Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Vina-FPGA2: a high level parallelized hardware accelerated molecular docking tool based on the inter module pipeline*

Ming LING†‡[1], Shidi TANG[1], Ruiqi CHEN†‡[2], Xin LI[1], Yanxiang ZHU[3]

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

[2]*Department of Electronics and Informatics, Vrije Universiteit Brussel, Brussels 1050, Belgium*

[3]*VeriMake Innovation Laboratory, Nanjing Renmian Integrated Circuit Company Ltd, Nanjing 210088, China*

†E-mail: trio@seu.edu.cn; ruiqi.chen@vub.be

Received Oct. 22, 2024; Revision accepted Sept. 28, 2025; Crosschecked

Unedited

Abstract: AutoDock Vina (Vina) is a widely adopted molecular docking tool, often regarded as a standard or used as a baseline in numerous studies. However, its computational process is highly time-consuming. The pioneering FPGA-based accelerator of Vina, known as Vina-FPGA, offers a high energy-efficiency approach to speedup the docking process. However, the computation modules in the Vina-FPGA design are not efficiently utilized. This is due to Vina exhibiting irregular behaviors in the form of nested loops with changing upper bounds and differing control flows. Fortunately, Vina employs the Monte Carlo iterative search method, which requires independent computations for different random initial inputs (tasks). This characteristic provides an opportunity to implement further parallel computation designs. To this end, this paper proposes Vina-FPGA2, an inter-module pipeline design for further accelerating Vina-FPGA. First, we utilized task independence by sequentially filling tasks into computation modules. Then, we implemented an inter-module pipeline parallel design by the Tag Checker module and architectural modifications, named Vina-FPGA2-Baseline. Next, to achieve resource-efficient hardware implementation, we described it as an optimization problem and developed a reinforcement learning-based solver. Targeting the Xilinx UltraScale XCKU060 platform, this solver yields a more efficient implementation, named Vina-FPGA2-Enhanced. Finally, experiments show that Vina-FPGA2-Enhanced achieves an average 12.6× performance improvement over the CPU and a 3.3× improvement over Vina-FPGA. Compared to Vina-GPU, Vina-FPGA2 achieves a 7.2× enhancement in energy efficiency.

Key words: AutoDock Vina; Hardware accelerator; Field-programmable gate array; Software/hardware co-design
https://doi.org/10.1631/FITEE.2400941 **CLC number:**

1 Introduction

Molecular docking is a pivotal step in drug design, leveraging computer algorithms to quickly screen candidate drug molecules (Chen, 2015; Caballero, 2021). Molecular docking tools employ specific search algorithms to speedup the process of searching for optimal results in a huge potential solution space (Salmaso and Moro, 2018). Diverse molecular docking tools adopt distinct search algorithms, and AutoDock Vina (Trott and Olson, 2010) stands out among numerous molecular docking tools with its excellent performance. This is attributed to the effectiveness of the multi-initial-state simulated annealing algorithm (Kirkpatrick et al., 1983) and the quasi-Newton optimization search algorithm used by Vina. The computation process across multiple ini-

‡ Corresponding authors

* Project supported by the National Nature Science Foundation of China (No. 92464301)

© Zhejiang University Press 2025

tial states is independent, allowing Vina to deploy the calculations of various initial states on multi-core CPUs for parallel processing. Simultaneously, the quasi-Newton optimization search algorithm expedites the attainment of the optimal value.

Although Vina delivers excellent docking accuracy and speed, its CPU-based execution remains time- and energy-intensive due to irregular control flows and deeply nested loops with dynamic bounds, which hinder parallelism. To mitigate this, researchers have explored various acceleration strategies. VirtualFlow (Gorgulla et al., 2020) employs up to 160,000 CPUs for large-scale virtual screening. Vina-GPU (Tang et al., 2022) and its successor Vina-GPU2 (Ding et al., 2023), integrating QuickVina2 (Alhossary et al., 2015) and QuickVina-W (Hassan et al., 2017), exploit GPU parallelism to achieve up to 21× speedups (Zhou et al., 2023). While these methods significantly enhance performance, they also incur high energy costs. FPGAs, with their reconfigurability, parallelism, and energy efficiency (Choi et al., 2019; Mittal, 2020; Shawahna et al., 2018; Belletti et al., 2008), offer a compelling alternative. In our previous work, we introduced Vina-FPGA (Ling et al., 2022), which leverages pipelined and partially parallel modules to deliver a 3.7× speedup over CPUs while consuming only 45% of the energy used by Vina-GPU.

However, the computation modules are underutilized and inefficient within Vina-FPGA. It is constrained by the extensive iterative dependencies in the Vina algorithm, that is, the input of the next computation module depends on the output of the previous one, and the input for the next iteration is determined by the output of the previous iteration. This means that during the computation process of Vina-FPGA, only one computation module is running while the others remain idle. Utilizing these idle computation modules can not only enhance the overall hardware resource efficiency of Vina-FPGA but also further accelerate the Vina computation process. Vina's computation process is based on the Monte Carlo method with multiple initial states for simulated annealing, in which each state is computed independently. This independence provides an opportunity for us to implement an inter-module pipeline design.

To achieve the pipelined parallel computing mode with multiple initial states among the hardware modules of Vina-FPGA, we proposed Vina-FPGA2, a High-Level Parallelized Hardware-Accelerated Molecular Docking Tool. First, we implemented an inter-module pipeline design through the Tag Checker module, Metropolis Accept module, and adjustments to the Vina-FPGA architecture, naming it Vina-FPGA2-Baseline. Next, we formally described Vina-FPGA2-Baseline to further improve deployment efficiency under FPGA resource constraints (LUT, FF, BRAM, etc.). Then, we transformed the entire problem into an optimization problem to minimize the power-delay product. To solve this problem, we designed a reinforcement learning-based solver for a rapid and automatic solution. Finally, under the constraints of deploying on the UltraScale XCKU060, we developed the Vina-FPGA2-Enhanced design. Our main contributions are as follows:

- **An innovative inter-module pipelined parallel architecture**: We proposed a Tag Checker module, which is the basic component for inter-module pipeline design with Vina. The Tag Checker module serves as the central controller for the multitasking computation flow of Vina. Based on these modules and some architectural adjustments, we developed Vina-FPGA2-Baseline that achieves an average speedup of 2.42× compared to Vina-FPGA. Furthermore, the energy efficiency ratio has been improved to 2.96× over Vina-FPGA. Remarkably, this performance enhancement is achieved with only a 15% increase in hardware resources overhead while under the same processing frequency (150 MHz).

- **A formular description and reinforcement learning-based solver**: Building on Vina-FPGA2-Baseline, we formulated its design to minimize the power-delay product and added several constraints. To rapidly and automatically determine the optimal number of computation lanes and task deployments for a specific FPGA, we designed a reinforcement learning-based solver.

- **A resource-efficient hardware module deployment strategy for Vina-FPGA2-Enhanced**: Benefiting from the solver, we got the recommended implementation strategy

Unedited

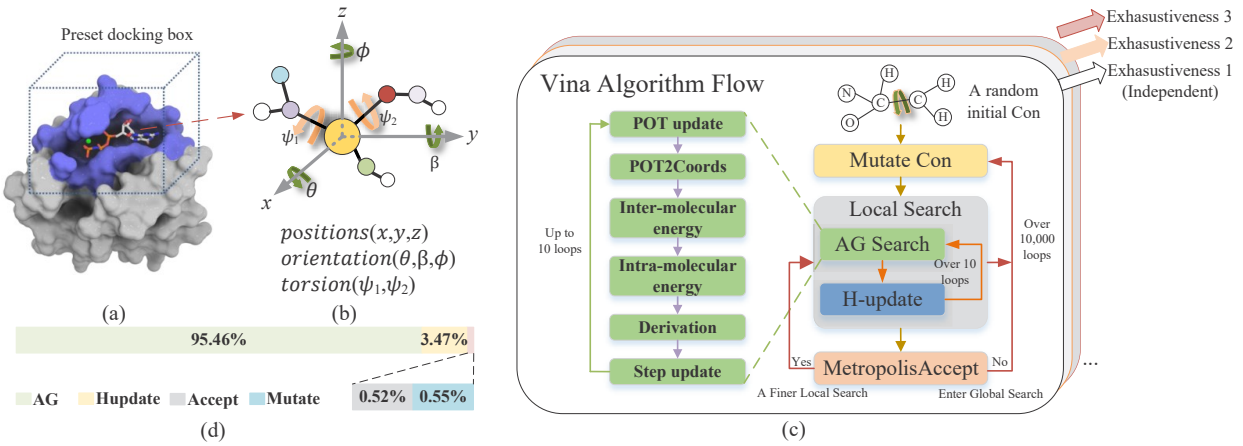

Fig. 1 (a) Schematic diagram of a ligand-receptor docking, (b) illustration of the conformation of a ligand molecule, (c) the algorithm flow of Vina, and (d) the profiling of AG , H-update, Accept, and Mutate, respectively.

to achieve optimal power latency production (PLP). Therefore, we also implemented it on the UltraScale XCKU060 FPGA and named it Vina-FPGA2-Enhanced. This implementation features two computational lanes and appropriate workload distribution. Moreover, it results in an average speedup of 12.6× and 3.3× compared to the CPU implementation and Vina-FPGA, respectively.

The rest of this paper is organized as follows. Section 2 introduces the background, related works, and motivation of this article. Section 3 provides the proposed inter-module pipeline architecture of Vina-FPGA2 based on Vina-FPGA, the formal description with reinforcement learning-based solver, and the implementation details of Vina-FPGA2-Baseline version and Vina-FPGA2-Enhanced version. Section 4 reports the experimental details and discusses the results. Section 5 presents the conclusion and looks forward to future work.

2 Background and motivations

2.1 AutoDock Vina

Molecular docking is used to predict the binding modes and affinities between small molecules and target proteins (Chaudhary and Mishra, 2016). The essence of molecular docking is finding the best conformation for the candidate drug molecule (i.e., the ligand) such that the energy or affinity between the ligand and the receptor (normally a target protein molecule) is the lowest (Fig. 1a). The conformation of the ligand is determined by three factors: *position* (P), *orientation* (O), and *torsion* (T). As shown in Fig. 1b, *position* represents the three-dimensional coordinates of the ligand, *orientation* represents the rotation angles of the ligand molecule itself in space, and *torsion* represents the rotation angles of the branch structures.

AutoDock Vina (Vina) (Trott and Olson, 2010) is an outstanding molecular docking tool that obtains the highest score in the latest test set CASF (Gaillard, 2018). Despite the emergence of ML/AI-based docking tools such as GNINA McNutt et al. (2021) and LeDock, AutoDock Vina still accounts for 13% of usage Muhammed and Aki-Yalcin (2024) and is widely adopted as a baseline in recent studies Vittorio et al. (2024); He et al. (2024); Carvajal-Patiño et al. (2025). Its computation involves an exhaustive conformational search and iterative refinement, resulting in significant runtime overhead. Thus, accelerating this process is of critical importance.

Vina uses an iterated local search global optimizer (ILSGO) to find the best ligand conformation, which consists of three nested loops, namely the global search, the local search, and the Armijo-Goldstein linear search, as depicted in Fig. 1c. For more details on the Vina algorithm, see the Section "The Algorithm Introduction for AutoDock-Vina" in Supplementary Materials.

2.2 Related works

The advancement of accelerators for molecular docking is a momentous subject in the realms of academia and industry, with a focus on three plat-

Unedited

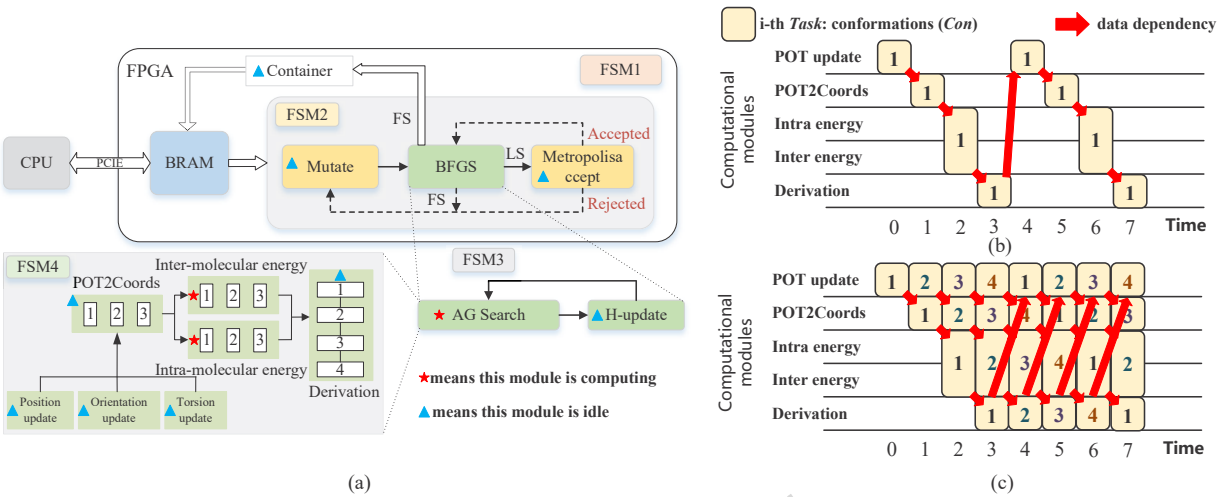

Fig. 2 (a) Hardware architecture of Vina-FPGA Ling et al. (2022). (b) The allocation of computational modules and *Task* in the Vina-FPGA design. (c) The ideal allocation of computational modules and Tasks. Note that the intervals for Time shown in the figure are not equal; they represent the unit time it takes for a module to complete its computations.

forms: CPU, GPU, and FPGA. The acceleration of molecular docking on CPU primarily concentrates on algorithmic enhancements. Handoko et al. (Handoko et al., 2012) presented QuickVina, which reduces the number of iterative searches by incorporating specific constraints. Alhossary et al. (Alhossary et al., 2015) and Hassan et al. (Hassan et al., 2017) proposed methodologies to further diminish these searches. VirtualFlow (Gorgulla et al., 2020) employed a multitude of 160,000 CPUs to significantly reduce Vina's computational time, albeit at the expense of resources and power consumption. GPU-based optimization encompasses Imre et al. (Pechan and Feher, 2011), who adapted AD4 for GPUs, Leonardo et al., who deployed AD4 on GTX260 (Solis-Vasquez and Koch, 2017) and Titan V (Solis-Vasquez et al., 2020) using OpenCL, who proposed the ADADELTA algorithm (Solis-Vasquez et al., 2020; Santos-Martins et al., 2021). We introduced Vina-GPU (Tang et al., 2022), which optimizes Vina by reducing search depth and increasing initial search points, relying on ample hardware resources. FPGA-based implementations feature Imre et al. (Pechan et al., 2010), developing an AD4 hardware accelerator using FPGA and pipeline design, and Leonardo et al., who employed OpenCL for FPGA deployment of AD4 (Solis-Vasquez and Koch, 2018). We presented Vina-FPGA (Ling et al., 2022), which achieves a 3.7× acceleration compared to CPUs through unified fixed-point quantization and pipeline design. Building on this foundation, we proposed Vina-FPGA-Cluster (Ling et al., 2024), which implements a multi-FPGA deployment of the Vina accelerator. Compared to Vina-GPU, this achieves a 1.38× performance improvement and a 4.76× energy efficiency enhancement.

2.3 Motivations

The Vina algorithm is an iterative computation involving data dependencies between computational modules, and most of these modules in Vina-FPGA remain idle during the computation process. Compared to neural network inference accelerators that can employ overlay computational module designs, the computation flow in Vina-FPGA is similar to the training process of neural networks. For example, as shown in Fig. 2a, on a coarse-grained level, the AG Search module is active while the Mutate, H-update, and Metropolis Accept modules are idle. On a fine-grained level, similarly, other sub-modules in the AG module remain idle while the inter-molecular and intra-molecular energy modules are computing. As introduced in the previous context, and profiled in Fig. 1d, the AG Search is the core computational module in the Vina algorithm. Additionally, Table 1 shows the module latency data for the Vina-FPGA design, indicating that numerous computational modules are idle throughout the Vina-FPGA computation process. Thus, enhancing the utilization of core computational modules in Vina-FPGA could further improve the efficiency of the accelera-

Unedited

Table 1 The operating cycle of each module of Vina-FPGA (Ling et al., 2022)

Computational modules		Cycles of computation
AG Search	POT update	43
	POT2Coords	$5\times N_{atom}+50\times N_{torsion}+2$
	Intra-molecular energy	$43+N_{pair-atom}$
	Inter-molecular energy	$47+N_{atom}$
	Derivation	$2+2\times N_{atom}+N_{torsion}\times(3+N_{torsion})$
H-update		190
Metropolis Accept		10
Mutate		47
Container		20

N_{atom}: number of atoms.
$N_{torsion}$: number of torsions.
$N_{pair-atom}$: number of active ligand atom pairs.

tor.

It is essential to recognize that Vina is a simulated annealing algorithm with multiple initial states. This means that the docking between ligands and receptors involves extensive computations for various initial conformations (*Cons*), with each *Cons* computation being entirely independent. Therefore, we regard the computational demands of each *Cons* as individual computational tasks (*Task*). According to the data in Table 1, the primary computational latency stems from the AG Search module. Consequently, as shown in Fig. 2b, the placement of computational modules and Tasks in the Vina-FPGA design is illustrated. Note that the intervals for Time shown in the figure are not equal, they represent the unit time it takes for a module to complete its computations. The ideal placement of computational modules and Tasks is shown in Fig. 2c. This setup ensures that during the Vina computation process, different *Cons* (*Task*s) sequentially utilize the idle computational modules based on their division. This strategy enables Exhaustiveness (task-level) parallelism on a single FPGA hardware computing core, significantly enhancing the efficiency and performance of the Vina-FPGA.

3 Methods

To allocate different tasks to the core computational modules and achieve an arrangement similar to what is shown in Fig. 2c, we introduced Vina-FPGA2. In Vina-FPGA2, we proposed a novel hardware architecture named Vina-FPGA2-Baseline to facilitate the inter-module pipeline design for various tasks. Moreover, we have developed a comprehensive model and a reinforcement learning-based solver to explore FPGA resource allocation. This allows us to determine the optimal number of computational lanes that can be implemented on a single FPGA, as well as the number of compatible tasks, thereby optimizing overall energy efficiency. Finally, targeting the UltraScale XCKU060, we implemented Vina-FPGA2-Enhanced, which is derived from the solver's results.

3.1 Vina-FPGA2-Baseline design

3.1.1 Vina-FPGA2-Baseline accelerator architecture

The primary difference between the inter-module pipeline design of Vina-FPGA2 and the traditional CPU pipeline design stems from the varying delays of different operations. This imbalance makes control and scheduling more challenging. The baseline architecture of our Vina-FPGA2 accelerator, as shown in Fig. 3a, eliminates the hierarchical control operations in Vina-FPGA and flattens all hardware computation modules to establish the inter-module pipeline. The connections between computation modules employ a backpressure mechanism. If the current-stage module is computing, the next-stage module, despite having completed its computation, will be blocked and unable to proceed. The original Vina-FPGA included four layers of state machines [AG layer (FSM4), BFGS layer (FSM3), ILSGO layer (FSM2), and the top layer (FSM1)], as shown in Fig. 2a, but only the top-level FSM1 is retained in Vina-FPGA2. Additionally, the original Metropolis Accept module has been removed. In its place, a Tag Checker module and a Metropolis Accept Checker module have been proposed. They will be introduced in the later paragraph. To prevent deadlock within the pipeline, we added two FIFO buffers to store the computed partial results. The first buffer is positioned between the Mutate module and the POT2Coords module, while the second buffer is placed before the POT update module. These specific designs have been carefully selected to ensure that the pipeline's endpoints (H-update and Metropolis Accept Checker) are not blocked. To be more specific, the size of FIFO is determined by the number of tasks. The bit-width of data associated with a single task, along with the corresponding LUT consumption. The locations of these two FIFOs correspond to the Vina iteration process output that also serves as an input for the next round. Although

Unedited

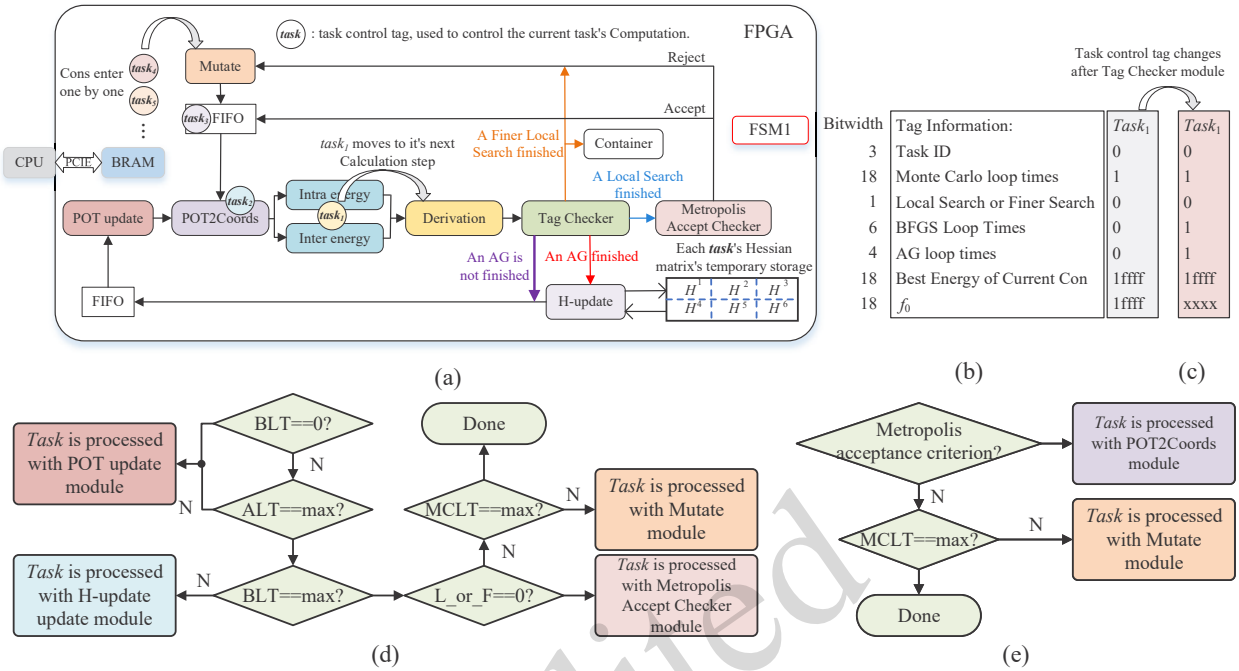

Fig. 3 (a) An inter-module pipelined architecture among multiple tasks driven by task tags, (b) Tag information, (c) demonstration of task tag's data changed subsequent to the Tag checker module, (d) the workflow of the Tag Checker module, (e) the workflow of the Metropolis Accept Checker module. *The workflow of the checker module. The maximum values (MAX) for MCLT, BLT, and ALT are $(N_{atom} + 110 + 10 \cdot N_{torsion}) \times 105$, and 10, respectively. ALT, AG loop times; BLT, BFGS loop times; MCLT, Monte Carlo loop times.

Unedited

the Vina pipeline is not fully balanced across different computation modules, data transfer between modules is controlled by the readiness of the next stage module. Therefore, when pipeline stalls occur, intermediate data can be temporarily buffered within individual modules. Moreover, it is worth emphasizing that the container module in Vina-FPGA extensively utilizes LUT resources for result caching, resulting in the container module consuming nearly half of the total hardware resources of Vina-FPGA. Consequently, in the design of Vina-FPGA2, we employed the LUTRAM resources in the container module to cache results, thereby liberating a substantial amount of LUT resources.

Tag Checker module: The Tag Checker module is composed of multiple task tables and muxes. The function of the Tag Checker is to guide the control flow of the current *Task*. Based on internal tags or control signals, it decides whether the task should: (1) continue processing within the AG module, (2) pass intermediate results to the H-update module, (3) invoke the Metropolis Accept Checker, or (4) route the task to the Mutate module to trigger another ILSGO cycle. The implementation of these functionalities primarily relies on the tag information, where each task is associated with its own independent tag information storing specific parameters, as shown in Fig. 3b. Here's a breakdown of how various entries in the tag information are utilized: Task ID: This serves as a key for retrieving corresponding values of the Hessian matrix from the temporary storage. Monte Carlo Loop Times (MCLT): This parameter is used to decide whether the execution should be terminated, as indicated in line 3 of Algorithm 1 in Supplementary Materials. It essentially checks if the number of iterations in the Monte Carlo loop has reached a predefined limit. Local Search or Finer Search (L_or_F): Determines whether the current BFGS operation is part of a local search (0 indicates local search) or a finer search (1 indicates finer search specifies a smaller search step size) impacting the precision and scope of the search process. BFGS Loop Times (BLT): Helps decide whether to terminate the BFGS loop for the current task, as specified in line 7 of Algorithm 2 in Supplementary Materials. This loop control is vital for managing the convergence of the BFGS optimization method. AG Loop Times (ALT): Used

to determine whether the AG loop for the current task should be terminated, as outlined in line 3 of Algorithm 3 in Supplementary Materials. This controls the iterations in the AG computation process. Best Energy of Current Con: This entry records the best energy value found for the current conformation. It is used to decide whether a finer search should be initiated, as per line 7 of Algorithm 1 in Supplementary Materials. *f0*: This global variable of the AG loop, calculated and updated by the BFGS layer, is recorded in the tag information. It represents an essential parameter within the optimization loop (referenced in line 10 of Algorithm 3 in Supplementary Materials).

The workflow of the Tag Checker module is shown in Fig. 3d. Upon the entry of tag information into the Tag Checker module, it first uses the BLT information from the task tag to determine whether the task should proceed with an AG search. If not, it continues to check the ALT information. Based on the information from ALT, the Tag Checker module decides whether the task should continue the AG search. If not, the BLT information is checked again. On this second inspection of BLT information, the Tag Checker decides whether to send the task to the H-update module for computation. If not, the module proceeds to check the L_or_F information. When the value of L_or_F is 0, the task is sent to the Metropolis Accept Checker module. Subsequently, the Tag Checker references the MCLT register to verify task progress. If the maximum number of ILSGO cycles has been reached, it terminates the associated tag. Otherwise, the task is dispatched to the Mutate module to continue the ILSGO workflow.

Metropolis Accept Checker module: The Metropolis Accept Checker module is modified from the Metropolis criterion modules within Vina-FPGA. The main difference is that the Metropolis Accept Checker takes not only the result from the BFGS calculation as input, but also includes intermediate results of the Task and tag information. The workflow of the Metropolis Accept Checker is shown in Fig. 3e. First, it performs the Metropolis acceptance criterion check. If the result is accepted, it sends the $Task$ information to the POT2Cords module and updates the value of L_or_F to 1. It determines whether the completed local search task can proceed to a fine-grained local search. If the result is not accepted, based on the information in the $Task$ Control Tag, it either sends the calculation task to the Mutate module or completely discards the task.

3.1.2 Computation flow of Vina-FPGA2-Baseline accelerator

The computation flow of the Vina-FPGA2-Baseline accelerator is shown in Fig. 3. First, the accelerator retrieves the initial $task1$ from BRAM and begins its computation. After completing the calculations within the Mutate module, the $task1$ passes through the FIFO into POT2Coords. At this point, the accelerator retrieves the $task2$ and starts its computation in the Mutate module. Upon completion of the molecular energy calculation, $task1$ proceeds to the next available stage, the Derivative module. Concurrently, $task2$ enters POT2Coords, $task3$ enters the Mutate module, and subsequent tasks follow this pattern. When $task1$ reaches the Tag Checker, it uses the tag information associated with $task1$ (illustrated by the purple arrow in Fig. 3a) to determine the next step. The Tag Checker then routes $task1$ to the FIFO preceding the POT update module. Simultaneously, the tag content of $task1$ is updated as shown in Fig. 3c, indicating that the task has completed the initial computation phase and entered the first AG loop of the local BFGS optimization, during which the value of *f0* is updated. According to the algorithmic flow, after *task1* completes the Mutate operation (Algorithm 1 in Supplementary Materials, line 5), it proceeds to the initial energy evaluation phase of the BFGS layer (Algorithm 2 in Supplementary Materials, lines 4-5). Following this, it transitions to the AG operation, specifically the POT update step (Algorithm 3 in Supplementary Materials, line 4). Meanwhile, the subsequent computational tasks are sequentially added to different computing modules for parallel computation, achieving parallelism among different Exhaustiveness. It is important to note that this control mode flattens the nested loops of the Vina algorithm, eliminating the need for separate control of the AG, BFGS, and ILSGO layers. The computation proceeds to the next step in order, without changing any steps of the Vina algorithm. This control method enables the construction of a pipelined architecture for independent computational tasks, making it an effective approach.

It is noteworthy that within this hardware computation pathway, there exist two data streams that

Unedited

Table 2 Notations used in Vina-FPGA2

Notation	Description
T	Set of $Tasks$
t	Index of $Tasks$, 1,2,...,$\|T\|$
a	Index of AGs, 1,2,...,A
WL	Constant; latency of $Task_t$ with one AG
EL	Variable; latency of $Task_T$ with N_A AGs
R_{AG}	Constant; FPGA resources used by AG
R_{Tag}	Constant; FPGA resources used by Tag checker
R	Constant; resource limitation in one FPGA
B_{AG}	Constant; BRAM resources used by AG
B	Constant; BRAM limitation in one FPGA
N_A	Variable; the number of AGs
$n_{t,a}$	Variable; the number of $Task_t$ allocated to AG_a
N_T	Variable; the number of $Task_T$ over all accelerator
P	Variable; the power consumption
L	Variable; the latency
PLP	Variable; the power latency product

are not visually represented by individual arrows. (1) The Mutate module receives data from both the Tag Checker module and the Metropolis Accept Checker module. (2) The POT2Coords module receives data from the Mutate, Tag Checker, and POT update modules. Consequently, both (1) and (2) have the potential to concurrently receive multiple computation requests from preceding modules. To address this matter, we have implemented a straightforward round-robin arbitration mechanism within the Mutate module and the POT2Coords module. When multiple computation requests are triggered simultaneously, each target module responds according to the Task ID. The remaining modules are stalled until they receive permission to proceed from either the Mutate or POT2Coords module.

3.2 Automatic optimization

Although Vina-FPGA2-Baseline proposes an inter-module pipeline design, determining the number of tasks based solely on the number of computation modules is inefficient, because the computation latency of different modules is not equal. Additionally, if sufficient hardware resources are available, a larger number of tasks and more AG computation lanes can be added to the system. Therefore, quickly enabling users to determine the optimal number of computation paths and tasks for the current hardware can further enhance efficiency and deployment performance. In this subsection, we introduced our optimization approach for Vina-FPGA2-Baseline and a Q-learning-based rapid solution method.

3.2.1 The formalization of the optimization problem

As shown in Table 2, we present abbreviations for our optimization descriptions. First, we defined T as the entire $Task$ set for Vina computation, and t corresponds to each $Task$. a is defined as the index of the AG computation lane. We then define WL as the worst latency of $Task_t$ in a single AG computation lane. Although there is some randomness in the Vina computation process, we made certain approximations to simplify the model description. Assuming the EL for all tasks on a single AG lane is given by:

$$EL = \lceil \frac{N_T}{n_{t,a}} \rceil \lceil \frac{n_{t,a}}{4} \rceil ((n_{t,a} - 1)\%4 + WL), \quad (1)$$

$\lceil \frac{N_T}{n_{t,a}} \rceil$ represents the ceiling function, indicating the total number of rounds required to allocate all tasks to a single lane. $\lceil \frac{n_{t,a}}{4} \rceil$ indicates the increase in the initiation interval when more than four tasks are added in a single round of computation. The $((n_{t,a} - 1)\%4 + WL)$ accounts for the AG pipeline depth of four and the additional computation delay WL caused by filling multiple tasks. Furthermore, we assumed that the total Vina computation time EL is proportional to the number of AG computation lanes and denote it as:

$$EL = \frac{\lceil \frac{N_T}{n_{t,a}} \rceil \lceil \frac{n_{t,a}}{4} \rceil ((n_{t,a} - 1)\%4 + WL)}{N_A} \quad (2)$$

Eq.(2) shows the optimal number of computation lanes N_A in the design goal. For all the AG lane, the number of tasks is represented as:

$$N_T = \sum_{a=1}^{A} n_{t,a}, \forall t \in T \quad (3)$$

For $Task$ allocation, since each computation of a $Task$ in Vina is independent, we adopt an intuitive dynamic load-balancing scheduling strategy. Specifically, an AG lane immediately retrieves the next set of tasks upon completing its current $Task$. If multiple AG lanes finish simultaneously, tasks are distributed in a round-robin manner.

For efficiency, we defined a power consumption calculation that correlates positively with resource usage. This includes the FPGA resources and BRAM consumption associated with increasing the

Unedited

number of AG lanes and tasks:

$$P = \gamma N_A(R_{AG} + \delta_R n_{t,a} \cdot R_{TAG}) + \delta_B \eta N_A B_{AG}. \tag{4}$$

In Eq.(4), γ and η refer to the resource power consumption weights, with their specific values derived from calculations using the XPE tool (AMD Xilinx, 2023). The δ_R represents the overhead that weight for the resources required when increasing the number of tasks assigned to a single AG lane. This is the core overhead that comes from R_{Tag} , which can be achieved by combining multiple LUTs and FFs. The δ_B represents the overhead weight for the resources required when increasing the number of AG.

Finally, we set the optimization goal to minimize the PLP (Hu et al., 2006), with weight values α and β, which are defined as Eq. 10. To achieve this computation, we established specific constraints to find the optimal values of N_A and $n_{t,a}$. We considered the limitations of FPGA computing resources (such as LUTs, FFs, DSPs, etc.) and on-chip memory (BRAM). Memory bandwidth is not considered because all computation data in Vina-FPGA are transmitted through BRAM.

$$\text{minimize } PLP = \alpha \cdot EL \times \beta \cdot P \tag{5}$$

subject to

$$N_A,\ n_{t,a} \geq 1,\ \forall t \in T \tag{6}$$

$$n_{t,a} \leq N_T,\ \forall t \in T \tag{7}$$

$$N_A(R_{AG} + \delta_R n_{t,a} \cdot R_{TAG}) \leq R,\ \forall t \in T \tag{8}$$

$$\delta_B N_A B_{AG} \leq B \tag{9}$$

Among the constraints mentioned above, Eq.(6) ensures that there is at least one lane and at least one *Task* in the design. Constraints Eq.(8) and Eq.(9) limit the use of FPGA resources and on-chip memory, respectively.

To get the minimal in Eq.(5), we need to find the best setting of N_A and $n_{t,a}$. We used reinforcement learning to solve this optimization problem. For a detailed solution procedure, see the Section "Reinforcement Learning Based Design Space Exploration" in Supplementary Material.

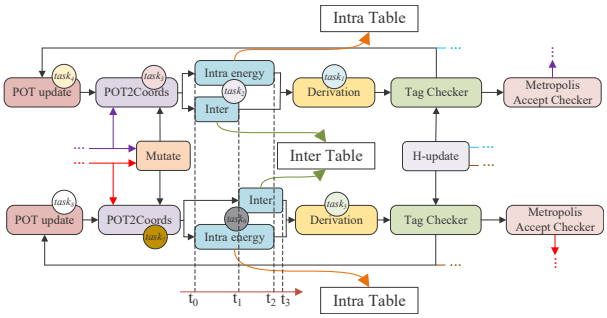

Fig. 4 The hardware implementation of Vina-FPGA2-Enhanced, two distinct AG computing lanes that share the Mutate and H-update modules and the two inter-molecular energy computing units share an inter-molecular energy table.

3.3 Vina-FPGA2-Enhanced

We set the target platform as UltraScale XCKU060, the same as Vina-FPGA. The solver determines that, under the resource constraints of this FPGA, the optimal PLP is achieved with two AG lanes and four tasks on each lane. We will introduce and discuss this result in Section 4. We deployed this configuration and named it Vina-FPGA2-Enhanced. First, we replicated a complete AG computation lane and set the maximum number of tasks per lane to four. Since the AG lanes do not include the Mutate and H-update modules, we allowed both AG lanes to share these two modules. Then, to implement two AG lanes while consuming only one BRAM, we made some access strategy adjustments, as shown in Fig. 4. By reducing the number of intermolecular energy tables to a maximum of four (it is normally the maximum number of intermolecular energy tables), we saved storage space. This saved space is used for the intramolecular energy tables required by the second lane. For the intermolecular energy tables, access is managed through polling arbitration between the two computation lanes. This design does not incur significant additional latency consumption, as intramolecular energy calculations typically take at least twice as long as intermolecular ones. As shown in Fig. 4, if the intramolecular and intermolecular calculations of the two energy computation units start simultaneously, they will both complete at time t2 and proceed to subsequent calculations. However, because the two energy computation units share one set of intermolecular tables, if the intramolecular runtime is not exactly twice that of the intermolecular runtime, there will be a slight increase in the computation module's total time (from t2 to t3, the

Unedited

Table 3 Resource utilization of Vina-FPGA2-Baseline.

	Mutate	H-update	AG lane[1]	Tag Checker	Metropolis Accept Checker	XDMA[2]	Container	Other	Total	Compare with Vina-FPGA
LUT	6,274	14,488	69,359	14,952	4,018	20,162	8,778	9,655	147,686 (44.5%)	**-111,900**
FF	5,074	15,342	77,615	12,420	6,137	19,625	5,466	8,354	150,033 (22.6%)	**-131,289**
DSP	36	219	193	2	1	0	63	0	514 (18.6%)	**+124**
BRAM	0	22	1,011	0	0	37	0	0	1,078 (99.8%)	**-2**

[1] Including POT update, POT2Coords, Inter-molecular energy, Intra-molecular energy, and Derivation.
[2] The DMA Subsystem for PCI Express IP (XDMA) provided by AMD (Xilinx) is a high-performance, configurable SG mode DMA suitable for PCIE2.0 and PCIE3.0 userselectable AXI4 interface or AXI4-Stream interface.

normally maximum number of cycles is 210). The performance of the final Vina-FPGA2-Enhanced design will be discussed in the next section.

4 Experiments and discussion

4.1 Experimental setup

4.1.1 Implementation details

We developed Vina-FPGA2 in Verilog HDL and synthesize the design using Xilinx Vivado 2020.2, which is deployed on Xilinx UltraScale XCKU060 customized board under 150MHz. The power (static and dynamic power of the FPGA chip) of Vina-FPGA2 is evaluated by Xilinx XPE (AMD Xilinx, 2023). The resource consumption of Vina-FPGA2-Baseline (set task=4) is shown in Table 3. The resource consumption of Vina-FPGA2-Enhanced will be discussed in Section 4.2.2. Additionally, to further demonstrate the scalability of our design, we evaluated Vina-FPGA2 on the Xilinx UltraScale VCU110 platform. For VCU110 evaluation, we implemented a cycle-accurate simulator Zeng et al. (2024), which has been verified with RTL emulation using Vivado 2020.2. Details of this evaluation are provided in Section 4.2.4.

4.1.2 Comparison platforms

As shown in Table 4, we compare our design with state-of-the-art implementations: CPU-only platform (Intel I7-12700KF, baseline), Vina-GPU (Nvidia RTX3090) (Tang et al., 2022), Vina-FPGA (Ling et al., 2022), and Vina-FPGA-Cluster (Single board) (Ling et al., 2024).

4.1.3 Benchmarks

Our benchmarking employs three representative molecular docking datasets that are consistent with Vina-FPGA: 85 complexes from the Astex Diversity Set (Hartshorn et al., 2007), 35 complexes from CASF-2013 (Su et al., 2018), and 20 complexes from the Protein Data Bank (Berman et al., 2000). These datasets represent a wide range of ligand complexities and target properties. Each complex file contains an X-ray structure, an initial random pose of its ligand, and the corresponding receptor. All formatted in PDBQT (Autodock structure) (Michaud-Agrawal et al., 2011; Wang et al., 2021).

4.1.4 Validation metrics

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. Docking results with RMSD less than < 2 Å are considered successful dockings (Goodsell et al., 2021).

4.1.5 Performance metrics

Latency: the total computation latency of Vina consists of the following parts: initialization, data transfer, and Vina main program computation (ILSGO). The initialization primarily involves analyzing the user's ligand/receptor files and parameters

Unedited

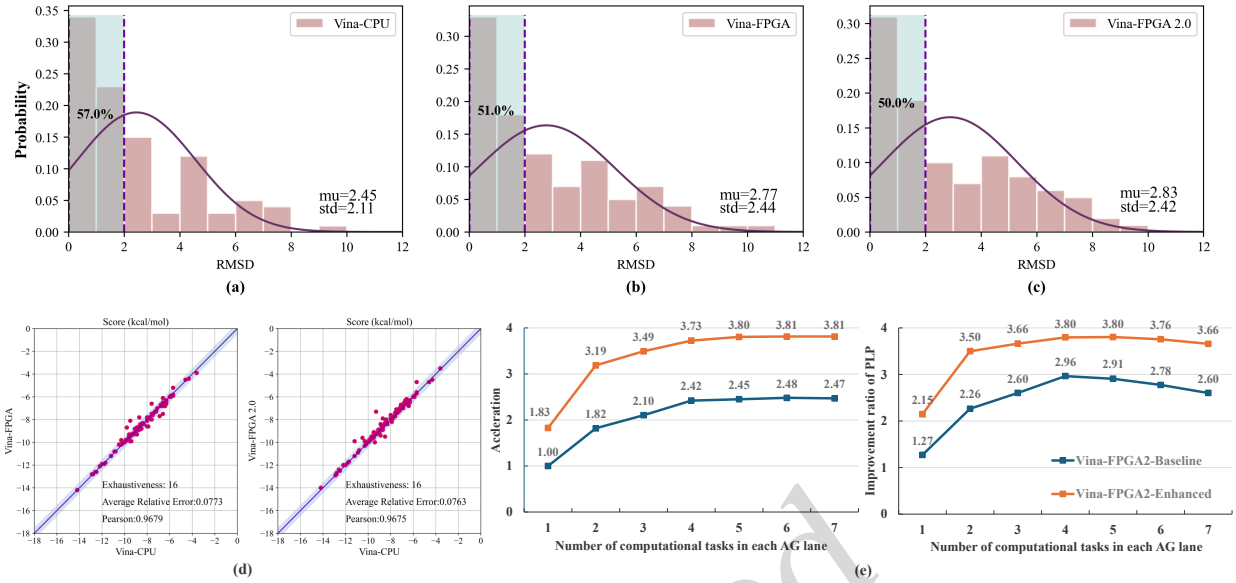

Fig. 5 (a)-(c) RMSD distribution of the dataset under different implementations. (d) Docking energy distribution comparisons of Vina-FPGA and Vina-FPGA2 under three representative molecular docking dataset. Vina-CPU is the original Vina that deployed on the Intel I7-12700KF CPU. The Pearson correlation coefficients of their docking scores are 0.9679 and 0.9675, respectively (indicated by "Pearson"). (e) The solver validation is compared to Vina-FPGA (ILSGO only) and measured by performance metrics. Specifically, the trends of acceleration and PLP improvement for Vina-FPGA2-Baseline and Vina-FPGA2-Enhanced (ILSGO only).

Unedited

to generate the data for computation. Data transfer involves sending the initialized data to the hardware accelerator via PCIe or Ethernet, and receiving the computation results. **Power Latency Production (PLP)**: we used the PLP as the energy efficiency performance metric for Vina-FPGA2. It should be noted that we use the core's power consumption as a multiplier since it is hard to obtain board-level power consumption data for the CPU. Therefore, the PLP is calculated as follows:

$$PLP = \text{Core's power consumption (W)} \times \text{Latency (S)}. \tag{10}$$

4.2 Vina-FPGA2 evaluation

We evaluated Vina-FPGA2 from the following aspects: first, validation, including comparisons of docking energy and RMSD. Then, we verified the effectiveness of the solver by comparing the actual experimental results with the solver's recommended results. Finally, we compared the performance of Vina-FPGA and Vina-FPGA2 (including Vina-FPGA2-Baseline and Vina-FPGA2-Enhanced) to verify the effectiveness of the hardware optimization.

4.2.1 Validation

We compared the docking results of Vina-FPGA2-Baseline with those of Vina-FPGA. It is important to note that due to the randomness of the ILSGO method, even if the initial points of Vina-FPGA and Vina-FPGA2 are identical, the computation results may differ. Moreover, to control the experimental variables, we only compared the computation results of Vina-FPGA and Vina-FPGA2. Excluding Vina-FPGA-Cluster, Vina-FPGA2 is a hardware architecture improvement for Vina-FPGA. Fig. 5d shows the energy differences between Vina-FPGA and Vina-FPGA2 compared to the original Vina computation when the exhaustiveness is set to 16. The dots in the figure represent the energy output for a given ligand and receptor complex, with the x and y coordinates representing the output results of the hardware accelerator and the CPU (the original Vina), respectively. The average relative error of Vina-FPGA compared to Vina-CPU is 0.0773, while that of Vina-FPGA2 is 0.0763. It can be observed that the energy differences between the two and the CPU results are minimal, without significant fluctuations. This indicates that the architectural modifications in Vina-FPGA2 do not affect its computa-

Table 4 Performance and energy comparison of CPU, Vina-GPU, Vina-FPGA, Vina-FPGA-cluster (single board), and Vina-FPGA2

Evaluation metric of performance		CPU	Vina-GPU (Tang et al., 2022)	Vina-FPGA (Ling et al., 2022)	Vina-FPGA-Cluster (single board) (Ling et al., 2024)	Vina-FPGA2-Enhanced
Platform	Processor	I7-12700KF	RTX3090	UltraScale XCKU060	UltraScale+ ZU7EV	UltraScale XCKU060
	Frequency	3600MHz	1700MHz	150MHz	200MHz	150MHz
	Technology	Intel 7nm	TSMC 7nm	TSMC 20nm	TSMC 14nm	TSMC 20nm
	Bit-width	FP32	FP32	18-bit fixed	18-bit fixed	18-bit fixed
	Memory (On-chip)	25MB (L3 cache)	6MB (L2 cache)	4.75MB (BRAM only)	4.75MB (BRAM+URAM)	4.75MB (BRAM only)
Latency (s)	Initialization	1.734	-	1.903	1.903	1.903
	Data Transfer	-	-	0.156	0.446	0.156
	ILSGO	180.55	7.27	46.34	11.88	12.44
	(Speed up)	(1×)	(24.8×)	(3.9×)	(15.2×)	(14.5×)
	Total	182.28	9.20	48.40	14.23	14.50
	(Speed up)	(1×)	(19.8×)	(3.7×)	(12.8×)	(12.6×)
Core Power (W)		47.34	67.2	4.7	4.9	4.7
PLP (J) (Improvement)		8547.09 (1×)	488.54 (17.5×)	217.81 (39.2×)	69.73 (122.6×)	68.15 (125.4×)

tion results. Furthermore, we compare the RMSD distribution of the original Vina, Vina-FPGA, and Vina-FPGA2 under the same dataset and settings, as shown in Fig. 5c. It shows that the docking success probabilities of Vina-FPGA 2 are comparable to those of Vina-FPGA, with both exhibiting similar trends in probability distribution. The differences between Vina-FPGA and Vina-FPGA2 compared to the original Vina are due to the quantization used in both FPGA implementations. Thus, we have verified the computational correctness of Vina-FPGA2 and confirmed that the architecture design in this paper does not affect the computational results.

For solver validation, we deployed a single AG lane with different maximum task numbers and two AG lanes (the maximum resource limit of the XCKU060 FPGA) with different maximum task numbers on the XCKU060 FPGA. Each deployment runs 10 times on the dataset, and we used the average ILGSO computation delay as the comparison metric. Power consumption values come from the XPE tool, and the baseline is the fixed ILGSO computation delay of Vina-FPGA. As shown in Fig. 5e, we illustrated the acceleration and PLP improvement relative to Vina-FPGA for both single AG lane and two AG lanes. It can be observed that as the number of tasks increases, the acceleration and PLP improvement ratios continuously increase. When the maximum number of tasks exceeds four, the acceleration ratio stabilizes, while PLP begins to decline. This result is consistent with the solver's recommended configuration, demonstrating the effectiveness of our proposed solver.

4.2.2 Performance improvement

The performance improvement is divided into two parts: one is the performance enhancement of Vina-FPGA2-Baseline compared to Vina-FPGA, and the other is Vina-FPGA2-Enhanced compared to Vina-FPGA2-Baseline. Combining the comparison of resource consumption (Table 3) with the performance comparison (Fig. 5e), although Vina-FPGA2-Baseline has lower total resource consumption it achieves a 2.42× speedup in ILGSO compared to Vina-FPGA. The resource benefit comes from Vina-FPGA2 replacing the LUTs in the container module of Vina-FPGA with LUTRAM. For Vina-FPGA2-Enhanced, it consumes an average of 46% more of the main resources (LUT, FF, and DSP, as shown in Fig. 6) compared to Vina-FPGA2-Baseline but achieves a 55% performance improvement. These results demonstrate the effectiveness of Vina-FPGA2's performance improvements.

4.2.3 Cross-platform comparison

We conduct a cross-platform performance comparison of Vina-FPGA2-Enhanced deployment results, comparing latency (including initialization, data transfer, ILGSO computation, and total time), power consumption, and PLP, shown in Table 4. The platforms compared are: A CPU-only platform (Intel I7-12700KF), Vina-GPU (Nvidia RTX 3090) (Tang et al., 2022), Vina-FPGA (Kintex UltraScale XCKU060) (Ling et al., 2022), and Vina-FPGA-Cluster (ZCU104, single board for a fair comparison) (Ling et al., 2024). All platforms use the

Unedited

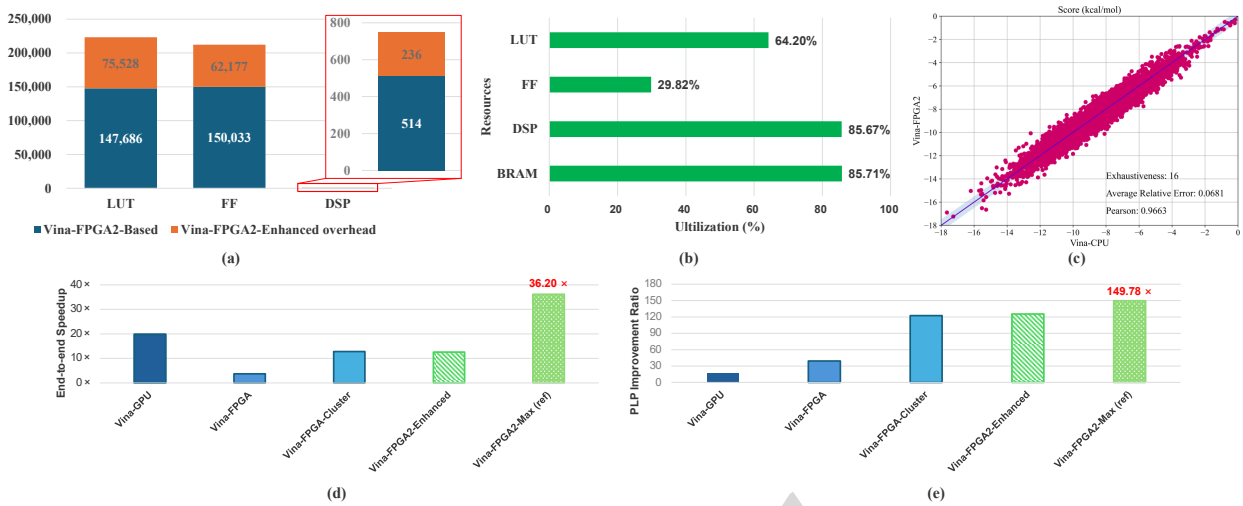

Fig. 6 (a) The main resource consumption comparison of whole Vina-FPGA2-Basedline and Vina-FPGA2-Enhanced. The overhead of Vina-FPGA2-Enhanced is highlighted by orange. (b) the resource utilization of Vina-FPGA2-Max (ref) on VCU110. The available resources on the VCU110 include 1 M LUTs, 2.1 M FFs, 1.8 K DSP slices, and 3.7 K BRAM blocks. (c) Docking scores of Vina-FPGA2 and Vina-CPU on the PDBbind v2020 dataset, evaluated across 5,316 complexes. The Pearson correlation coefficient of its docking scores is 0.9663 (indicated by "Pearson"). (d) End-to-end speedup of Vina on different platforms relative to the CPU baseline. (e) PLP improvement ratio of Vina on different platforms relative to the CPU baseline.

same preset docking pocket location and size, and the same initial random seeds. In terms of initialization, the three FPGA-based accelerators use the same software for data initialization, resulting in consistent latencies. Vina-FPGA-Cluster uses an Ethernet port for communication, which results in the longest data transfer time. In the core ILGSO computation phase of Vina, Vina-FPGA2-Enhanced achieves a 14.5× speedup compared to the CPU and a 3.7× speedup compared to Vina-FPGA. The difference between Vina-FPGA2-Enhanced and Vina-FPGA-Cluster is only about 0.6 s. It is important to note that Vina-FPGA-Cluster runs at a frequency of 200 MHz, while Vina-FPGA2-Enhanced runs at 150 MHz. The latency difference between Vina-FPGA2-Enhanced and Vina-GPU mainly arises because Vina-GPU effectively reduces the computational workload of the original Vina algorithm by approximately twofold through algorithmic modifications. Regarding overall latency, Vina-FPGA2-Enhanced provides a 12.6× improvement over the CPU and a 3.3× improvement over Vina-FPGA. In terms of power consumption, Vina-FPGA2-Enhanced and Vina-FPGA have the lowest power usage, consuming only 6.9% of the GPUs'. Although Vina-FPGA-Cluster consumes fewer resources than these two, the presence of the ARM CPU results in higher power consumption. Regarding energy efficiency, Vina-FPGA2-Enhanced exhibits the best performance, achieving a 125.4× improvement over the CPU and a 7.2× and 3.2× improvement over the GPU and Vina-FPGA, respectively. Compared to Vina-FPGA-Cluster, the energy of Vina-FPGA2-Enhanced shows almost no difference. Additionally, it is important to emphasize that the optimizations of Vina-FPGA2 and Vina-FPGA-Cluster are orthogonal and non-conflicting, meaning that the optimizations from Vina-FPGA-Cluster can be combined with the architecture of Vina-FPGA2.

4.2.4 Scalable evaluation

To further demonstrate the scalability of our design, we evaluated Vina-FPGA2 on the VCU110 platform and referred to this configuration as Vina-FPGA2-Max. We observed that on the VCU110 board, the design supports up to six lanes. The deployment is primarily constrained by BRAM availability. The linear growth in LUT and FF usage corresponds to the modular and independent design of each Vina lane. The DSP utilization appears high, but this is due to the limited number of DSP slices available on the VCU110 (1800 in total).

With the additional BRAM capacity available on the VCU110, we extended our evaluation of

Unedited

Vina-FPGA2 to the PDBbind v2020 dataset, which includes 5,316 molecular complexes. The results, shown in Fig. 6(c), exhibit a similar accuracy trend to Fig. 5(d). The Pearson correlation coefficient is evaluated using the original Vina (Vina-CPU) as the baseline, because this work aims to accelerate the docking process without altering Vina's scoring function or algorithmic logic, in line with our previous studies Tang et al. (2022); Ding et al. (2023).

Fig. 6d and Fig. 6e show the end-to-end speedup and PLP improvement of Vina-FPGA2-Max. With a higher number of deployed lanes, Vina-FPGA2-Max achieves a 36.2× speedup over CPU and 1.82× over GPU. Compared to the 2-lane Vina-FPGA2-Extended, it reaches a 2.87× acceleration. This sublinear scaling is due to variability introduced by Vina's stochastic search, which causes non-uniform execution latency across lanes. In terms of PLP, Vina-FPGA2-Max achieves a 149.78× improvement over the CPU, but only a 1.19× gain compared to Vina-FPGA2-Enhanced. This aligns with the trend observed in the solver results shown in Fig. 5e, indicating that simply increasing the number of tasks or lanes does not lead to linear improvements in PLP.

5 Conclusions and future work

In this paper, we present Vina-FPGA2, an FPGA-based molecular docking acceleration tool. Building upon our previous efforts, Vina-FPGA2 implements an inter-module pipelined design, further accelerating the Vina computation process. Additionally, we developed a reinforcement learning-based rapid solver that allows users to quickly obtain deployment parameters tailored to their target FPGA. Experimental results demonstrate that Vina-FPGA2-Enhanced achieves an average 12.6× performance improvement over the CPU and a 3.3× improvement over Vina-FPGA. Compared to Vina-GPU, Vina-FPGA2 achieves a 7.2× enhancement in energy efficiency. In terms of energy efficiency, Vina-FPGA2 achieves a 125.4× improvement compared to the CPU deployment and a 3.2× improvement compared to Vina-FPGA. In contrast to the most advanced GPU (Vina-GPU) deployments, our system consumes a mere 6.9% of the energy while delivering over 65% of the performance.

In the context of Vina-FPGA2-Enhanced, future endeavors could contemplate alternative approaches for accessing molecular energy tables to fully cater to the access requirements of multiple AG lanes. This could involve network-on-chip interconnects to mitigate access conflicts, thereby resolving the issue of insufficient BRAM in Vina-FPGA2-Enhanced. Combining this approach with the optimization methods of Vina-FPGA-Cluster to achieve even better performance is an ongoing work. Additionally, hardware acceleration for emerging deep neural network–based molecular docking algorithms (e.g., DiffusionDock (Corso et al. (2023))) is also an ongoing area of work.

Unedited

Contributors

Ming LING designed the research. Shidi Tang, Ruiqi CHEN, and Xin LI processed the data. Ming LING, Ruiqi CHEN, and Xin LI drafted the paper. Shidi Tang and Yanxiang ZHU helped organize the paper. Ming LING, Shidi Tang, and Ruiqi CHEN finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Acknowledgements

The authors would like to thank VeriMake Innovation Lab for providing developing and testing equipment. This work was supported by the OPPO research fund.

References

Alhossary A, Handoko SD, Mu Y, et al., 2015. Fast, accurate, and reliable molecular docking with quickvina 2. *Bioinformatics*, 31(13):2214-2216. https://doi.org/10.1093/bioinformatics/btv082

AMD, 2023. Xilinx Power Estimator (XPE). https://www.xilinx.com/products/technology/power/xpe.html

Belletti F, Cotallo M, Cruz A, et al., 2008. Janus: An fpga-based system for high-performance scientific computing. Computing in Science & Engineering, 11(1):48-58. https://doi.org/10.1109/MCSE.2009.11

Berman HM, Westbrook J, Feng Z, et al., 2000. The protein data bank. Nucleic acids research, 28(1):235-242. https://doi.org/10.1093/nar/28.1.235

Caballero J, 2021. The latest automated docking technologies for novel drug discovery. Expert opinion on drug discovery, 16(6):625-645. https://doi.org/10.1080/17460441.2021.1858793

Carvajal-Patiño JG, Mallet V, Becerra D, et al., 2025. Rnamigos2: accelerated structure-based rna virtual screening with deep graph learning. Nature Communications, 16(1):1-12. https://doi.org/10.1038/s41467-025-57852-0

Chaudhary KK, Mishra N, 2016. A review on molecular docking: novel tool for drug discovery. Databases, 3(4):1029. https://doi.org/10.47739/2334-1831/1029

Chen YC, 2015. Beware of docking! Trends in pharmacological sciences, 36(2):78-95. https://doi.org/10.1016/j.tips.2014.12.001

Choi YK, Cong J, Fang Z, et al., 2019. In-depth analysis on microarchitectures of modern heterogeneous cpu-fpga platforms. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 12(1):1-20. https://doi.org/10.1145/3294054

Corso G, StÃ H, Jing B, et al., 2023. Diffdock: Diffusion steps, twists, and turns for molecular docking. International Conference on Learning Representations (ICLR 2023), p.1-12.

Ding J, Tang S, Mei Z, et al., 2023. Vina-gpu 2.0: further accelerating autodock vina and its derivatives with graphics processing units. Journal of chemical information and modeling, 63(7):1982-1998. https://doi.org/0.1021/acs.jcim.2c01504

Gaillard T, 2018. Evaluation of autodock and autodock vina on the casf-2013 benchmark. Journal of chemical information and modeling, 58(8):1697-1706. https://doi.org/10.1021/acs.jcim.8b00312

Goodsell DS, Sanner MF, Olson AJ, et al., 2021. The autodock suite at 30. Protein Science, 30(1):31-43. https://doi.org/10.1002/pro.3934

Gorgulla C, Boeszoermenyi A, Wang ZF, et al., 2020. An open-source drug discovery platform enables ultra-large virtual screens. Nature, 580(7805):663-668. https://doi.org/10.1038/s41586-020-2117-z

Handoko SD, Ouyang X, Su CTT, et al., 2012. Quickvina: accelerating autodock vina using gradient-based heuristics for global optimization. IEEE/ACM transactions on computational biology and bioinformatics, 9(5):1266-1272. https://doi.org/10.1109/TCBB.2012.82

Hartshorn MJ, Verdonk ML, Chessari G, et al., 2007. Diverse, high-quality test set for the validation of protein-ligand docking performance. Journal of medicinal chemistry, 50(4):726-741. https://doi.org/10.1021/jm061277y

Hassan NM, Alhossary AA, Mu Y, et al., 2017. Protein-ligand blind docking using quickvina-w with inter-process spatio-temporal integration. Scientific reports, 7(1):15451. https://doi.org/10.1038/s41598-017-15571-7

He X, Zhao L, Tian Y, et al., 2024. Highly accurate carbohydrate-binding site prediction with deepglycansite. Nature Communications, 15(1):5163. https://doi.org/10.1038/s41467-024-49516-2

Hu Y, Zhu Y, Chen H, et al., 2006. Communication latency aware low power noc synthesis. Proceedings of the 43rd annual Design Automation Conference (DAC), New York, NY, USA, p.574–579. https://doi.org/10.1145/1146909.1147058

Kirkpatrick S, Gelatt Jr CD, Vecchi MP, 1983. Optimization by simulated annealing. *science*, 220(4598):671-680. https://doi.org/10.1126/science.220.4598.671

Ling M, Lin Q, Chen R, et al., 2022. 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, 31(4):484-497. https://doi.org/10.1109/TVLSI.2022.3217275

Ling M, Feng Z, Chen R, et al., 2024. Vina-fpga-cluster: Multi-fpga based molecular docking tool with high-accuracy and multi-level parallelism. IEEE Transactions on Biomedical Circuits and Systems, 18(6):1321-1337. https://doi.org/10.1109/TBCAS.2024.3388323

McNutt AT, Francoeur P, Aggarwal R, et al., 2021. Gnina 1.0: molecular docking with deep learning. Journal of cheminformatics, 13(1):43. https://doi.org/10.1186/s13321-021-00522-2

Michaud-Agrawal N, Denning EJ, Woolf TB, et al., 2011. Mdanalysis: a toolkit for the analysis of molecular dynamics simulations. Journal of computational chemistry, 32(10):2319-2327. https://doi.org/10.1002/jcc.21787

Mittal S, 2020. A survey of fpga-based accelerators for convolutional neural networks. Neural computing and applications, 32(4):1109-1139. https://doi.org/10.1007/s00521-018-3761-1

Muhammed MT, Aki-Yalcin E, 2024. Molecular docking: Principles, advances, and its applications in drug discovery. Letters in Drug Design & Discovery, 21(3):480-495. https://doi.org/10.2174/1570180819666220922103109

Pechan I, Feher B, 2011. Molecular docking on fpga and gpu platforms. 2011 21st International Conference on Field Programmable Logic and Applications (FPL), p.474-477. https://doi.org/10.1109/FPL.2011.93

Pechan I, Fehér B, Bérces A, 2010. Fpga-based acceleration of the autodock molecular docking software. 6th Conference on Ph D Research in Microelectronics & Electronics, p.1-4.

Salmaso V, Moro S, 2018. Bridging molecular docking to molecular dynamics in exploring ligand-protein recognition process: An overview. Frontiers in pharmacology, 9:923. https://doi.org/10.3389/fphar.2018.00923

Santos-Martins D, Solis-Vasquez L, Tillack AF, et al., 2021. Accelerating autodock4 with gpus and gradient-based local search. Journal of chemical theory and computation, 17(2):1060-1073. https://doi.org/10.1021/acs.jctc.0c01006

Shawahna A, Sait SM, El-Maleh A, 2018. Fpga-based accelerators of deep learning networks for learning and classification: A review. ieee Access, 7:7823-7859. https://doi.org/10.1109/ACCESS.2018.2890150

Solis-Vasquez L, Koch A, 2017. A performance and energy evaluation of opencl-accelerated molecular docking. Proceedings of the 5th International Workshop on OpenCL, New York, NY, USA, p.1-11. https://doi.org/10.1145/3078155.3078167

Solis-Vasquez L, Koch A, 2018. A case study in using opencl on fpgas: Creating an open-source accelerator of the autodock molecular docking software. FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers, p.1-10.

Solis-Vasquez L, Santos-Martins D, Tillack AF, et al., 2020. Parallelizing irregular computations for molecular docking. 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3), p.12-21. https://doi.org/10.1109/IA351965.2020.00008

Uncorrected

Su M, Yang Q, Du Y, et al., 2018. Comparative assessment of scoring functions: the casf-2016 update. *Journal of chemical information and modeling*, 59(2):895-913. https://doi.org/10.1021/acs.jcim.8b00545

Tang S, Chen R, Lin M, et al., 2022. Accelerating autodock vina with gpus. *Molecules*, 27(9):3041. https://doi.org/10.3390/molecules27093041

Trott O, Olson AJ, 2010. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. Journal of computational chemistry, 31(2):455-461. https://doi.org/10.1002/jcc.21334

Vittorio S, Lunghini F, Morerio P, et al., 2024. Addressing docking pose selection with structure-based deep learning: Recent advances, challenges and opportunities. Computational and Structural Biotechnology Journal, 23:2141-2151. https://doi.org/10.1016/j.csbj.2024.05.024

, 2021. MDAnalysis User Guide's documentation. https://userguide.mdanalysis.org/1.0.1/formats/reference/pdbqt. html#pdbqt-autodock-structure

Zeng S, Liu J, Dai G, et al., 2024. Flightllm: Efficient large language model inference with a complete mapping flow on fpgas. Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, p.223-234. https://doi.org/10.1145/3626202.3637562

Zhou X, Ling M, Lin Q, et al., 2023. 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, 20(6):3830-3841. https://doi.org/10.1109/TCBB.2023.3323552

Unedited