Container module. Upon completion of all computations, the data is transferred to the CPU via PCIe to output the final result file. Due to the cyclic and iterative characteristics of the Vina algorithm, the parallel design within Vina-FPGA is deliberately focused. This includes a pipeline architecture within each module and parallel computations of both inter-molecular and intra-molecular energies in the AG level.

D. Motivations

1) Quantization: Quantization is a critical step for algorithm deployment on FPGA, necessitates a trade-off between resource utilization, dictated by data bit-width, and the resultant computational precision [28], [29], [30]. Vina-FPGA employs a uniform fixed-point format, iteratively adjusting the allocation of integer and fractional bit-width based on variations in accuracy to determine the optimal data bit-width. However, while a uniform quantization simplifies hardware design, it affects the accuracy of the final Vina implementation. Consequently, a reevaluation of the quantization strategy is necessary to ensure the competitive accuracy of Vina hardware implementation.

2) Rapid Evaluation & Optimization: In the deployment of algorithms with complex computational processes like Vina, the FPGA development phase typically spends several months [31]. Any modifications by designers to the modules can significantly extend this timeline due to steps such as logic synthesis, placement and routing, and simulation. Therefore, a framework is required for the rapid evaluation and optimization to accelerate the hardware architecture design and implement of Vina, as well as the software development.

3) Parallel AG Computing: AG takes 95% of the total computations in Vina, that means parallel optimization of AG can yield further acceleration. Although AG seems like an iterative process (i.e., the input of the next iteration is the output of the previous iteration), we find that the line search algorithm utilized in Vina is fundamentally a pseudo-iterative algorithm, which can be parallelized.

4) Multi-FPGA Implementation: The overall design of Vina-FPGA is unwieldy, which constrains its scalability. The limitations include: (1) The extensive usage of BRAM for

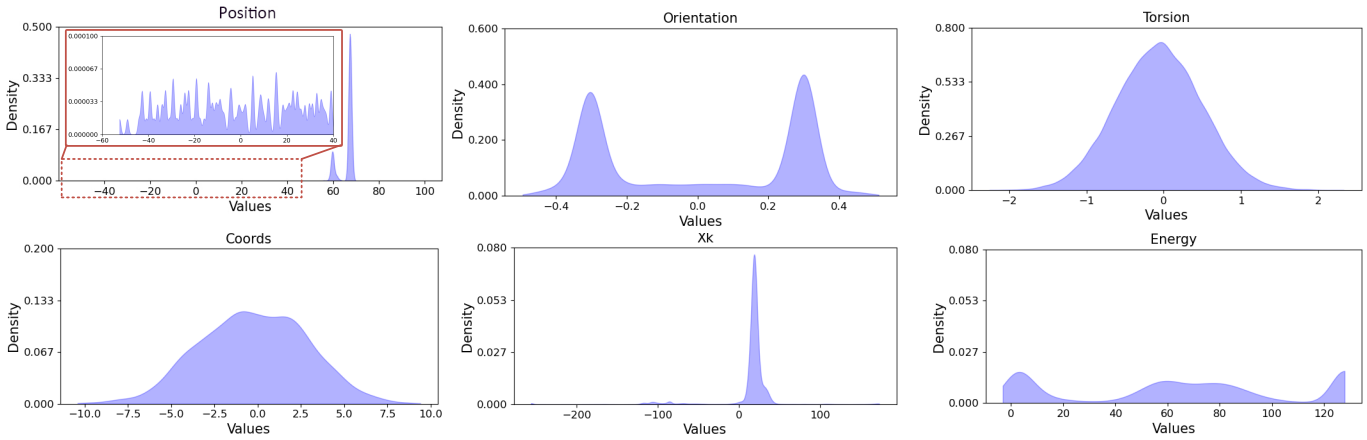


Fig. 3: Probability density functions of Vina computational data.

TABLE II: Fixed-point formats for different computational data

Data Type	Vina-FPGA Fixed-Point Format	Proposed Fixed-Point Formats
Position	<1,18,10>	<1, 18, 10>
Orientation		<1, 18, 17>
Torsion		<1, 18, 15>
Coords		<1, 18, 12>
XK		<1, 18, 9>
energy		<1, 18, 10>

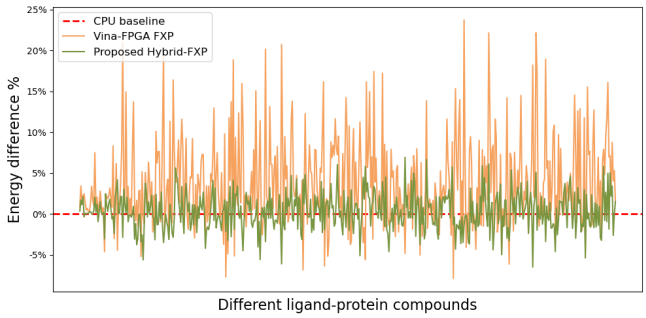


Fig. 4: Comparison of energy differences under different fixed-point quantization strategies. The 'FXP' indicates fixed-point. The 'CPU baseline' indicates the original Vina results from floating-point computations.

parameter storage, although beneficial for memory access latency, leads to routing challenges and a consequent system frequency decrease. (2) The container module, holding and sorting the ultimate 20 docking results, consumes a significant amount of resources.

III. Methods

A. Hybrid Fixed-point Quantization

In the hardware implementation of Vina-FPGA-cluster, we employ a hybrid fixed-point quantization strategy to meet the accuracy and low hardware footprint requirements. This strategy offers more flexibility than the fixed-point quantization strategy in Vina-FPGA, which fixes the number of fractional bits for all computational data. The hybrid fixed-point quantization strategy adapts to the precision needs of different computational processes to reduce the accumulation of errors and ultimately minimize the loss of accuracy. Initially, we perform a statistical characteristics of the numerical value distribution for each data of the computational modules in the Vina algorithm. Then, the distribution helps us determining the maximum and minimum values of each input data and assess the density of data. Finally, we establish the integer bit-width for the fixed-point data representation and adjust the fractional bit-width in response to the data distribution [32].

As shown in Fig. 3, the statistical distributions of numerical values required for computation are presented. It is observed that the value range of most computation data is concentrated around a central point, with the probability density gradually decreasing as the absolute value of the number increases. In Vina-FPGA, the bit-width for all computational data is 18 bits, consisting of 1 bit for the sign, 7 bits for the integer part, and 10 bits for the fractional part, denoted as $\langle 1, 18, 10\rangle$. Similar to Vina-FPGA, the Position data shows the largest numerical range, but this value does not exceed 128, thus we continue to retain a maximum of 7 bits for the integer part. Based on this, we further modify the bit-width of the fractional part according to the displayed probability function, with the results shown in Table II. It should be noted that for 'position' and 'energy', to ensure their effective integer range, we have preserved the data format of $\langle 1, 18, 10\rangle$. The XK is used to record all the coordinates of each atom in the docking box. numerical range lies within ±250 and necessitates a larger allocation of integer bits. Hence, the data format for XK is set to $\langle 1, 18, 9\rangle$. As for other computational data, the fractional bit-width is maximized while ensuring the integer bit-width requirements are met. Energy is a crucial metric in evaluating the accuracy of Vina process. The lower energy indicates higher reliability of docking results. Firstly, we configure a bit-width allocation based on the hybrid fixed-point strategy on the same Matlab platform as Vina-FPGA, identical to the one presented in Table II. Then, we test the energy of the ligand complexes calculated using Vina-FPGA's fixed-point quantization strategy (FXP) and hybrid fixed-point quantization strategy (Hybrid-FXP) under mainstream datasets [33], [34], [35]. The energy difference is denoted by:

$$Energy_{diff} = \frac{(Energy_{FXP} - Energy_{FLP})}{Energy_{FLP}}, \tag{3}$$

where $Energy_{FLP}$ indicates energy results from original Vina with floating-point computations.

Fig. 4 illustrates the precision comparison between fixed-point and hybrid fixed-point quantization strategies, with the original Vina (CPU) serving as the baseline reference. The result evidences that the energy difference significantly decreases with the use of hybrid fixed-point quantization. More results to be discussed in Section V-B1.

B. A SystemC-based Model

The architecture of Vina-FPGA presents some opportunities for optimization. Utilizing a pure FPGA as the target deployment device is not suitable for complex and irregular algorithms like Vina. Heterogeneous platforms, represented by Zynq, hold significant potential for hardware acceleration of such irregular algorithms [36]. However, this approach requires extensive time for logic synthesis to evaluate and validate the performance of the design, thereby hindering the iterative development period of accelerators. The specific details are as follows: (1) In the Vina-FPGA design phase, the Register-transfer Level (RTL) based simulation is highly time-consuming due to the performance constraints of EDA tools [37]; (2) In the heterogeneous development, software design on the processing system (PS) side requires results from hardware design for optimization, but the prolonged hardware design cycle significantly increases the total duration required for system design [38]. Therefore, we propose a SystemC-based SW/HW co-design model. By simulating the hardware architecture design through SystemC and leveraging the Xilinx-Qemu [39] platform to simulate the ARM core, we can comprehensively evaluate and optimize the entire heterogeneous acceleration system.

As shown in Fig. 5, we propose a hardware-software co-design model, encompassing a hardware accelerator architecture based on SystemC and Xilinx-Qemu. These components are coupled through a Remote Port. The Vina hardware accelerator model is encapsulated as a TLM (Transaction-Level Modeling) device and mounted on the AXI bus. Relevant data from the ARM core is transmitted via a SystemC TLM-SoC wrapper and Remote port IPC (Inter-Process Communication). Specifically, (1) The SystemC model excludes interactions with external and memory data like PC, DDR, and BRAM. Data needed for these interactions is stored in a txt file, encapsulated within a globally defined structure. This structure contains data such as inter-molecular energy, intra-molecular energy, and various initial global variables. At the start of the model, the file is processed to load the initialization data into the global data area. (2) We construct all computational components in the Vina algorithm flow. Since Vina's source code is developed in C/C++, we can quickly build computational components and ensure accurate results. Additionally, Vina-FPGA already obtains detailed hardware design specifics, such as the operational clock cycles for each module, from basic mathematical operation units (e.g., the latency for a division operation is 5 cycles) to the module level. Thus, we can use time annotation to replace precise clocks in the RTL model. Time annotation is a mechanism for adding temporal information to components, enabling us to specify or re-write the expected duration for component operations.

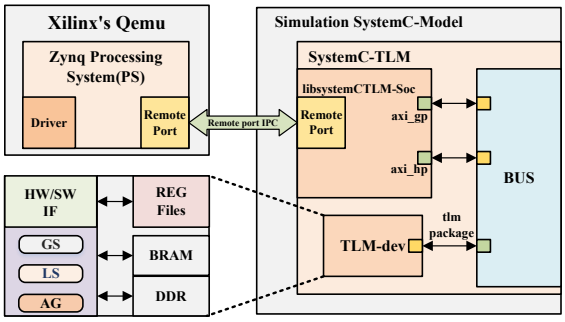


Fig. 5: The SystemC-based SW/HW co-design model.

(3) The computational components are organized into three distinct controllers (GS, LS, and AG controllers) for high-level interface. This structure aligns with Vina's computational flow. (4) We simplify the design of memory components, taking advantage of the fixed cycles of DDR and BRAM accesses. We model memory accesses using assignment operations combined with time annotations that are sourced from AXI bus and memory access cycle consumption identified in the Vina-FPGA design. (5) After rewriting the computation processes of various computational components and setting their operation delays, we divided them into controller models and submodule models. The controller models correspond to the three levels of loops in Vina computations: GS, LS, and AG. These controller models use SC_METHOD processes in SystemC to facilitate data transfer and information synchronization among various functional modules. The submodule models represent specific computational operations involved in each controller model, as shown in Fig. 2a. During modeling, all submodule models employ SC_THREAD and are suspended with $wait()$ at the end of their scheduling to save simulation time. This approach allows us to replicate the loops and the corresponding cycles of computation as they occur in Vina-FPGA runtime. (6) The previous work [40] provides precise cycle accuracy for data transfer on the AXI bus. Moreover, the hardware accelerator receives data from the ARM side via REG Files [39] that are virtual representations within the emulator of the hardware registers. (7) Xilinx-Qemu performs the functional modeling and simulation of the ARM side. To implement data transmission between hardware model and ARM via Remote port IPC, disabling the remote port and the Socket protocol is required.

The proposed SystemC model not only facilitates the establishment of a hardware accelerator model but also ensures compatibility with heterogeneous architectures like Zynq. This means that if there are collaborative tasks on the PS side, joint debugging after completing the PL side is not necessary; development can proceed concurrently with the PL side, shown as Fig. 6(a). For the crucial hardware evaluation and optimization process, we conducted a comparative analysis of the simulation time for the entire model on a typical docking dataset. As shown in Fig. 6(b), due to the extensive simulation time required by the RTL model to complete a full docking, the proposed SystemC model averages a 101-fold speed increase over the RTL model. This demonstrates that the SystemC model can significantly reduce the time required for hardware design evaluation and optimization through simulation results.

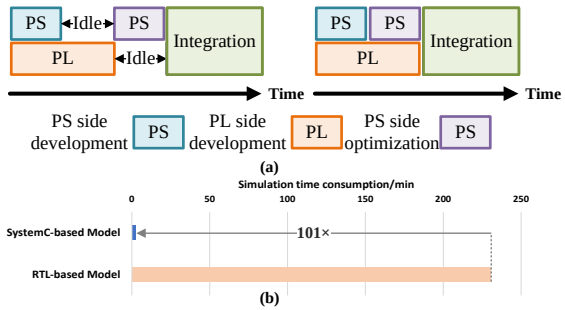


Fig. 6: (a) Comparison of development workflows: Traditional RTL model-based co-design (left) requires PS side optimization to wait for PL side completion. SystemC model-based co-design (right) enables simultaneous development without waiting for PL side. (b) Comparison of simulation time consumption under different models.

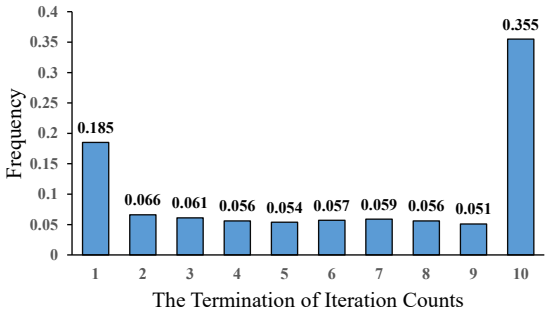


Fig. 7: The distribution of AG's termination iteration counts.

C. Bidirectional-AG

The AG level, serving as the innermost nested loop iteration within Vina, occupies 95% of the execution time of Vina. Therefore, parallelization of the AG level is capable of substantially enhancing the computational efficiency of Vina. We fully analyze the flow of AG line search algorithm (Algorithm 2) in Vina, and the distributions of iteration counts at the termination of each AG search. We can summarize two key observations: (1) The computing input (Con_i) required for each AG iteration is predetermined, allowing for independent pre-computation; (2) Leveraging the prior observation, starting from the iteration points where the highest possibility occurs can minimize the computational load, considering AG's non-uniform distribution of iteration counts.

We first give an equivalent transformation of the update of Con_i (line 5 of Algorithm 2) as:

$$Con_i = Con_0 + 2^{1-i} \times \alpha \times d, \tag{4}$$

where the input Con_i of each iteration depends on i, Con_0, α, and d. Notably, Con_0, α and d are three constant values, and Con_i will not change once it is calculated at the beginning of the iteration. Therefore, all Con_is $(i = 1, 2, \ldots, 10)$ can be pre-computed and the iterations can be performed independently, indicating a potential transfer from the iterative processing to a parallel one.

Leveraging the prior observation, pre-computing all $Con_i s (i = 1, 2, \ldots, 10)$ permits parallel computation of the AG line search algorithm. However, deploying this design in hardware might entail unacceptable resource overheads. We execute Vina on three representative molecular docking datasets [33], [34], [35] and record the iteration counts at the termination of AG searches, as shown in Fig. 7. The distribution of AG's termination iterations is non-uniform and skewed, with over 50% lying at the first (i.e., $i = 1$) and the last (i.e., $i = 10$) iteration. Moreover, other iteration counts average a termination rate of about 5%. Given this distribution, an intuitive approach is to deploy bidirectional search using two sets of AG computational hardware units, operating from both (front and end) sides towards to the center. Moreover, by strategically selecting initial points that are probably close to the termination of iteration counts, we can notably reduce the computational load.

To obtain a better trade-off between the acceleration ratio and resource consumption, we propose the Bidirectional-AG architecture, which incorporates two AG modules that compute simultaneously. The Bidirectional-AG architecture includes the Start-From-Edge (SFE) strategy and Start-From-Middle (SFM) strategy, as illustrated in Fig. 8(a).

The SFE strategy enables two AG modules (AG1 and AG2) to start from the first (i.e., Con_1) and the last (i.e., Con_{10}) initial points, respectively. Both of the AG modules perform their corresponding calculations and move towards to the middle (i.e., Con_5) in parallel. The output of the SFE strategy is based on a corresponding configuration table. The configuration table is shown in Fig. 8(b), where we denote an AG module's result as "Y" if the criterion in line 11 of Algorithm 2 is satisfied, otherwise, we denote "N". The "AG1" and "AG2" in the configuration table output indicate the calculation results of the AG1 and AG2 modules, respectively. The "Continue" denotes both of the two AG modules to continue their iteration. The SFE strategy leverages the non-uniform distribution of the termination of iteration count and reduces computational load, because the two AG modules are highly possible to satisfy the criterion (i.e., terminate) on either front or end sides. The total execution time of the SFE strategy depends mainly on the N_{iter}, i.e.:

$$T_{\text{SFE}} = \begin{cases} T_{\text{iter}} \times N_{\text{iter}}, & \text{if } 1 \leqslant N_{\text{iter}} \leqslant 5 \\ T_{\text{iter}} \times (11 - N_{\text{iter}}), & \text{if } 6 \leqslant N_{\text{iter}} \leqslant 10 \end{cases} \tag{5}$$

where N_{iter} represents the iteration count needed to satisfy the termination criteria in the origianl Vina algorithm.

Therefore, the acceleration ratio achieved by the Bidirectional-AG design of SFE strategy compared to the original AG design can be obtained as:

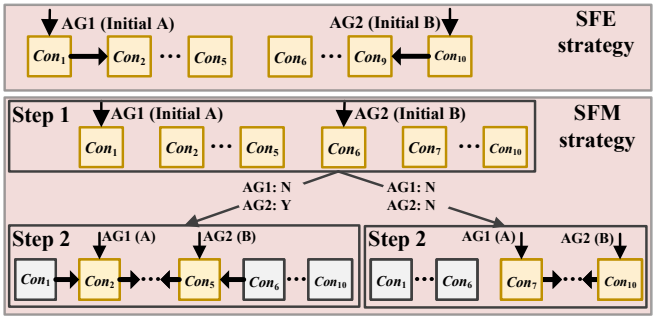
$$\frac{T_{AG}}{T_{SFE}} = \begin{cases} 1, & \text{if } 1 \leqslant N_{iter} \leqslant 5 \\ \frac{N_{iter}}{11 - N_{iter}}, & \text{if } 6 \leqslant N_{iter} \leqslant 10 \end{cases} \tag{6}$$

Hence, the performance of the Bidirectional-AG architecture can only be harnessed when the N_{iter} exceeds 5, and as the N_{iter} increases, the performance improves correspondingly. The average acceleration ACC_{SFE} of the SFE strategy is given by:

$$ACC_{SFE} = \sum_{N_{iter}}^{10} P(N_{iter}) \times \frac{T_{AG}}{T_{SFE}}. \tag{7}$$

Whereas $P(N_{iter})$ denotes the probability associated with different N_{iter}.

The SFM strategy is complementary to the SFE strategy, and it is suitable for cases where most of the terminations



(a)

SFE	Input	AG1	N	N	Y	Y
		AG2	N	Y	N	Y
	Output		AG2	Continue	AG1	AG1

SFM	Input	AG1	N	N	Y	Y
		AG2	N	Y	N	Y
	Output	Step 1	Continue	Continue	AG1	AG1
		Step 2	AG2			

(b)

Fig. 8: (a) The SFE and SFM strategies, where two AG (AG1 and AG2) modules are performed from different initial points. The SFM strategy performs distinctly if the result of Con_6 satisfies the criterion (line 11 of Algorithm 2) (Y) or not (N). (b) The configuration table of SFE and SFM strategies. The "Continue" signifies the ongoing iteration of AG1 and AG2 computations under the corresponding strategy.

of iteration counts are gathered in the middle (e.g., Con_5 and Con_6). This strategy (see Fig. 8) contains two sequential steps. In step one, AG1 starts from the first (i.e., Con_1) initial point and AG2 starts from the middle (i.e., Con_6) point. In step two, the initial points of AG1 and AG2 will be determined by the results of step one, as illustrated in Fig. 8. The outputs of the two steps can be referred to the corresponding configuration table in Fig. 8(b). Notably, the SFM strategy ensures the correctness of the AG algorithm, and it only requires negligible resource overhead compared to that of SFE. Similar to SFE, the acceleration ratio of the SFM can be calculated by

$$ACC_{SFM} = \sum_{N_{iter}}^{10} P(N_{iter}) \times \frac{T_{AG}}{T_{SFM}}, \tag{8}$$

where the $\frac{T_{AG}}{T_{SFM}}$ can be obtained as:

$$\frac{T_{AG}}{T_{SFM}} = \begin{cases} 1, & \text{if1} \leqslant N_{iter} \leqslant 3 \\ \frac{N_{iter}}{7-N_{iter}}, & \text{if4} \leqslant N_{iter} \leqslant 6 \\ \frac{N_{iter}}{N_{iter}-5}, & \text{if7} \leqslant N_{iter} \leqslant 8 \\ \frac{N_{iter}}{12-N_{iter}}, & \text{if9} \leqslant N_{iter} \leqslant 10 \end{cases} \tag{9}$$

We further design a selection module that dynamically selects which one of the two strategies are utilized based on the previous iteration terminations. Details of the selection module are described in Section IV-C2.

The results of the SystemC model show that the Bidirection-AG architecture, including SFE and SFM strategies, obtains a theoretical acceleration ratio up to 5.87× over the original AG architecture in Vina-FPGA, with merely 2× resources overhead.

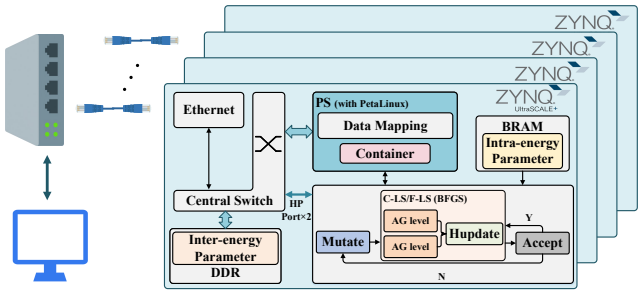


Fig. 9: Overview of Vina-FPGA-cluster.

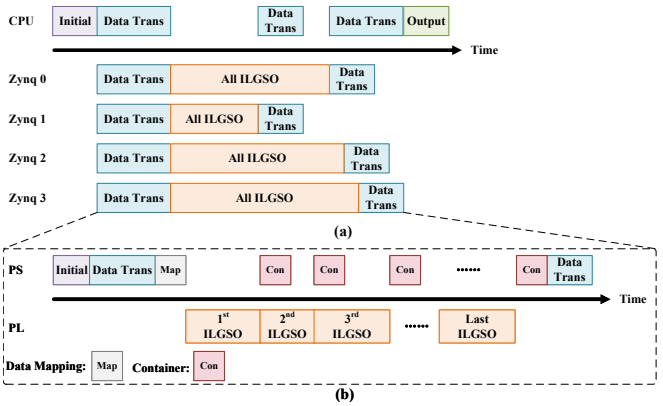


Fig. 10: (a) The scheduling between the CPU and Zynq FPGAs. (b) The scheduling between the PS and PL.

IV. Architecture of Vina-FPGA-cluster

A. Overview of Vina-FPGA-cluster

Fig. 9 depicts the proposed hardware architecture of Vina-FPGA-cluster. There are four Zynq Ultrascale+ FPGAs that perform the ILSGO calculations in parallel. The ligands, receptors, and Exhaustiveness data required for their computations are transmitted from a PC via an Ethernet connection. The PS (Processing System) side runs the PetaLinux operating system, which is in charge of managing the data received via the Ethernet and storing it into the necessary on-chip memory and DDR on the PL (Programmable Logic) side via the AXI bus according to computational needs. Each Zynq's computation can be considered independent, signifying that Vina-FPGA-cluster is capable of parallelizing the Exhaustiveness Level computations. Specifically, the parallel computation of the Exhaustiveness Level involves different Zynq FPGAs calculating various conformations of the same Ligand. The scheduling relationship between the CPU and Zynq FPGAs under these circumstances is illustrated in Fig. 10(a). Since it is the same Ligand, the volume of data to be transferred remains consistent. The variance in the computation latency of ILGSO is attributed to different initial conformations corresponding to different loop nets, which is a random process [12]. The PL side executes the ILGSO computations. There are listed the architecture differences from Vina-FPGA:

Data Communication: The data communication between FPGA and CPU has been changed from PCIe to Ethernet, making our design plug-and-play. This means users can utilize it without altering their current hardware setups, simply by connecting an Ethernet cable. Nevertheless, the choice be-

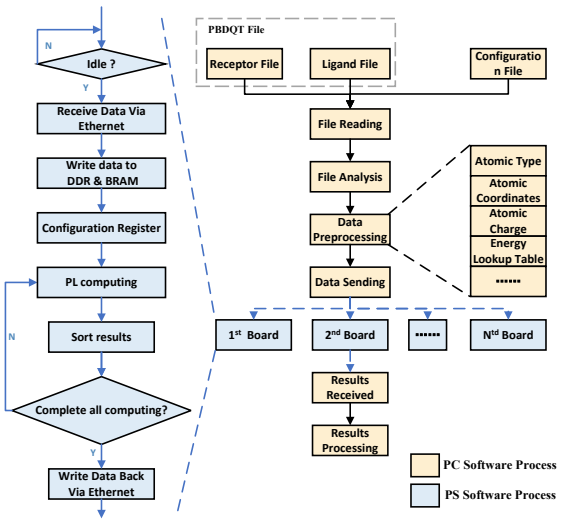


Fig. 11: The flowchart of Vina-FPGA-cluster software.

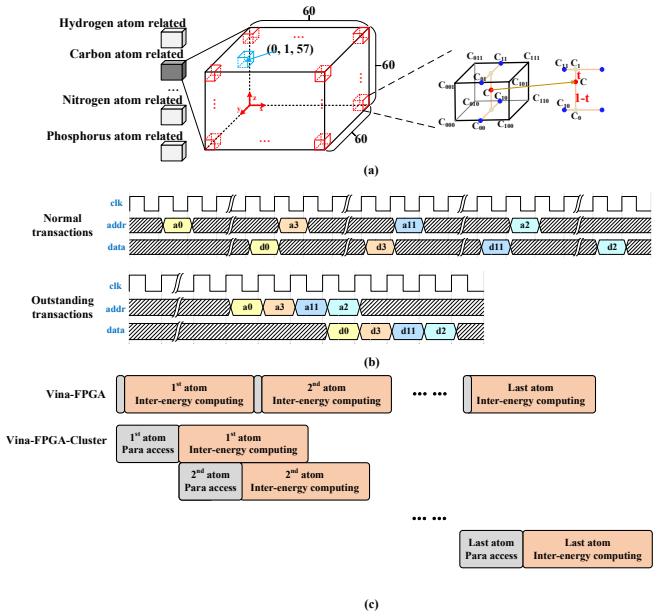


Fig. 12: (a) Diagram of trilinear interpolations for inter-molecular energy calculation. (b) Different transactions are used for continuous access to non-continuous addresses. The double wavy lines represent the time taken to establish AXI protocol handshakes. (c) The pipeline access mode of Vina-FPGA-cluster introduces a minimal increase in parameter accessing latency.

tween PCIe and Ethernet as the communication port in our approach is not conflicting. As long as communication is not the bottleneck of the cluster, as we shown in Section V-D, it is not difficult to switch from one to another. In this paper, we demonstrate and experiment with the design implemented on 4 FPGAs. Moreover, in theory, this cluster can be easily scaled to 64 or even more FPGAs.

Parameter Accessing: The memory accessing of parameters required for computation (Intra-energy, Inter-energy) has been optimized. In Vina-FPGA, all computational parameters were stored in the on-chip memory. This led to nearly 100% consumption of on-chip BRAM memory and caused difficulties in layout and routing, which in turn pulled down the system frequency. In Vina-FPGA-cluster, we have placed the larger and more frequently randomly accessed Inter-energy parameters into DDR with 2 HP-Ports accessing. The inter-molecular energy consists of 17 sheets, each with a size of 281,515 (entries for different atom distances) $\times$ 14 (bits for each grid energy in a fixed-point value), with the 14 bits actually stored in DDR as 32 bits. For intar-energy parameters, they are stored in the on-chip memory (BRAM & URAM). The memory modification details will be introduced in Section IV-C1.

Bidirectional-AG: As introduced in Section III-C, the Bidirectional-AG module parallelizes the originally serial AG iterative computations. This module can achieve more than $2\times$ speed gain under twice the resource consumption of the original AG module. The micro-architecture design specifics of the Bidirectional-AG will be detailed in Section IV-C2.

Heterogeneous Architecture: The introduction of an ARM co-processor (PS) effectively enhances the efficiency of data processing. The PS is responsible for the reception and organization of the initialization data for computation, mapping it to the corresponding memory units (DDR, BRAM). Additionally, the PS undertakes the function of the container module, which is in charge of results sorting in Vina-FPGA. Specifically, the energies corresponding to the ligand conformations post-ILGSO computation are sorted, containing the smallest top 20. However, this imposes a significant on-chip memory burden, hence the introduction of the PS further alleviates the on-chip memory cost pressure on the PL side. The collaboration between the PS and PL sides is depicted in Fig. 10(b).

B. Software Architecture

The software architecture of Vina-FPGA-Cluster primarily comprises two parts, PC and PS, as shown in Fig 9. The PC side mainly handles the reading of files required for molecular docking and the preprocessing of data to acquire what is necessary for the Vina computation module. The PS side is responsible for receiving network data, mapping data to DDR and BRAM for computation by the PL side, sorting the computational results, and transmitting them back to the PC via the network. Specifically, as shown in Fig. 11, the interactive process between PC and PS software is demonstrated. The PC starts by reading parameter files, including receptor, ligand, and configuration files. It then parses the necessary computational parameters, such as atom types, coordinates, and charges.The initialized data sent from the CPU to the FPGAs is shown in the Table VII within the appendix. For an explanation of the above data, please refer to the appendix. The PC quantizes the data and sends it to various Zynq devices via the network. The PS side of Zynq receives the necessary data through the network and writes it correspondingly into DDR and BRAM. PS informs the PL side to start computation by configuring communication registers. Each computation result from the PL side is sent to PS for sorting, which is executed in parallel with the ongoing computation of the PL. Once all Vina computations are complete, PS sends the data back to the PC via the network. The PC side performs the final sorting of all received data and outputs the final results.

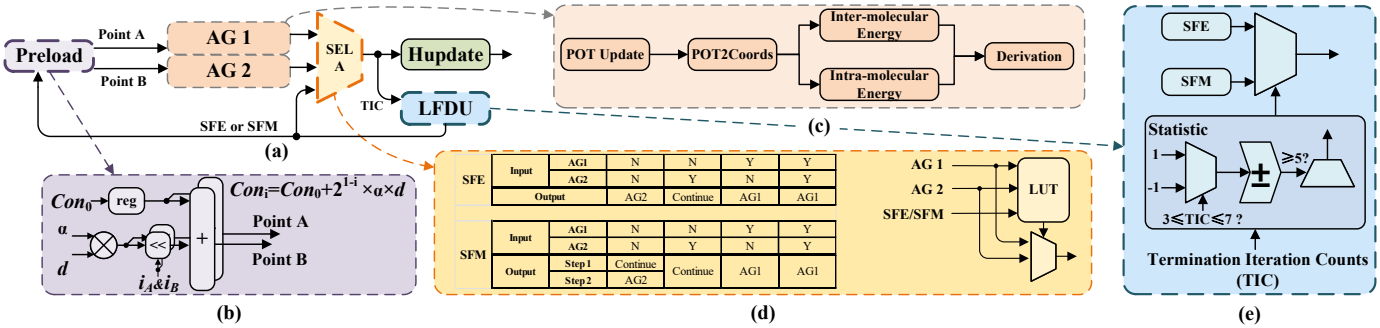


Fig. 13: (a) The architecture of the Bidirectional-AG; (b) Preload unit, it computes based on the equation 4; (c) Five distinct modules in AG; (d) Input and out configurations for selector A; (e) Least frequency decision unit.

C. Hardware Microarchitecture

1) Parameter Accessing: The inter-molecular energy is the sum of 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 energy with a reasonable time overhead. To lower the computation complexity, the inter-molecular energy calculation in Vina is implemented by trilinear interpolations based on the precalculated grid data from the host CPU [41]. As depicted in Fig. 12(a), grid data refers to the corresponding energy values for different types of atoms at various distances, known as the inter-energy parameter. In Vina, there exist 153 different types atom grid data tables such as for Hydrogen atoms, Carbon atoms, Nitrogen atoms, Phosphorus atoms, and so forth. The acquisition of specific data from these tables is accomplished using molecular distance as the index value. In Vina-FPGA-cluster, these grid data tables are stored in the DDR, and the 8 required data for each computation are retrieved beforehand. However, the access to these 8 data points is random and the addresses are discontinuous. To minimize the latency caused by data access, we employed Outstanding transactions [42] in AXI to achieve continuous data access at non-continuous addresses. As depicted in Fig. 12(b), there is a comparison between conventional transactions and outstanding transactions under the AXI protocol. It is evident that for continuous access at non-continuous addresses using conventional transactions, each access necessitates completing a master-slave handshake based on the AXI protocol, which significantly increases data access overhead. In contrast, using Outstanding transactions allows continuous data access with just a single handshake, even when the addresses are non-continuous. Outstanding transactions effectively lowers the latency for master-slave handshakes in non-continuous address data accesses, leading to an average $3.4\times$ increase in data access speed [43]. Subsequently, we optimized the process for inter-molecular energy calculation, separating the memory access, which was tightly coupled in the computational process of Vina-FPGA, into an independent memory access module. This module, along with the computational module, is designed to be fully pipelined, as illustrated in Fig. 12(c). This design ensures efficient data retrieval and maximally conceals the latency of DDR data access. More importantly, it reduces the consumption of on-chip memory resources, such as BRAM and URAM.

2) Bidirectional-AG: The Bidirectional-AG includes a Preload Unit, dual parallel AG units, a Data Selector A, a Hessian Matrix Update Module, and a Least Frequent Decision Unit (LFDU), shown in Fig. 13.

The Preload Unit prepares the necessary input $Cons$ for Bidirectional-AG computations. It computes based on the equation 4, where division operations can be substituted with shift operations, as shown in Fig. 13(b). The a, d, and $Con0$ are pre-determined values. The only variable that needs to be updated is the current iteration value i, which is determined based on the SFE and SFM strategies. Once the SFE and SFM strategies are set, it becomes feasible to prepare in advance for the next computation required values of Point A and Point B.

The dual parallel AG units initiate their search from distinct points, point A and point B. As illustrated in Fig. 13(c) and referenced by Vina-FPGA, each AG unit comprises five sub-modules: POT Update, POT2Coords, Inter-molecular, Intra-molecular, and Derivation modules. Notably, the Inter-molecular and Intra-molecular modules execute in parallel.

The Data Selector A serves to select the output following AG's iterative computation. As illustrated in Fig. 13(d). Data Selector A takes the results from two AG modules and LFDU as inputs, and it produces outputs that comprise both the final computation results and the termination iteration counts based on the strategies outlined in the configuration table. Specifically, under the SFE strategy, computation continues only if AG2 satisfies the condition in Algorithm 2, and it proceeds until other conditions are met to output corresponding results. Under the SFM strategy, the next computational direction is determined based on the current value of AG2 (corresponding to $Con6$). If AG2 meets the judgment condition, the remaining computations are completed between $Con1$ and $Con5$. If AG2 does not meet the condition, the remaining computations are carried out between $Con7$ and $Con10$.

The LFDU determines the strategy for the subsequent AG iterative loops. As depicted in Fig. 13(e), the LFDU receives the termination of iteration counts from Data Selector A. This data undergoes statistical processing via an up-down counter mechanism. When the termination of iteration counts value falls within the range from 3 to 7, the counter increments by one; otherwise, it decreases. If the counter value exceeds 5, the SFM strategy is selected; otherwise, the SFE strategy is chosen.

TABLE III: Resource utilization of Vina–FPGA-cluster (single board)

	Mutate	Metropolis accept	Hupdate	Conformation update	Coords generation	Inter-molecular energy	Intra-molecular energy	Derivation	Others	Total
LUT	6,745	4,666	18,512	24,794	18,725	7,563	4,334	29,530	14,258	129,127 (56.04%)
FF	5,427	5,792	15,966	17,102	26,580	7,135	4,600	22,824	17,104	122,530 (26.59%)
DSP	36	0	499	136	136	72	28	30	4	941 (54.46%)
BRAM	0	0	9.5	0	0	0	251	0	2	262.5 (84.13%)
URAM	0	0	0	0	0	0	4	0	0	4 (4.16%)

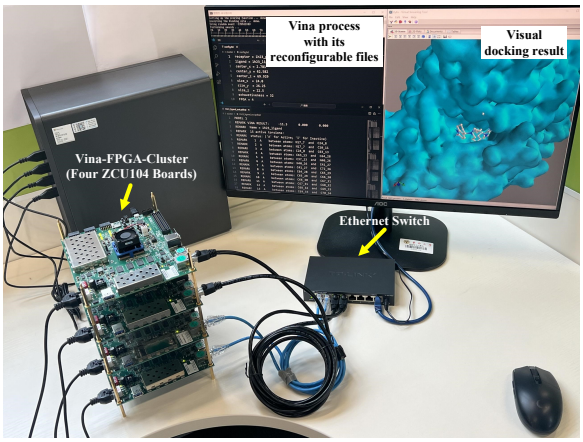


Fig. 14: Experimental setup of Vina-FPGA-cluster: Hardware implementation of four Xilinx Zynq UltraScale+ ZCU104 boards.The visual docking result is generated by specific software that displays information based on the docking result files and this process not included in the computation (T_{E2E}) time.

V. Experiments

A. Experimental Setup

1) Implementation Details: We implement our design on four FPGA boards, Xilinx Zynq UltraScale+ ZCU104, as shown in Fig. 14. Each board consists of a quad-core ARM Cortex-A53 applications processor, 504K system logic cells, 38Mb on-chip memory, and 1728 DSP slices. The FPGA DDR memory has four channels (HP-Port) with 77GB/s memory bandwidth. We develop Vina-FPGA-cluster in Verilog HDL and synthesize the design by Xilinx Vivado 2020.2. The resource utilization and clock rate report are also obtained through Vivado. Each FPGA board utilizes identical resources and operates at the same frequency. As shown in Table III, the Vina-FPGA-cluster on single ZCU104 consumes 129K LUTs (56.04%), 122K FFs (26.59%), 941 DSPs (54.46%), 262.5 BRAMs (84.13%), and 4 URAMs (4.16%). Vina-FPGA-cluster runs at 200MHz. To achieve this frequency under conditions of high BRAM utilization, we generate explicit physical location mappings for BRAM and related components, including CLBs and DSPs. We measure the resulting frequency of the mapped design and interconnect utilization metrics to assess the extent of wiring reduction. It should be emphasized that, despite the substantial residual capacity of URAM, it doesn't guarantee sufficient on-chip memory for full floating-point computational requirements. Additionally, full floating-point operations would lead to an extended computation latency on the FPGA, attributable to the latency enhancements introduced by floating-point IP cores [44].

2) Baselines: As shown in Table IV, we compare our design with state-of-the-art baselines: CPU-only platform (Intel I7-12700KF CPU), Vina-GPU (Nvidia RTX3090) [11], Vina-FPGA [19].

3) Benchmarks: We use the three representative molecular docking datasets as the benchmarks, that are comprised of 85 complexes from the Astex Diversity Set [33], 35 complexes from CASF-2013 [35], and 20 complexes from the Protein Data Bank [34]. They cover a wide range of ligand complexities and targets properties. Each complex file includes an X-ray structure, an initial random pose of its ligand and the corresponding receptor (in .pdbqt format).

4) Accuracy Metrics: We evaluate the accuracy of Vina-FPGA-cluster by:

Docking Energy: Vina typically aims to simulate the docking process between a small molecule (ligand) and a larger molecule (often a protein receptor). Binding energy, often measured in kcal/mol, serves as a key indicator of this affinity. A lower docking energy (numerically more negative) typically signifies a stronger docking affinity.

RMSD: The docking results are typically validated using the output energy between the ligand and receptor, along with the root-mean-square deviation (RMSD) between the output ligand conformation and the X-ray measurements, which serve as the ground truth.

5) Performance Metrics: We evaluate the performance of Vina-FPGA-cluster by:

Latency of hardware execution (LoH) T_{LoH}: The latency of hardware execution refers to the time taken to perform the Vina computation on the hardware accelerator. Prior to runtime, essential data such as global data, inter-energy parameters, and intra-energy parameters are pre-stored in the FPGA's registers, DDR, and on-chip memory (BRAM & URAM), respectively.

End-to-End (E2E) latency T_{E2E}: The T_{E2E} of Vina-FPGA-cluster includes (1) the latency of CPU preparing initial data T_{ini}, (2) the latency of CPU-FPGA data transaction T_{tran} (including result transaction), (3) the latency of executing Vina computation on the accelerator (Latency of hardware execution T_{LoH}). Then, the E2E latency of Vina-FPGA-cluster is calculated by: T_{E2E}=T_{ini}+T_{tran}+T_{LoH}.

Energy Efficiency (EE): The energy efficiency means the value of useful power-latency-product [46], denoted as below:

$$EE = \text{Core's power consumption (W)} \times T_{E2E}. \quad (10)$$

B. Impact of the Optimizations

To demonstrate the effectiveness of the optimizations proposed in Vina-FPGA-Cluster, we run the docking process on a single board within Vina-FPGA-Cluster and compared it with

TABLE IV: Specifications of platforms

Platforms	Processor	Platform Technology	Frequency	On-chip Memory	Memory Bandwidth
CPU	Intel I7-12700KF	Intel 7 nm	3.6 GHz	25MB L3 cache	76.8 GB/s (DDR 5)
Vina-GPU [11]	Nvidia RTX3090	TSMC 7 nm	1.7 GHz	6 MB L2 cache	936.2 GB/s
Vina-FPGA [19]	Kintex UltraScale XCKU060	TSMC 20 nm	150 MHz	4.75 MB (BRAM only)	77 GB/s
Vina-FPGA-cluster	Zynq UltraScale+ ZCU104	TSMC 16 nm	200 MHz	4.75 MB (BRAM + URAM)	77 GB/s

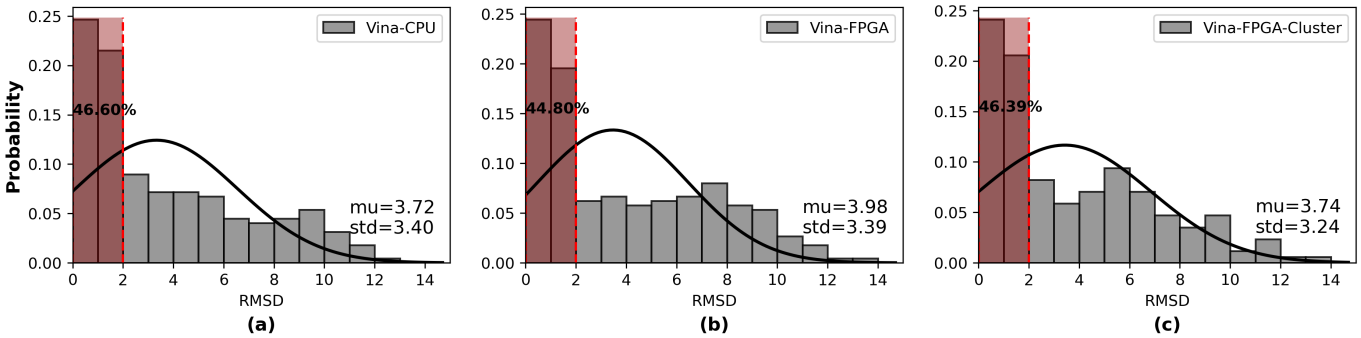


Fig. 15: Distribution of RMSD for different implementations. Vina-CPU is the original Vina deployed in CPU. The crimson rectangle denotes the percentage of successful docking. (Normally, a docking conformation with a RMSD less than 2Å is considered a successful one [45]).

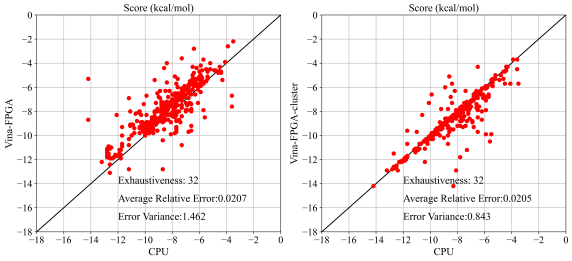


Fig. 16: Docking energy comparisons of Vina-FPGA and Vina-FPGA-cluster under three representative molecular docking dataset.

Vina-FPGA in terms of accuracy, latency and scalability. Despite Vina-FPGA and Vina-FPGA-cluster being implemented on separate FPGA platforms, the comparison of their performance on the same dataset also provides meaningful insights. This approach is consistent with the established methodology for contrasting FPGA-based deployment designs and related work in the field [47], [48].

1) Accuracy optimization: To highlight the improvements in molecular docking accuracy from our quantization strategy, we assess its impact by examining docking energy and RMSD, fundamental measures of docking performance. Fig. 16 shows the energy differences in docking performed by Vina-FPGA and Vina-FPGA-cluster under the same random seed at an exhaustiveness of 32. Each red dot in the figure represents the docking energy for a given ligand-receptor complex, with the x and y coordinates of the dot indicating the original Vina docking energy and FPGA docking energy, respectively, under the same random seed. Ideally, all red dots should align along the diagonal line. However, it's observed that while most output energies align with the diagonal, there are deviations for some points. Notably, data from Vina-FPGA-cluster tends to align more closely with the diagonal line, showing improved accuracy and reduced variance compared to Vina-FPGA. Similarly, we reconstruct the RMSD experiment at an exhaustiveness of 32 for three typical datasets, with results shown in Fig. 15. Vina-FPGA-Cluster shows significant

TABLE V: The latency improvement of Vina-FPGA-Cluster (on a single board) compared to Vina-FPGA

	Vina-FPGA	Vina-FPGA-Cluster	Improvement
T_{ini}	1.903 s	1.903 s	1.00×
T_{tran}	0.156 s	0.446 s	0.35×
T_{LoH}	46.343 s	11.883 s	3.90×
T_{E2E}	48.402 s	14.232 s	3.40×

improvements in mean value (mu) and variance (std) over Vina-FPGA, with values of 3.74 and 3.24, respectively, closely aligning with the original Vina. The success ratio loss is less than 0.2%. These experimental results effectively demonstrate the accuracy enhancement of Vina-FPGA-Cluster compared to Vina-FPGA.

2) Latency optimization: To demonstrate the latency reduction brought by Vina-FPGA-cluster's optimized design, we compare the key latency metrics of Vina-FPGA and Vina-FPGA-cluster on a single board. These metrics include T_{ini}, T_{tran}, and T_{LoH}, detailed in Table V. Since Vina-FPGA and Vina-FPGA-cluster use the same total bit widths for quantization data, their T_{ini} values match. Vina-FPGA communicates with the PC via PCIe, offering greater bandwidth and thus shorter transmission times. However, this setup limits scalability, as discussed further in Section V-B3. Benefiting from the Bidirectional-AG design, Vina-FPGA-cluster's T_{LoH} improves by 3.90× compared to Vina-FPGA. This substantial boost in T_{LoH} leads to a 3.40× acceleration in T_{E2E} for Vina-FPGA-cluster, even on a single board, relative to Vina-FPGA.

3) Scalability optimization: Regarding scalability optimization, we analyze from resource consumption. Fig. 17 illustrates the LUTs and FFs utilization for various modules in Vina-FPGA and Vina-FPGA-cluster, which are the basic FPGA resources [49]. The data reveals that Vina-FPGA-cluster expends over twice the resources in its core computational module. However, considering this leads to more than a twofold increase in processing speed, the resource trade-off is justified. Significantly, thanks to Vina-FPGA-cluster's transition from PCIe to an ARM-driven Ethernet port and offloading container module computations to ARM, it demonstrates reduced overall

TABLE VI: Performance and energy comparison of CPU, Vina-GPU, Vina-FPGA, and Vina-FPGA-cluster (on four boards)

Evaluation metric of performance	CPU-only	Vina-GPU [11]	Vina-FPGA [19]	Vina-FPGA-Cluster	Improvement over CPU	Improvement over Vina-GPU	Improvement over Vina-FPGA
T_{LoH}	180.54s	7.27s	46.34s	**2.98s**	60.58×	2.44×	15.55×
T_{E2E}	182.28s	9.20s	48.40s	**6.67s**	27.33×	1.38×	7.26×
Power	Core: 47.34W	Core: 67.2W	Core: **4.70W**	Core: 19.48W	2.43×	3.45×	0.24×
	Board:N/A	Board: 203W	Board: **12.84W**	Board: 53.16W			
EE	8629.14J	618.24J	227.48J	**129.93J**	66.41×	4.76×	1.75×

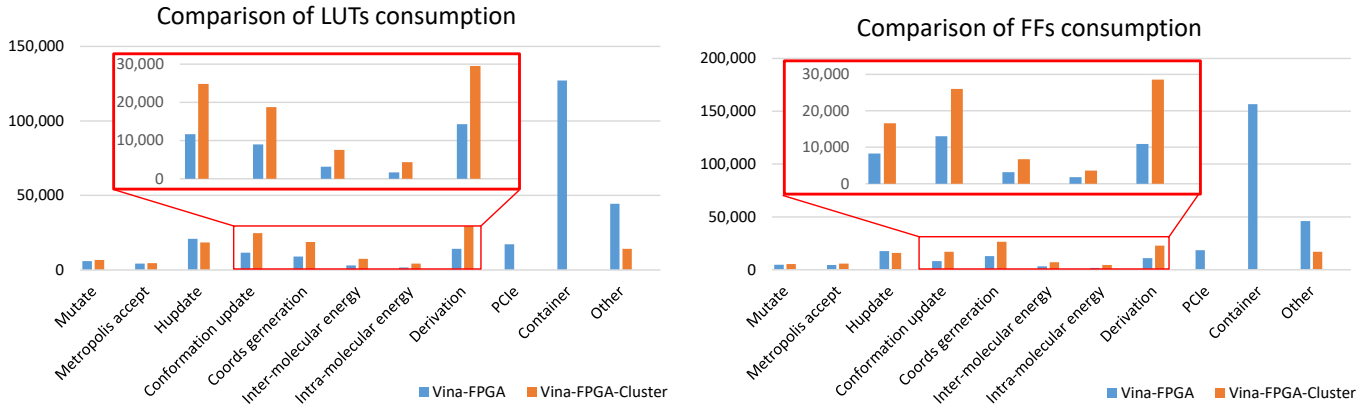


Fig. 17: The basis FPGA resource consumption comparison of Vina-FPGA and Vina-FPGA-cluster (on a single board).

resource consumption. These optimizations will be critical for future endeavors in deploying multiple computation units on a single high-capacity FPGA, optimizing both performance and resource efficiency.

C. Cross Platform Comparison

In our study, we configure a cluster using four ZCU104 boards to execute molecular docking process. We compare the performance of this setup with three baseline platforms in terms of T_{LoH}, T_{E2E}, power consumption, and energy efficiency. The baseline benchmark platforms are: (1) a CPU-only platform (Intel I7-12700KF), (2) Vina-GPU (Nvidia RTX 3090), and (3) Vina-FPGA (Kintex UltraScale XCKU060). The configurations for the molecular docking pockets are uniformly maintained across all platforms, with identical initial random seeds and the exhaustiveness set to 32. The comparative results are presented in Table VI.

In evaluating the T_{LoH} metric, the Vina-FPGA-cluster, leveraging its multi-tiered parallel architecture, achieves a substantial speedup ranging from 2.44× ∼ 60.58× relative to the three established baseline platforms. This acceleration is attributed to the multi-level parallelism: fine-grained module-level pipeline parallel processing, concurrent computation of core computations enabled by the Bidirectional-AG design, and the parallel computations across multiple boards.

Regarding the T_{E2E}, the increased data transmission requirements inherent to the Vina-FPGA-cluster lead to a slight reduction in speedup. However, given the scalability of this architecture, supporting expansions beyond 64 boards, such trade-offs are considered within acceptable bounds.

In terms of power consumption, the energy usage of Vina-FPGA-cluster (four FPGA boards) remains significantly lower than that of the CPU and GPU. Compared to the Vina-FPGA, the power consumption of each FPGA in the Vina-FPGA-cluster is slightly higher due to the presence of the ARM hard core in the Zynq used in the cluster.

In the context of energy efficiency (EE), the Vina-FPGA-cluster demonstrates enhancements ranging from 1.75× ∼ 66.41× in comparison to three baseline platforms. A key observation from our experiments emphasizes that the Vina-FPGA-cluster, while achieving a 1.38× speed increase compared to Vina-GPU, maintains core power consumption and board-level power consumption at only 28.99% and 26.19%, respectively. It highlights the efficacy and potential of FPGA-based solutions in high-performance computing scenarios.

D. Analysis

The results of Table V indicate that choosing Ethernet as the communication method for the cluster, to meet the plug-and-play requirements, introduces some communication latency. Therefore, to further analyze the entire system, we construct a roofline model. The original roofline model [50] is as follows:

$$\frac{\#FP\_ops}{Latency} = min(Peak\_FLOPS, \frac{\#FP\_ops}{Bytes} \times Peak\_MBW). \quad (11)$$

where the $\#FP\_ops$ represents the number of floating-point operations, $Latency$ represents the time consumption of computing, and $Peak\_FLOPS$ represents the system's performance (throughput). The $\#FP\_ops/Bytes$ denotes the operational intensity, where $Bytes$ refers to the amount of data that needs to be accessed from memory for computation. The $Peak\_MBW$ represents the peak memory bandwidth.

Due to the variable data bit widths in Vina-FPGA-Cluster, we equate the number of docking iterations (Exhaustiveness, Ex) to the number of floating-point operations ($\#FP\_ops$) in the original model. The peak bandwidth ($Peak\_BW$) of Ethernet communication is equated to the $Peak\_MBW$, and the volume of data transferred is defined as the amount of

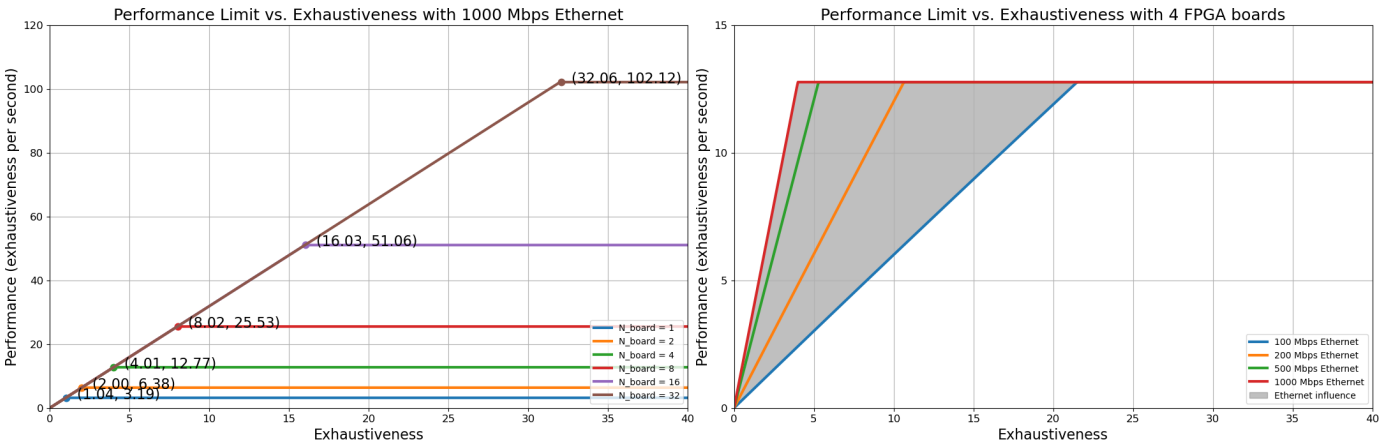


Fig. 18: (a) Analysis for multi-board designs. (b) Roofline model with ceilings under different peak bandwidth of Ethernet.

data needed to complete one docking iteration (including data required for computation and data returned with results). Therefore, our model is defined as:

$$\frac{Ex}{Latency} = min(Peak\_perf, \frac{Ex}{Bytes} \times Peak\_BW). \quad (12)$$

Each Ex is indivisible, and a FPGA board can only execute one Ex at a time. Therefore, when the Ex is less than the number of boards, the system's peak performance ($Peak\_perf$) is the number of Ex operations; when the Ex is greater than or equal to the number of boards, the $Peak\_perf$ is limited by the number of boards. The definition of $Peak\_perf$ is as follows:

$$Peak\_perf = \begin{cases} Ex \times \frac{1}{T_{HW}}, & Ex < N_{board} \\ N_{board} \times \frac{1}{T_{HW}}, & N_{board} \leqslant Ex \end{cases}, \quad (13)$$

where N_{board} represents the number of boards, T_{HW} is the time it takes for the hardware to perform one docking operation, and $Bytes$ is the amount of data that communicated between the host CPU to the FPGA for the whole computation. Specifically, the definition of $Bytes$ is as follow:

$$Bytes = Bytes_{\text{intra}} + Bytes_{\text{inter}} + (Bytes_{\text{global}} + Bytes_{\text{results}}) \times Ex. \quad (14)$$

The data above corresponds to the intra-molecular energy table (up to 2,510,424 Bytes), inter-molecular energy table (up to 19,143,020 Bytes), global variables (up to 18,816 Bytes), and computation results (up to 13,840 Bytes), respectively. During the computation process, $Bytes_{\text{intra}}$ and $Bytes_{\text{inter}}$ data are only required to be sent from the host to the FPGA during the first Ex iteration. After that, the communication demand ($Bytes_{\text{global}}$ and $Bytes_{\text{results}}$) for each Ex iteration is less than 35KB.

Under these definitions, we analyze the system's performance across various numbers of boards and communication bandwidths. Fig. 18(a) displays the impact of different board quantities on performance under 1000Mbps Ethernet. It can be observed that when the number of Exhaustiveness iterations required by Vina approximately exceeds the number of boards, the performance is compute-bound. Otherwise, the performance is ultimately communication-bound. Fig. 18(b) shows how different communication bandwidths affect performance with 4 boards. It can be concluded that communication bandwidth is not the bottleneck of the proposed FPGA cluster, presently, considering the huge workload of Vina computation and the relatively small volume of data transmission.

VI. Conclusion and Future Work

In this work, we present Vina-FPGA-cluster, an FPGA cluster-based molecular docking acceleration tool. Building upon our previous efforts, Vina-FPGA-cluster enhances docking precision through optimized fixed-point quantization data formats, accelerates evaluation and design optimization via a SystemC-based hardware model, and achieves optimized speedup by refining the innermost iterative algorithm. Experimental results demonstrate that compared to the state-of-the-art CPU and FPGA (Vina-FPGA) platforms, our implementation achieves an acceleration of 27.33× and 7.26×, respectively. Compared to the most advanced GPU (Vina-GPU) deployments, our system consumes only 28.99% of the power while delivering 1.38× performance. In future work, we plan to further optimize the computational flow of Vina-FPGA, aiming to enhance the efficiency of different computational modules and reduce idle time.

Future work involves further optimizing and expanding Vina-FPGA-cluster with two primary objectives: (1) Revise the Vina software application to support virtual screening similar within Vina-GPU2. This adaptation will enhance the tool's utility in large-scale drug discovery efforts. (2) We plan to develop a new microarchitecture that facilitates pipeline between different computational modules, thereby improving the operational efficiency of the processing units. These advancements aim to refine Vina-FPGA-cluster's performance, aligning it more closely with cutting-edge computational needs in bioinformatics.

Appendix

Table VII presents the variable names, counts, and data bit widths for the data communicated between the CPU and FPGA during initialization, where ID1-ID23 represent global data. Table VIII presents the meanings of the data communicated between the CPU and FPGA during initialization.

TABLE VII: The list of initialized data sent from CPU to FPGAs

ID	Name	Entry Number	Bits for each entry	ID	Name	Entry Number	Bits for each entry
1	lim	3 × 1	7	**14**	atom	Natom × 3	18
2	m_factor_inv	3 × 1	18	**15**	relative_origin	(Ntorsion+1) × 3	18
3	m_factor	3 × 1	18	**16**	relative_axis	(Ntorsion+1) × 3	18
4	m_init	3 × 1	18	**17**	parent_root	Ntorsion+1	6
5	num_torsion	1 × 1	6	**18**	children_map	32 × 32	1
6	num_coords	1 × 1	7	**19**	atom_range	(Ntorsion+1) × 2	7
7	atoms_el	Natom × 1	5	**20**	derivative table	313803 × 1	14
8	table	Natom × 1	5	**21**	Position	3 × 1	18
9	pair_atom	1 × 1	12	**22**	Orientation	4 × 1	18
10	a	Npair-atom × 1	7	**23**	Torsion	Ntorsion × 1	18
11	b	Npair-atom × 1	7	**24**	Intra-molecular energy table	313803 × 1	14
12	type	Npair-atom × 1	8	**25**	Inter-molecular energy table	281515 × 17	14
13	coords	Natom × 3	18				

TABLE VIII: The list of notes for initialized data

ID	Note	ID	Note
1	The size of the search space	**14**	The absolute coordinates of ligand atoms
2	The invert scaling factor of the search space	**15**	The current sub-root node minus its parent root node
3	The scaling factor of the search space	**16**	The normalization of the current sub-root node minus its parent node
4	The starting position of the search space	**17**	The coordinates of the parent root node
5	The number of torsions	**18**	The child node information of various nodes
6	The number of atoms	**19**	The range of variation for the Torsion part of the ligand.
7	The types of atoms	**20**	The lookup table for intra-molecular derivatives
8	The index ID for lookup	**21**	Initial Position
9	The number of active ligand atom pairs	**22**	Initial Orientation
10	The index IDs of two neighboring atoms	**23**	Initial Torsion
11	The index IDs of two neighboring atoms	**24**	The lookup table for intra-molecular energy
12	The types of atom pairs with forces acting	**25**	The lookup table for inter-molecular energy
13	The initialization of coordinates		

ACKNOWLEDGMENT

The authors would like to thank VeriMake Innovation Lab for providing developing and testing equipment. This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 61974024 and OPPO research Fund.

REFERENCES

[1] J. Caballero, "The latest automated docking technologies for novel drug discovery," *Expert Opinion on Drug Discovery*, vol. 16, no. 6, pp. 625–645, 2021.

[2] K. Xu, J. Zhang, X. Duan, X. Wan, N. Huang, B. Schmidt, W. Liu, and G. Yang, "Redesigning and optimizing ucsf dock3.7 on sunway taihulight," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4458–4471, 2022.

[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," *Journal of computational chemistry*, vol. 31, no. 2, pp. 455–461, 2010.

[4] T. N. H. Pham, T. H. Nguyen, N. M. Tam, T. Y. Vu, N. T. Pham, N. T. Huy, B. K. Mai, N. T. Tung, M. Q. Pham, V. V. Vu *et al.*, "Improving ligand-ranking of autodock vina by changing the empirical parameters," *Journal of Computational Chemistry*, vol. 43, no. 3, pp. 160–169, 2022.

[5] R. Agarwal and J. C. Smith, "Speed vs accuracy: effect on ligand pose accuracy of varying box size and exhaustiveness in autodock vina," *Molecular Informatics*, vol. 42, no. 2, p. 2200188, 2023.

[6] G. M. Morris, R. Huey, W. Lindstrom, M. F. Sanner, R. K. Belew, D. S. Goodsell, and A. J. Olson, "Autodock4 and autodocktools4: Automated docking with selective receptor flexibility," *Journal of computational chemistry*, vol. 30, no. 16, pp. 2785–2791, 2009.

[7] L. Solis-Vasquez, A. F. Tillack, D. Santos-Martins, A. Koch, S. LeGrand, and S. Forli, "Benchmarking the performance of irregular computations in autodock-gpu molecular docking," *Parallel Computing*, vol. 109, p. 102861, 2022.

[8] 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 transactions on computational biology and bioinformatics*, vol. 9, no. 5, pp. 1266–1272, 2012.

[9] 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," *Scientific reports*, vol. 7, no. 1, p. 15451, 2017.

[10] C. Gorgulla, A. Boeszoermenyi, Z.-F. Wang, P. D. Fischer, P. W. Coote, K. M. Padmanabha Das, Y. S. Malets, D. S. Radchenko, Y. S. Moroz, D. A. Scott *et al.*, "An open-source drug discovery platform enables ultra-large virtual screens," *Nature*, vol. 580, no. 7805, pp. 663–668, 2020.

[11] S. Tang, R. Chen, M. Lin, Q. Lin, Y. Zhu, J. Ding, H. Hu, M. Ling, and J. Wu, "Accelerating autodock vina with gpus," *Molecules*, vol. 27, no. 9, p. 3041, 2022.

[12] X. Zhou, M. Ling, Q. Lin, S. Tang, J. Wu, and H. Hu, "Effectiveness analysis of multiple initial states simulated annealing algorithm, a case study on the molecular docking tool autodock vina," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2023.

[13] J. Ding, S. Tang, Z. Mei, L. Wang, Q. Huang, H. Hu, M. Ling, and J. Wu, "Vina-gpu 2.0: Further accelerating autodock vina and its derivatives with graphics processing units," *Journal of Chemical Information and Modeling*, vol. 63, no. 7, pp. 1982–1998, 2023.

[14] 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, 2015.

[15] A. A. Khan and M. Zakarya, "Energy, performance and cost efficient cloud datacentres: A survey," *Computer Science Review*, vol. 40, p. 100390, 2021.

[16] Y.-K. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "In-depth analysis on microarchitectures of modern heterogeneous cpu-fpga platforms," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–20, 2019.

[17] I. Pechan, B. Fehér, and A. Bérces, "Fpga-based acceleration of the autodock molecular docking software," in *6th Conference on Ph. D. Research in Microelectronics & Electronics*. IEEE, 2010, pp. 1–4.

[18] 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 *FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers*. VDE, 2018, pp. 1–10.

[19] M. Ling, Q. Lin, R. Chen, H. Qi, M. Lin, Y. Zhu, and J. Wu, "Vina-fpga: A hardware-accelerated molecular docking tool with fixed-point quantization and low-level parallelism," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 484–497, 2022.

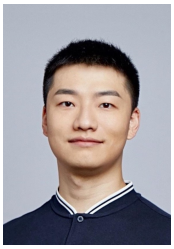
[20] K. K. Chaudhary and N. Mishra, "A review on molecular docking: novel tool for drug discovery," *Databases*, vol. 3, no. 4, p. 1029, 2016.
[21] K. B. Santos, I. A. Guedes, A. L. Karl, and L. E. Dardenne, "Highly flexible ligand docking: Benchmarking of the dockthor program on the leads-pep protein–peptide data set," *Journal of Chemical Information and Modeling*, vol. 60, no. 2, pp. 667–683, 2020.
[22] D. S. Goodsell and A. J. Olson, "Automated docking of substrates to proteins by simulated annealing," *Proteins: Structure, Function, and Bioinformatics*, vol. 8, no. 3, pp. 195–202, 1990.
[23] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson, "Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function," *Journal of computational chemistry*, vol. 19, no. 14, pp. 1639–1662, 1998.
[24] R. Huey, G. M. Morris, A. J. Olson, and D. S. Goodsell, "A semiempirical free energy force field with charge-based desolvation," *Journal of computational chemistry*, vol. 28, no. 6, pp. 1145–1152, 2007.
[25] S. Ruiz-Carmona, D. Alvarez-Garcia, N. Foloppe, A. B. Garmendia-Doval, S. Juhos, P. Schmidtke, X. Barril, R. E. Hubbard, and S. D. Morley, "rdock: a fast, versatile and open source program for docking ligands to proteins and nucleic acids," *PLoS computational biology*, vol. 10, no. 4, p. e1003571, 2014.
[26] Z. Wang, H. Sun, X. Yao, D. Li, L. Xu, Y. Li, S. Tian, and T. Hou, "Comprehensive evaluation of ten docking programs on a diverse set of protein–ligand complexes: the prediction accuracy of sampling power and scoring power," *Physical Chemistry Chemical Physics*, vol. 18, no. 18, pp. 12 964–12 975, 2016.
[27] G. M. Verkhivker, D. Bouzida, D. K. Gehlhaar, P. A. Rejto, S. Arthurs, A. B. Colson, S. T. Freer, V. Larson, B. A. Luty, T. Marrone *et al.*, "Deciphering common failures in molecular docking of ligand-protein complexes," *Journal of computer-aided molecular design*, vol. 14, pp. 731–751, 2000.
[28] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in fpga design for cnn-based object detectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 6, pp. 2450–2464, 2020.
[29] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K.-H. So, X. Qian, Y. Wang, and X. Lin, "Mix and match: A novel fpga-centric deep neural network quantization framework," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 208–220.
[30] Q. Huang, D. Wang, Z. Dong, Y. Gao, Y. Cai, T. Li, B. Wu, K. Keutzer, and J. Wawrzynek, "Codenet: Efficient deployment of input-adaptive object detection on embedded fpgas," in *2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 206–216.
[31] Y.-H. Lai, E. Ustun, S. Xiang, Z. Fang, H. Rong, and Z. Zhang, "Programming and synthesis for software-defined fpga acceleration: status and future prospects," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 14, no. 4, pp. 1–39, 2021.
[32] D. Lohar, C. Jeangoudoux, A. Volkova, and E. Darulova, "Sound mixed fixed-point quantization of neural networks," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–26, 2023.
[33] M. J. Hartshorn, M. L. Verdonk, G. Chessari, S. C. Brewerton, W. T. Mooij, P. N. Mortenson, and C. W. Murray, "Diverse, high-quality test set for the validation of protein- ligand docking performance," *Journal of medicinal chemistry*, vol. 50, no. 4, pp. 726–741, 2007.
[34] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.
[35] M. Su, Q. Yang, Y. Du, G. Feng, Z. Liu, Y. Li, and R. Wang, "Comparative assessment of scoring functions: the casf-2016 update," *Journal of chemical information and modeling*, vol. 59, no. 2, pp. 895–913, 2018.
[36] T. Geng, C. Wu, C. Tan, C. Xie, A. Guo, P. Haghi, S. Y. He, J. Li, M. Herbordt, and A. Li, "A survey: Handling irregularities in neural network acceleration with fpgas," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2021, pp. 1–8.
[37] Y. Zhang, H. Ren, and B. Khailany, "Opportunities for rtl and gate level simulation using gpus," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–5.
[38] P. Ravi, "Future of iot: Design and deployment challenges," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 26, no. 1, pp. 54–58, 2019.
[39] AMD Xilinx. Qemu user documentation. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/821395464/QEMU+User+Documentation
[40] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Extending the transaction level modeling approach for fast communication architecture exploration," in *Proceedings of the 41st annual Design Automation Conference (DAC)*, 2004, pp. 113–118.
[41] M. M. Jaghoori, B. Bleijlevens, and S. D. Olabarriaga, "1001 ways to run autodock vina for virtual screening," *Journal of computer-aided molecular design*, vol. 30, pp. 237–249, 2016.
[42] ARM. AMBA® AXI protocol specification. [Online]. Available: https://developer.arm.com/documentation/ihi0022/latest
[43] P. Holzinger, D. Reiser, T. Hahn, and M. Reichenbach, "Fast hbm access with fpgas: Analysis, architectures, and applications," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 152–159.
[44] A. Sanchez, E. Todorovich, and A. De Castro, "Impact of the hardened floating-point cores on hil technology," *Electric Power Systems Research*, vol. 165, pp. 53–59, 2018.
[45] 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, 2022.
[46] Y. Hu, Y. Zhu, H. Chen, R. Graham, and C.-K. Cheng, "Communication latency aware low power noc synthesis," in *Proceedings of the 43rd annual Design Automation Conference (DAC)*, 2006, pp. 574–579.
[47] X. Cai, Y. Wang, X. Ma, Y. Han, and L. Zhang, "Deepburning-seg: Generating dnn accelerators of segment-grained pipeline architecture," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1396–1413.
[48] P. Peng, K. Jiang, M. You, J. Xie, H. Zhou, W. Xu, J. Lu, X. Li, and Y. Xu, "Design of an efficient cnn-based cough detection system on lightweight fpga," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 17, no. 1, pp. 116–128, 2023.
[49] D. B. Thomas and W. Luk, "Using fpga resources for direct generation of multivariate gaussian random numbers," in *2009 International Conference on Field-Programmable Technology*. IEEE, 2009, pp. 344–347.
[50] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, p. 65–76, apr 2009.

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.

Zhihao Feng received the B.S. degree from Tibet University, Lhasa, China, in 2021. 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.



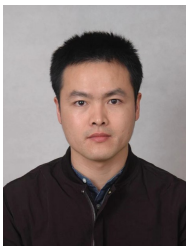
Ruiqi Chen (Member, IEEE) received the B.S. degree in electronic science and technology from Southeast University Chengxian College, Nanjing, China, in 2017, and the M.S. degree in integrated circuit engineering from Fuzhou University, Fuzhou, China, in 2020. From 2020 to 2023, he served as a research assistant at VeriMake Innovation Lab and Fudan University, respectively. He is currently a research assistant with Southeast University. His current research interests include domain-specific architecture and FPGA-based accelerators.

Yi Shao received the BS, degree from Nanjing Tech University, Nanjing, China, in 2023. 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.

Shidi Tang received the BS degree in communication engineering from Central China Normal University, Wuhan, China, in 2020, and the MS degree in biomedical engineering from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2023. He is currently working toward the PhD degree at the Southeast University, Nanjing, China.

His current research interests include domain specific accelerator, efficient machine learning and parallel computing.



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.