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]*School of Integrated Circuits, 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 Oct. 31, 2025; Published online Dec. 3, 2025

**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 field-programmable gate array (FPGA)-based accelerator of Vina, known as Vina-FPGA, offers a high energy-efficiency approach to speed up the docking process. However, the computation modules in the Vina-FPGA design are not efficiently used. 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. 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 use individual computational task (Task) independence by sequentially filling Tasks into computation modules. Then, we implement 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 describe it as an optimization problem and develop 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 central processing unit (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 (Vina); Hardware accelerator; Field-programmable gate array; Software/hardware co-design

https://doi.org/10.1631/FITEE.2400941 **CLC number:** TP332.1

## 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 speed up the process of searching for optimal results in a huge potential solution space (Salmaso and

‡ Corresponding authors

* Project supported by the National Natural Science Foundation of China (No. 92464301) and the Big Data Computing Center of Southeast University

# Electronic supplementary materials: The online version of this article (https://doi.org/10.1631/FITEE.2400941) contains supplementary materials, which are available to authorized users

ORCID: Ming LING, https://orcid.org/0000-0002-8866-7189; Ruiqi CHEN, https://orcid.org/0000-0001-6837-5675

Moro, 2018). Diverse molecular docking tools adopt distinct search algorithms, and AutoDock Vina (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 initial states is independent, allowing Vina to deploy the calculations of various initial states on multi-core central processing units (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-GPU 2.0 (Ding et al., 2023), integrating QuickVina 2 (Alhossary et al., 2015) and QuickVina-W (Hassan et al., 2017), exploit graphics processing unit (GPU) parallelism to achieve up to 21× speedups (Zhou et al., 2023). While these methods significantly enhance performance, they also incur high energy costs. Field-programmable gate arrays (FPGAs), with their reconfigurability, parallelism, and energy efficiency (Belletti et al., 2009; Choi et al., 2019; Shawahna et al., 2019; Mittal, 2020), 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 3.8× 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. Using 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 propose Vina-FPGA2, a high-level parallelized hardware-accelerated molecular docking tool. First, we implement 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 describe Vina-FPGA2-Baseline to further improve deployment efficiency under FPGA resource constraints (LUT, FF, BRAM, etc.). Here, LUT is short for lookup table, FF is short for flip-flop, and BRAM is short for block random access memory. Then, we transform the entire problem into an optimization problem to minimize the power-delay product. To solve this problem, we design a reinforcement learning-based solver for a rapid and automatic solution. Finally, under the constraints of deploying on the UltraScale XCKU060, we develop the Vina-FPGA2-Enhanced design. Our main contributions are as follows:

1. An innovative inter-module pipelined parallel architecture. We propose 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 multi-tasking computation flow of Vina. Based on these modules and some architectural adjustments, we develop 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 resource overhead while under the same processing frequency (150 MHz).

2. A formula description and reinforcement learning-based solver. Building on Vina-FPGA2-Baseline, we formulate its design

to minimize the power-delay product and add several constraints. To rapidly and automatically determine the optimal number of computation lanes and individual computational task (Task) deployments for a specific FPGA, we design a reinforcement learning-based solver.

3. A resource-efficient hardware module deployment strategy for Vina-FPGA2-Enhanced. Benefiting from the solver, we derive the recommended implementation strategy to achieve optimal power latency production (PLP). Therefore, we implement it on the UltraScale XCKU060 FPGA and name 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.

## 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 comparative assessment of scoring function (CASF) (Gaillard, 2018). Despite the emergence of machine learning/artificial intelligence (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 (He et al., 2024; Vittorio 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 (AG) linear search, as depicted in Fig. 1c. For more details on the Vina algorithm, see the algorithm introduction for AutoDock-Vina in the 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 platforms: CPU, GPU, and FPGA. The acceleration of molecular docking on CPU primarily concentrates

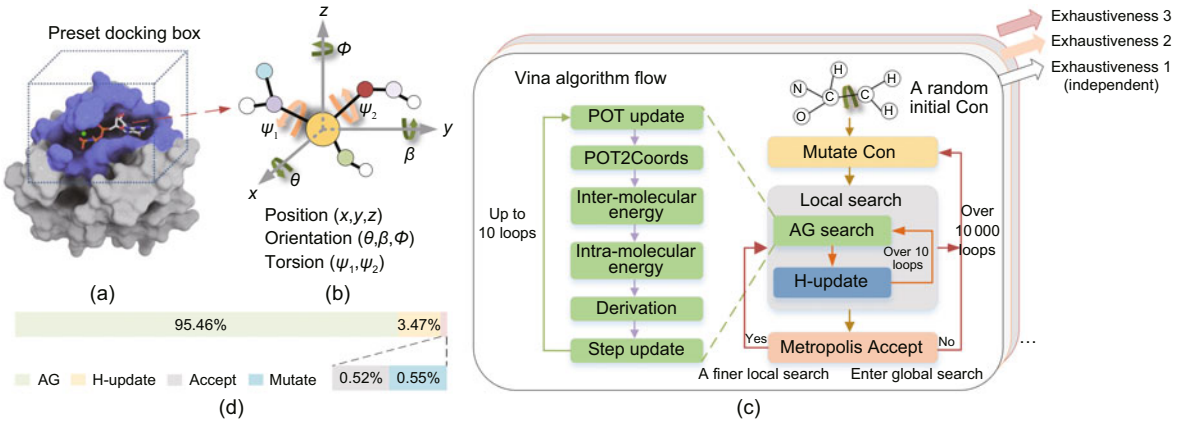


**Fig. 1 Schematic diagram of a ligand-receptor docking (a), illustration of the conformation of a ligand molecule (b), the algorithm flow of Vina (c), and the profiling of AG, H-update, Accept, and Mutate, respectively (d)**

on algorithmic enhancements. Handoko et al. (2012) presented QuickVina, which reduces the number of iterative searches by incorporating specific constraints. Alhossary et al. (2015) and Hassan et al. (2017) proposed methodologies to further diminish these searches. VirtualFlow (Gorgulla et al., 2020) employs 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 includes the work of Pechan and Feher (2011), who adapted AD4 for GPUs, and Solis-Vasquez and Koch (2017), who deployed AD4 on GTX260 and later on Titan V using OpenCL and introducing the ADADELTA algorithm (Solis-Vasquez et al., 2020; Santos-Martins et al., 2021). Vina-GPU (Tang et al., 2022) enhances Vina by reducing search depth and increasing initial search points, leveraging abundant hardware resources. In the domain of FPGA-based implementations, Pechan et al. (2010) developed an AD4 hardware accelerator using FPGA and pipeline design, while Solis-Vasquez and Koch (2018) employed OpenCL to deploy AD4 on FPGAs. Another notable implementation, Vina-FPGA (Ling et al., 2022), achieves a 3.8× speedup over 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, Vina-FPGA-Cluster 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 submodules 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 accelerator.

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 Con computation being entirely independent. Therefore,

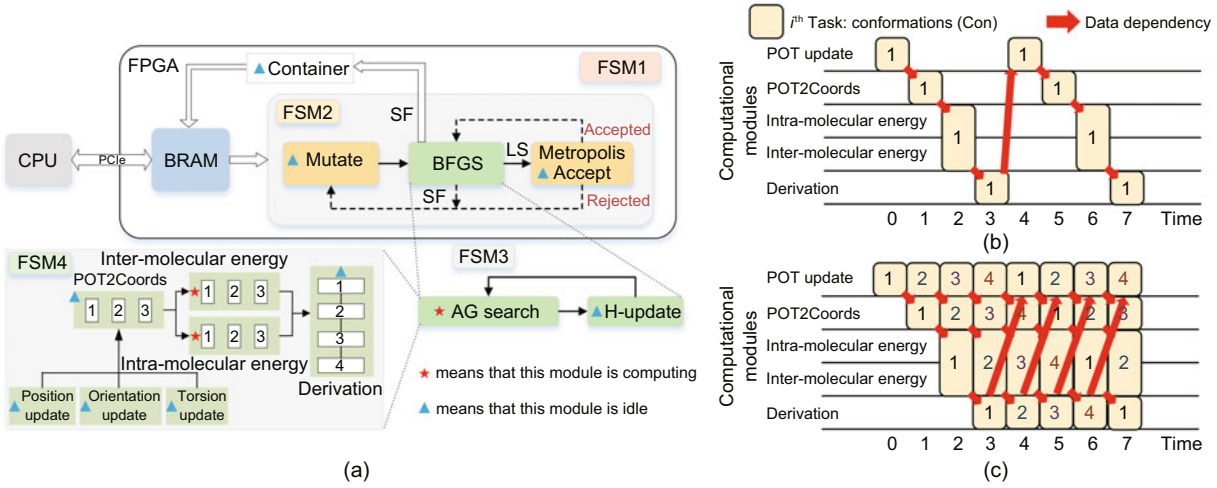


**Fig. 2 Hardware architecture of Vina-FPGA (Ling et al., 2022) (a), the allocation of computational modules and Task in the Vina-FPGA design (b), the ideal allocation of computational modules and Tasks (c). SF: scoring function; LS: local search; PCIe: peripheral component interconnect express**

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

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

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

we regard the computational demands of each Con as 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 (Tasks) sequentially use 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 introduce Vina-FPGA2. In Vina-FPGA2, we propose a novel hardware architecture named Vina-FPGA2-Baseline to facilitate the inter-module pipeline design for various Tasks. Moreover, we develop 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 implement 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 includes 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. Here, FSM is short for finite state machine. 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 add two first-in-first-out (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 position orientation torsion (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 is 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 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

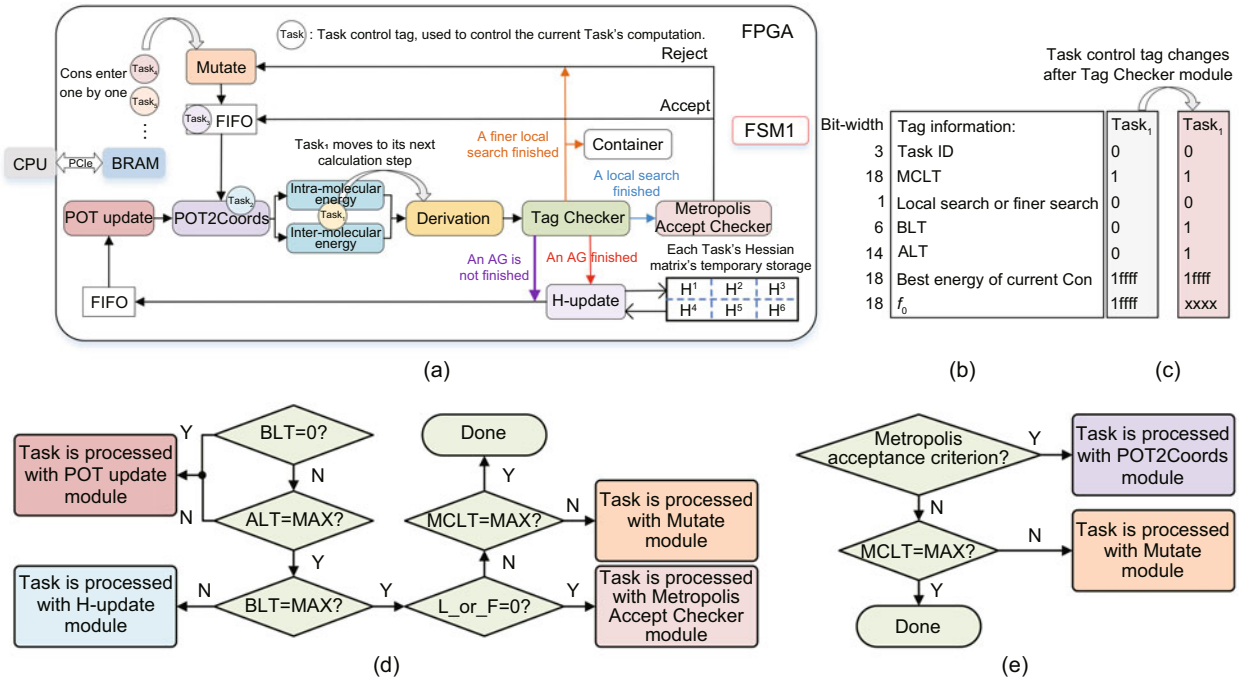


**Fig. 3 (a) An inter-module pipelined architecture among multiple Tasks driven by Task tags; (b) Tag information; (c) demonstration of Task tag's data changed subsequent to the Tag Checker module; (d) the workflow of the Tag Checker module; (e) the workflow of the Metropolis Accept Checker module. The maximum values (MAX) for MCLT, BLT, and ALT are $(N_{\text{atom}}+110+10\cdot N_{\text{torsion}})\cdot 105$, 10, and $(25+N_{\text{atom}})/3$, respectively. ALT: AG loop times; BLT: Broyden–Fletcher–Goldfarb–Shanno (BFGS) loop times; MCLT: Monte Carlo loop times. References to color refer to the online version of this figure**

is worth emphasizing that the container module in Vina-FPGA extensively uses 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 employ the lookup table random access memory (LUTRAM) resources in the container module to cache results, thereby liberating a substantial amount of LUT resources.

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 used.

1. Task ID: This serves as a key for retrieving corresponding values of the Hessian matrix from the temporary storage.

2. Monte Carlo loop times (MCLT): This parameter is used to decide whether the execution should be terminated, as indicated in line 3 of Algorithm S1 in the supplementary materials. It essentially checks if the number of iterations in the Monte Carlo loop has reached a predefined limit.

3. Local search or finer search (L_or_F): This determines whether the current BFGS operation is part of a local search (0 indicates local search) or a finer search (1 indicates finer search that specifies a smaller search step size) impacting the precision and scope of the search process.

4. BFGS loop times (BLT): This helps decide whether to terminate the BFGS loop for the current Task, as specified in line 7 of Algorithm S2 in the supplementary materials. This loop control is vital for managing the convergence of the BFGS optimization method.

5. AG loop times (ALT): This is used to

determine whether the AG loop for the current Task should be terminated, as outlined in line 3 of Algorithm S3 in the supplementary materials. This controls the iterations in the AG computation process.

6. 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 S1 in the supplementary materials.

7. $f_0$: 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 S3 in the 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 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 POT2Coords 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 $\text{Task}_1$ from BRAM and begins its computation. After completing the calculations within the Mutate module, $\text{Task}_1$ passes through the FIFO into POT2Coords. At this point, the accelerator retrieves $\text{Task}_2$ and starts its computation in the Mutate module. Upon completion of the molecular energy calculation, $\text{Task}_1$ proceeds to the next available stage, the Derivation module. Concurrently, $\text{Task}_2$ enters POT2Coords, $\text{Task}_3$ enters the Mutate module, and subsequent Tasks follow this pattern. When $\text{Task}_1$ reaches the Tag Checker, it uses the tag information associated with $\text{Task}_1$ (illustrated by the purple arrow in Fig. 3a) to determine the next step. The Tag Checker then routes $\text{Task}_1$ to the FIFO preceding the POT update module. Simultaneously, the tag content of $\text{Task}_1$ 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 $f_0$ is updated. According to the algorithmic flow, after $\text{Task}_1$ completes the Mutate operation (Algorithm S1 in the supplementary materials, line 5), it proceeds to the initial energy evaluation phase of the BFGS layer (Algorithm S2 in the supplementary materials, lines 4–5). Following this, it transitions to the AG operation, specifically the POT update step (Algorithm S3 in the supplementary materials, line 4). Meanwhile, the subsequent 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 Tasks,

making it an effective approach.

It is noteworthy that within this hardware computation pathway, there exist two data streams that 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 introduce our formalization of the optimization approach for Vina-FPGA2-Baseline and a Q-learning-based rapid solution method.

As shown in Table 2, we present abbreviations for our optimization descriptions. First, we define $T$ 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 $\text{Task}_t$ in a single AG computation lane. Although there is some randomness in the Vina computation process, we make certain approximations to simplify the model description. Assuming the EL for all Tasks on a single AG lane is given by

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

**Table 2 Notations used in Vina-FPGA2**

| Notation | Description |
|---|---|
| $T$ | Set of Tasks |
| $t$ | Index of Tasks, 1, 2, ..., $\lvert T\rvert$ |
| $a$ | Index of AGs, 1, 2, ..., $A$ |
| WL | Constant; latency of $\text{Task}_t$ with one AG |
| EL | Variable; latency of $\text{Task}_T$ with $N_\text{A}$ AGs |
| $R_\text{AG}$ | Constant; FPGA resources used by AG |
| $R_\text{Tag}$ | Constant; FPGA resources used by Tag Checker |
| $R$ | Constant; resource limitation in one FPGA |
| $B_\text{AG}$ | Constant; BRAM resources used by AG |
| $B$ | Constant; BRAM limitation in one FPGA |
| $N_\text{A}$ | Variable; the number of AGs |
| $n_{t,a}$ | Variable; the number of $\text{Task}_t$ allocated to $\text{AG}_a$ |
| $N_T$ | Variable; the number of $\text{Task}_T$ over all accelerators |
| $P$ | Variable; the power consumption |
| $L$ | Variable; the latency |
| PLP | Variable; the power latency product |

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

$$\text{EL} = \frac{\left\lceil \frac{N_T}{n_{t,a}} \right\rceil \left\lceil \frac{n_{t,a}}{4} \right\rceil ((n_{t,a}-1)\%4 + \text{WL})}{N_\text{A}}. \quad (2)$$

Eq. (2) shows the optimal number of computation lanes $N_\text{A}$ in the design goal. For all the AG lanes, 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 define a power consumption calculation that correlates positively with resource usage. This includes the FPGA resources and BRAM consumption associated with increasing the

number of AG lanes and Tasks:

$$P = \gamma N_{\mathrm{A}}(R_{\mathrm{AG}} + \delta_{\mathrm{R}} n_{t,a} R_{\mathrm{Tag}}) + \delta_{\mathrm{B}} \eta N_{\mathrm{A}} B_{\mathrm{AG}}. \quad (4)$$

In Eq. (4), $\gamma$ and $\eta$ refer to the resource power consumption weights, with their specific values derived from calculations using the Xilinx power estimator (XPE) tool (Xilinx, 2023). The $\delta_{\mathrm{R}}$ represents the overhead that weights for the resources required when increasing the number of Tasks assigned to a single AG lane. This is the core overhead that comes from $R_{\mathrm{Tag}}$, which can be achieved by combining multiple LUTs and FFs. The $\delta_{\mathrm{B}}$ represents the overhead that weights for the resources required when increasing the number of AG lanes.

Finally, we set the optimization goal to minimize the PLP (Hu et al., 2006), with weight values $\alpha$ and $\beta$, which are defined as Eq. (5). To achieve this computation, we establish specific constraints to find the optimal values of $N_{\mathrm{A}}$ and $n_{t,a}$. We consider the limitations of FPGA computing resources (such as LUTs, FFs, and digital signal processors (DSPs)) and on-chip memory (BRAM). Memory bandwidth is not considered because all computation data in Vina-FPGA are transmitted through BRAM.

$$\mathrm{PLP} = \text{Core's power consumption} \cdot \text{Latency}. \quad (5)$$

$$\text{minimize PLP} = \alpha \mathrm{EL} \cdot \beta \cdot P \quad (6)$$

subject to

$$N_{\mathrm{A}},\ n_{t,a} \geq 1,\ \forall t \in T, \quad (7)$$

$$n_{t,a} \leq N_T,\ \forall t \in T, \quad (8)$$

$$N_{\mathrm{A}}(R_{\mathrm{AG}} + \delta_{\mathrm{R}} n_{t,a} \cdot R_{\mathrm{Tag}}) \leq R,\ \forall t \in T, \quad (9)$$

$$\delta_{\mathrm{B}} N_{\mathrm{A}} B_{\mathrm{AG}} \leq B. \quad (10)$$

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

To obtain the minimum in Eq. (5), we need to find the best setting of $N_{\mathrm{A}}$ and $n_{t,a}$. We use reinforcement learning to solve this optimization problem. For a detailed solution procedure, see the reinforcement learning based design space exploration in the supplementary materials.

### 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 deploy this configuration and name it Vina-FPGA2-Enhanced. First, we replicate 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 allow both AG lanes to share these two modules. Then, to implement two AG lanes while consuming only one BRAM, we make some access strategy adjustments, as shown in Fig. 4. By reducing the number of inter-molecular energy tables to a maximum of four (it is normally the maximum number of inter-molecular energy tables), we save storage space. This saved space is used for the intra-molecular energy tables required by the second lane. For the inter-molecular energy tables, access is managed through polling arbitration between the two computation lanes. This design does not incur significant additional latency consumption, as intra-molecular energy calculations typically take at least twice as long as inter-molecular ones. As shown in Fig. 4, if the intra-molecular and inter-molecular calculations of the two energy computation units start simultaneously, they will both complete at time $t_2$ and proceed to subsequent calculations. However, because the two energy computation units share one set of inter-molecular tables, if the intra-molecular runtime is not exactly twice that of the inter-molecular runtime, there will be

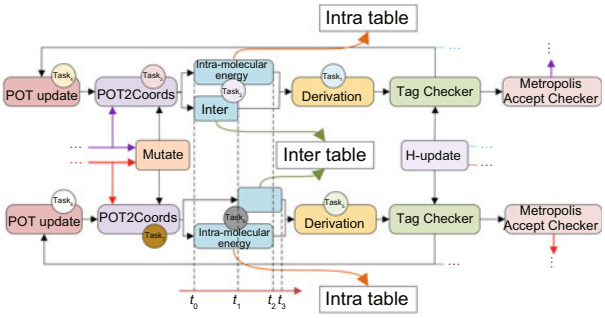


**Fig. 4 Hardware implementation of Vina-FPGA2-Enhanced: two distinct AG computing lanes sharing the Mutate and H-update modules and the two inter-molecular energy computing units sharing an inter-molecular energy table**

a slight increase in the computation module's total time (from $t_2$ to $t_3$, the normally maximum number of cycles is 210). The performance of the final Vina-FPGA2-Enhanced design is discussed in the next section.

## 4 Experiments and discussions

### 4.1 Experimental setup

#### 4.1.1 Implementation details

We develop Vina-FPGA2 in Verilog HDL and synthesize the design using Xilinx Vivado 2020.2, which is deployed on Xilinx UltraScale XCKU060 customized board under 150 MHz. The power (static and dynamic power of the FPGA chip) of Vina-FPGA2 is evaluated by XPE (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 evaluate Vina-FPGA2 on the Xilinx UltraScale VCU110 platform. For VCU110 evaluation, we implement a cycle-accurate simulator (Zeng et al., 2024), which has been verified with register-transfer level (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 the files are formatted in PDBQT (AutoDock structure) (Michaud-Agrawal et al., 2011).

#### 4.1.4 Validation metrics

1. 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

**Table 3 Resource utilization of Vina-FPGA2-Baseline**

| Type | Resource utilization | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Mutate | H-update | AG lane[1] | Tag Checker | Metropolis Accept Checker | XDMA[2] | Container | Other | Total | Compared with Vina-FPGA |
| LUT | 6274 | 14 488 | 69 359 | 14 952 | 4018 | 20 162 | 8778 | 9655 | 147 686 (44.5%) | −111 900 |
| FF | 5074 | 15 342 | 77 615 | 12 420 | 6137 | 19 625 | 5466 | 8354 | 150 033 (22.6%) | −131 289 |
| DSP | 36 | 219 | 193 | 2 | 1 | 0 | 63 | 0 | 514 (18.6%) | +124 |
| BRAM | 0 | 22 | 1011 | 0 | 0 | 37 | 0 | 0 | 1070 (99.1%) | −10 |

[1] Including POT update, POT2Coords, inter-molecular energy, intra-molecular energy, and derivation;
[2] The direct memory access (DMA) subsystem for PCI Express IP (XDMA) provided by AMD (Xilinx) is a high-performance, configurable scatter–gather (SG) mode DMA suitable for PCIe2.0 and PCIe3.0 userselectable AXI4 interface or AXI4-Stream interface

**Table 4 Platform specification comparison of Vina-CPU, Vina-GPU, Vina-FPGA, Vina-FPGA-cluster (single board), and Vina-FPGA2**

| Model | Platform specification | | | | |
|---|---|---|---|---|---|
| | Processor | Frequency | Technology | Bit-width | Memory (on-chip) |
| Vina-CPU (Trott and Olson, 2010) | I7-12700KF | 3600 MHz | Intel 7 nm | FP32 | 25 MB (L3 cache) |
| Vina-GPU (Tang et al., 2022) | RTX3090 | 1700 MHz | TSMC 7 nm | FP32 | 6 MB (L2 cache) |
| Vina-FPGA (Ling et al., 2022) | UltraScale XCKU060 | 150 MHz | TSMC 20 nm | 18-bit fixed | 4.75 MB (BRAM only) |
| Vina-FPGA-Cluster (Ling et al., 2024) (single board) | UltraScale+ ZUTEV | 200 MHz | TSMC 14 nm | 18-bit fixed | 4.75 MB (BRAM+URAM) |
| Vina-FPGA2-Enhanced | UltraScale XCKU060 | 150 MHz | TSMC 20 nm | 18-bit fixed | 4.75 MB (BRAM only) |

docking energy (numerically more negative) typically signifies a stronger docking affinity.

2. Root-mean-square deviation (RMSD). The docking results are typically validated using the output energy between the ligand and receptor, along with the RMSD between the output ligand conformation and the X-ray measurements. RMSD serves as the ground truth. Docking results with RMSD <2 Å (1 Å $= 1 \times 10^{-10}$ m) are considered successful dockings (Goodsell et al., 2021).

#### 4.1.5 Performance metrics

1. Latency. 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 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.

2. PLP. We use 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 shown in Eq. (5)

### 4.2 Vina-FPGA2 evaluation

We evaluate Vina-FPGA2 from the following aspects: first, validation, including comparisons of docking energy and RMSD. Then, we verify the effectiveness of the solver by comparing the actual experimental results with the solver's recommended results. Finally, we compare 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 compare 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 compare 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 (Vina-CPU) 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 accelerators. 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 computation results. Furthermore, we compare the RMSD distribution of the Vina-CPU, Vina-FPGA, and Vina-FPGA2 under the same dataset and settings, as shown in Figs. 5a–5c. It shows that the docking success probabilities of Vina-FPGA2 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 Vina-CPU 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 deploy 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 use the average ILSGO computation delay as the comparison metric. Power consumption values come from the XPE tool, and the baseline is the fixed ILSGO computation delay of Vina-FPGA. As shown in Fig. 5e, we illustrate 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.

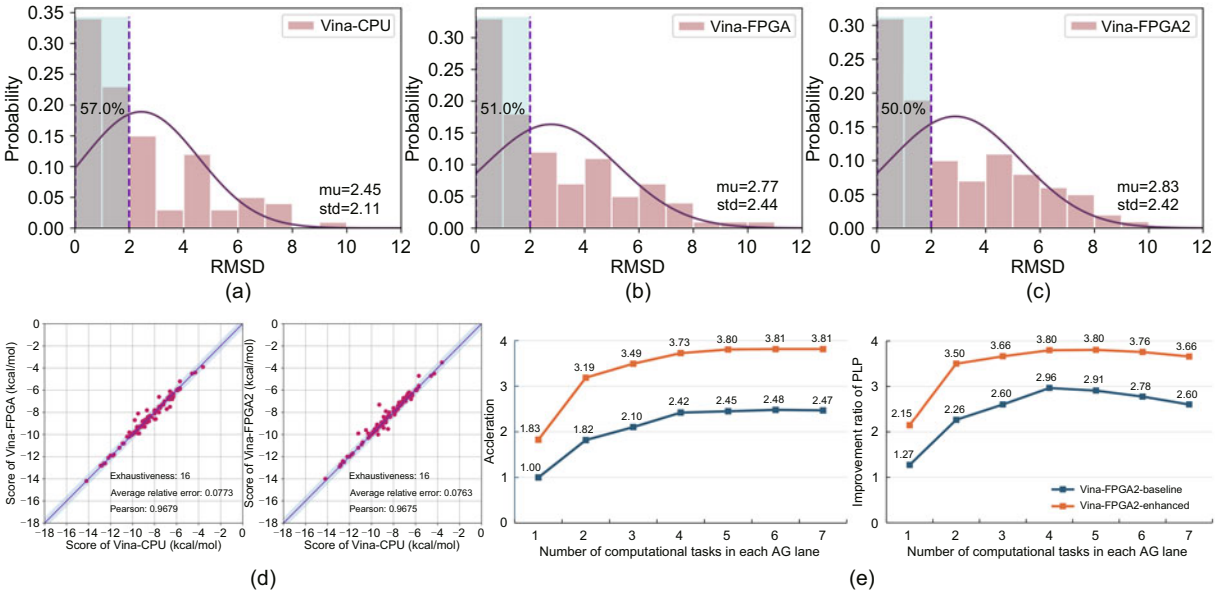


**Fig. 5 (a)–(c) RMSD distribution of the dataset under different implementations. (d) Docking energy distribution comparisons of Vina-FPGA and Vina-FPGA2 under three representative molecular docking dataset. Vina-CPU is deployed on the Intel I7-12700KF CPU. The Pearson correlation coefficients of their 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, this shows the trends of acceleration and PLP improvement for Vina-FPGA2-Baseline and Vina-FPGA2-Enhanced (ILSGO only). mu: mean; std: standard deviation**

### 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 ILSGO 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, ILSGO computation, and total time), power consumption, and PLP, shown in Tables 5 and 6. The platforms compared are a CPU-only platform (Intel I7-12700KF), Vina-GPU (NVIDIA RTX3090) (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 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 ILSGO computation phase of Vina, Vina-FPGA2-Enhanced achieves a 14.5× speedup compared to the CPU and a 3.8× 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

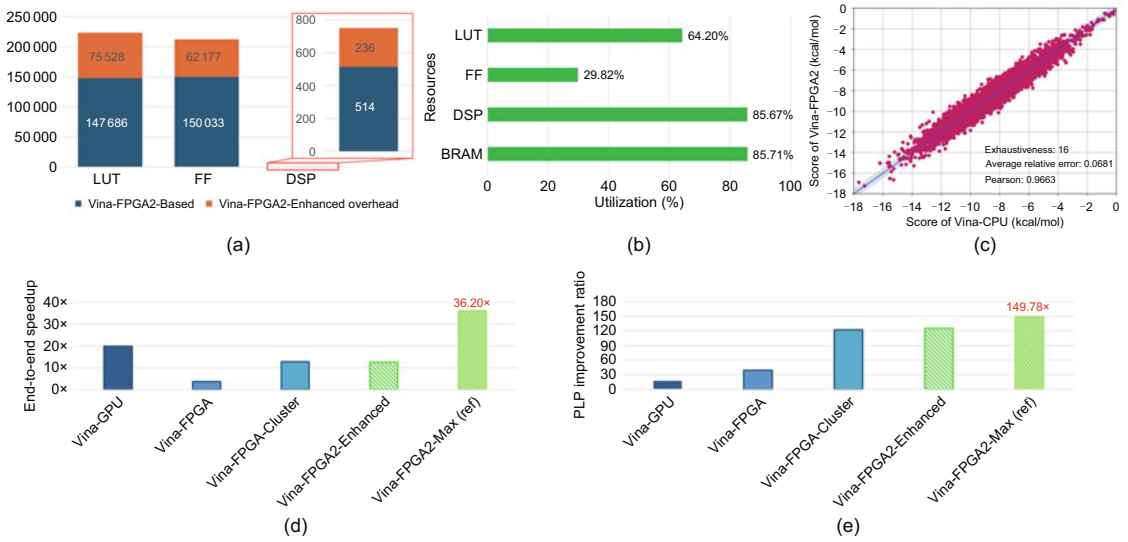


**Fig. 6 (a) The main resource consumption comparison of whole Vina-FPGA2-Baseline and Vina-FPGA2-Enhanced. The overhead of Vina-FPGA2-Enhanced is highlighted by orange. (b) The resource utilization of Vina-FPGA2-Max (ref) on VCU110. The available resources on the VCU110 include about $1 \times 10^6$ LUTs, about $2.1 \times 10^6$ FFs, about 1800 DSP slices, and about 3700 BRAM blocks. (c) Docking scores of Vina-FPGA2 and Vina-CPU on the PDBbind v2020 dataset, evaluated across 5316 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. References to color refer to the online version of this figure**

**Table 5 Latency comparison of CPU, Vina-GPU, Vina-FPGA, Vina-FPGA-Cluster (single board), and Vina-FPGA2**

| Model | Evaluation metric of performance | | | |
|---|---|---|---|---|
| | Initialization | Data transfer | ILSGO (Speedup) | Total (Speedup) |
| Vina-CPU (Trott and Olson, 2010) | 1.734 | | 180.55 (1×) | 182.28 (1×) |
| Vina-GPU (Tang et al., 2022) | | | 7.27 (24.8×) | 9.20 (19.8×) |
| Vina-FPGA (Ling et al., 2022) | 1.903 | 0.156 | 46.34 (3.9×) | 48.40 (3.8×) |
| Vina-FPGA-Cluster (Ling et al., 2024) (single board) | 1.903 | 0.446 | 11.88 (15.2×) | 14.23(12.8×) |
| Vina-FPGA2-Enhanced | 1.903 | 0.156 | 12.44 (14.5×) | 14.5 (12.6×) |

**Table 6 Power and energy consumption comparison of CPU, Vina-GPU, Vina-FPGA, Vina-FPGA-Cluster (single board), and Vina-FPGA2**

| Model | Evaluation metric of performance | |
|---|---|---|
| | Core power (W) | PLP (J) (Improvement) |
| Vina-CPU (Trott and Olson, 2010) | 47.34 | 8547.09 (1×) |
| Vina-GPU (Tang et al., 2022) | 67.2 | 488.54 (17.5×) |
| Vina-FPGA (Ling et al., 2022) | 4.7 | 217.81 (39.2×) |
| Vina-FPGA-Cluster (Ling et al., 2024) (single board) | 4.9 | 69.73 (122.6×) |
| Vina-FPGA2-Enhanced | 4.7 | 68.15 (125.4×) |

and Vina-GPU mainly arises because Vina-GPU effectively reduces the computational workload of Vina-CPU 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 Acorn RISC machine (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 evaluate Vina-FPGA2 on the VCU110 platform and refer to this configuration as Vina-FPGA2-Max. We observe 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 extend our evaluation of Vina-FPGA2 to the PDBbind v2020 dataset, which includes 5316 molecular complexes. The results, shown in Fig. 6c, exhibit a similar accuracy trend to Fig. 5d. The Pearson correlation coefficient is evaluated using 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).

Figs. 6d and 6e show the end-to-end speedup and PLP improvement of Vina-FPGA2-Max, respectively. With a higher number of deployed lanes, Vina-FPGA2-Max achieves a 36.20× 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 develop 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., DiffDock (Corso et al., 2023)) is also an ongoing area of work.

### Contributors

Xin LI and Ruiqi CHEN drafted the paper. Ming LING, Shidi TANG, Ruiqi CHEN, and Yanxiang ZHU revised the paper. Ruiqi CHEN and Shidi TANG finalized the paper.

### Conflict of interest

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

### Data availability

The data that support the findings of this study are available from the corresponding authors upon reasonable request.

## References


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

Belletti F, Cotallo M, Cruz A, et al., 2009. Janus: an FPGA-based system for high-performance scientific computing. *Comput Sci Eng*, 11(1):48-58. https://doi.org/10.1109/MCSE.2009.11

Berman HM, Westbrook J, Feng ZK, et al., 2000. The Protein Data Bank. *Nucleic Acids Res*, 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 Opin Drug Discov*, 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. *Nat Commun*, 16(1):2799. https://doi.org/10.1038/s41467-025-57852-0

Chaudhary KK, Mishra N, 2016. A review on molecular docking: novel tool for drug discovery. *JSM Chem*, 4(43):1029. https://doi.org/10.47739/2334-1831/1029

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

Choi YK, Cong J, Fang ZM, et al., 2019. In-depth analysis on microarchitectures of modern heterogeneous CPU-FPGA platforms. *ACM Trans Reconfig Technol Syst*, 12(1):4. https://doi.org/10.1145/3294054

Corso G, Stärk H, Jing BW, et al., 2023. DiffDock: diffusion steps, twists, and turns for molecular docking. Proc 11th Int Conf on Learning Representations, p.1-12.

Ding J, Tang SD, Mei ZM, et al., 2023. Vina-GPU 2.0: further accelerating AutoDock Vina and its derivatives with graphics processing units. *J Chem Inform Model*, 63(7):1982-1998. https://doi.org/10.1021/acs.jcim.2c01504

Gaillard T, 2018. Evaluation of AutoDock and AutoDock Vina on the CASF-2013 benchmark. *J Chem Inform Model*, 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 Sci*, 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 Trans Comput Biol Bioinform*, 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. *J Med Chem*, 50(4):726-741. https://doi.org/10.1021/jm061277y

Hassan NM, Alhossary AA, Mu YG, et al., 2017. Protein-ligand blind docking using QuickVina-W with inter-process spatio-temporal integration. *Sci Rep*, 7(1):15451. https://doi.org/10.1038/s41598-017-15571-7

He XH, Zhao LF, Tian YP, et al., 2024. Highly accurate carbohydrate-binding site prediction with DeepGlycanSite. *Nat Commun*, 15(1):5163. https://doi.org/10.1038/s41467-024-49516-2

Hu YF, Zhu Y, Chen HY, et al., 2006. Communication latency aware low power NoC synthesis. Proc 43rd Annual Design Automation Conf, 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 QD, Chen RQ, et al., 2022. Vina-FPGA: a hardware-accelerated molecular docking tool with fixed-point quantization and low-level parallelism. *IEEE Trans Very Large Scale Integr Syst*, 31(4):484-497. https://doi.org/10.1109/TVLSI.2022.3217275

Ling M, Feng ZH, Chen RQ, et al., 2024. Vina-FPGA-cluster: multi-FPGA based molecular docking tool with high-accuracy and multi-level parallelism. *IEEE Trans Biomed Circ Syst*, 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. *J Cheminform*, 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. *J Comput Chem*, 32(10):2319-2327. https://doi.org/10.1002/jcc.21787

Mittal S, 2020. A survey of FPGA-based accelerators for convolutional neural networks. *Neur Comput Appl*, 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. *Lett Drug Des Discov*, 21(3):480-495. https://doi.org/10.2174/1570180819666220922103109

Pechan I, Feher B, 2011. Molecular docking on FPGA and GPU platforms. Proc 21st Int Conf on Field Programmable Logic and Applications, 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. Proc 6th Conf 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. *Front Pharmacol*, 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. *J Chem Theory Comput*, 17(2):1060-1073. https://doi.org/10.1021/acs.jctc.0c01006

Shawahna A, Sait SM, El-Maleh A, 2019. 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. Proc 5th Int Workshop on OpenCL, 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. Proc 5th Int 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. Proc 10th IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3), p.12-21. https://doi.org/10.1109/IA351965.2020.00008

Su M, Yang Q, Du Y, et al., 2018. Comparative assessment of scoring functions: the CASF-2016 update. *J Chem Inform Model*, 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. *J Comput Chem*, 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. *Comput Struct Biotechnol J*, 23:2141-2151. https://doi.org/10.1016/j.csbj.2024.05.024

Xilinx, 2023. Xilinx Power Estimator (XPE). https://www.xilinx.com/products/technology/power/xpe.html [Accessed on Oct. 22, 2024].

Zeng S, Liu J, Dai G, et al., 2024. FlightLLM: efficient large language model inference with a complete mapping flow on FPGAs. Proc ACM/SIGDA Int Symp 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 Trans Comput Biol Bioinform*, 20(6):3830-3841. https://doi.org/10.1109/TCBB.2023.3323552

## List of supplementary materials