VinaSC: Scalable Autodock Vina with Fine-grained Scheduling on Heterogeneous Platform

Lang Yu*‡2, Zhongzhi Luan*, Xiangzheng Sun†, Zhe Wang†, Hailong Yang*1
*Sino-German Joint Software Institute, Beihang University, Beijing, China
†Software & Service Group, Intel China Ltd., Beijing, China
‡Department of Computer Science, University of Chicago, USA
{lang.yu, 07680, hailong.yang}@buaa.edu.cn, {xiangzheng.sun, zhe.wang}@intel.com

***Abstract*—In this paper we present VinaSC, an improved version of Autodock Vina, that performs molecular docking simulation efficiently on large-scale heterogeneous cluster for massive docking scenario. Both application and platform optimizations are implemented to fully exploit performance potentials of heterogeneous platforms. Specifically, computation is offloaded to Intel Many Integrated Core (MIC) using Intel Coprocessor Offload Infrastructure (COI) to make host CPU and coprocessor collaborate during docking simulation. Moreover, a dynamic scheduling framework is implemented in VinaSC using MPI and Pthread to leverage heterogeneous resources. Our work makes the following improvements: 1) Compared to original Vina that only supports single-node CPU platform, VinaSC fully utilizes computing resources including CPU and MIC coprocessor. 2) Load unbalance due to the random algorithm and heterogeneous platform is alleviated. 3) Utilization of vector units on MIC is significantly improved. 4) VinaSC scales well on heterogeneous cluster, which enables mass docking using clusters. Experiments on a cluster with 6 CPU+MIC nodes using PDBBIND dataset demonstrate that VinaSC outperforms original Vina by more than 2.3x. In addition, VinaSC maintains scalable performance speedup as the docking scale increases.**

***Index Terms*—molecular docking, many core architecture, fine-grained scheduling, heterogeneous resource, load balance**

I. Introduction

By simulating the process of molecular docking, virtual screening filters out potential drug molecular, which significantly reduces the cost and improves the efficiency of drug discovery. Autodock Vina [1] is the most widely used open source software in virtual drug screening. Many high performance clusters embrace heterogeneous architecture with CPU and coprocessors such as GPU and MIC [2]. For example, Tianhe-2 consists of 16,000 nodes and each node contains two Intel Xeon processors and three Intel Xeon Phi coprocessors (architecture name MIC). Although incorporating coprocessors provides more potential parallelism (e.g, with more than 200 cores on Xeon Phi), it imposes difficulties on applications to adapt and fully utilize available resources [3]. Moreover, heterogeneous architecture generates another source of load unbalance due to the varying computing capability.

Autodock Vina is the latest version of Autodock, which demonstrates its superiority in molecular docking for both fast speed and high accuracy [4]. Although Vina achieves satisfying performance, it has several drawbacks impeding its adoption in large-scale docking simulation. The current limitations of Vina includes:

- ***Confined Scalability*** Each Vina instance can only run on a single-node, which leads to the reliance on manual scripts to execute multiple instances in cluster, which is difficult to optimize the load across multiple nodes. In addition, the current implementation of Vina is unaware of the heterogeneous computing resources, leaving the Xeon Phi coprocessors un-utilized during the simulation.
- ***Load Imbalance*** The docking point of each Vina instance is selected randomly, which leads to unbalanced computation across the Vina instances. The situation is exacerbated when incorporating the massive computing resources of Xeon Phi coprocessor, due to the intrinsic performance heterogeneity between CPU and MIC.

To address the above limitations, we propose VinaSC, a scalable Vina with fine-grained scheduling on CPU+MIC architecture. In this paper, we target our work on the second generation of Xeon Phi Coprocessor (Knights Corner). To fully exploit the parallelism of Xeon Phi processor while taking advantage of unicore performance on CPU, we implement our framework with the offloading mode [5], where MIC serves as coprocessor and sends computation results back to host CPU.

Specifically, this paper makes the following contributions:

- We incorporate the massive computation resources of MIC into the docking simulation of Vina. Specifically, space searching is offloaded to MIC that effectively improves the performance of docking.
- We implement a dynamic fine-grained scheduling framework, which enables Vina scaling out to a cluster and achieves load balance within single node and across different nodes.
- We propose several optimizations to fully exploit the heterogeneous hardware resources and further accelerates the docking simulation.

The remainder of the paper is organized as follows. Section II describes the background of molecular docking and Intel MIC architecture. Section III discusses the detailed design and optimizations of the fine-grained scheduling frame-

[1]Corresponding author.
[2]This work was done while Lang Yu was an undergraduate student of Beihang University and interning at Intel.

work. Section IV presents experimental results of VinaSC on PDBBIND data. Section V summarizes our work.

II. Background

We first describe the background of MIC and Autodock Vina. After that we elaborate the load balance issue when applying MIC+CPU architecture in docking simulation.

A. Molecular Docking

Molecular docking is the process of searching for the ligand-receptor pair that could becomes potential drug complex. Autodock Vina generally conducts docking in a user-specified space through configure file. User-specified number of docking points will be randomly selected in that space and then the corresponding search threads are created to evaluate docking conformation in given docking point. Search results of different threads are merged and refined when all search threads exit. In this work, we refer one instance of Vina as a job, and each search thread of a job as a task. The local search method used by a task enables the search threads run independently, which provides enough opportunity for parallelism.

B. MIC Architecture

Intel Xeon Phi is the Intel Many Integrated Core Architecture-based coprocessor that provides many advantages over traditional CPU and GPU accelerator [6]–[8]. It offers a high degree of parallelism with multiple small cores (60 70), that run at a relative low clock rate (usually 1.3 1.5GHz). Programs that run on Xeon processor can be directly compiled to MIC. In addition, a simplified Linux OS is re-installed on MIC. Such support greatly relieves the burden of the programmers to learn new programming models. Although applications can run on Xeon Phi natively, domain specific optimizations are still necessary to fully exploit performance potential of massive computing resources [9].

C. Load Imbalance

The load imbalance emerges as critical performance issue when incorporating both CPU and MIC for the docking simulation. There are two sources load imbalance comes from. 1) *Process-level*, which refers to imbalance mainly caused by the difference of spatial structure of ligands and receptors. Process-level imbalance degrades docking performance within a single node as well as across multiple nodes. The problem is exacerbated on MIC due to its lower core performance compared to CPU. 2) *Thread-level*, which refers to the performance variance between search threads (tasks). Thread-level imbalance is intrinsic due to the randomness of docking point selection. In extreme cases, the performance difference could be more than 30% according to our empirical study.

III. System Design and Optimization

In this section, we first give a general description of the overall framework, and then expand on details about implementation and scheduling strategy. In addition, several optimizations are proposed to further improve the performance.

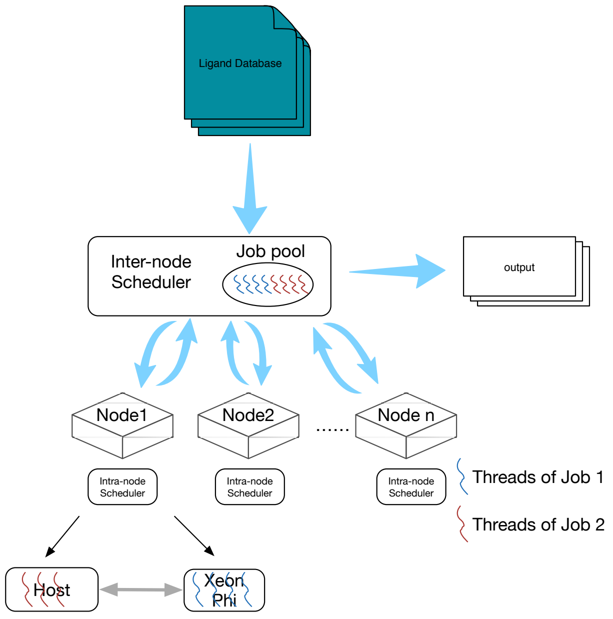

Fig. 1: The Design Overview of VinaSC.

A. Design Overview

The design overview of the fine-grained scheduling framework for scalable Vina is shown in Figure 1. We implement an inter-node scheduler using MPI. The total ligand library is scanned by the scheduler that monitors the utilization of computing nodes and distributes docking jobs accordingly. Instead of docking for only one pair of molecular in a single run, we modified Vina to be able to dock multiple pairs. Input data such as ligands, receptors as well as configuration files are shared among all computing nodes through NFS. The goal of scheduling at inter-node level is to assign more jobs to nodes with stronger computation capability.

Inside a computing node, an intra-node scheduler manages the job pool of this node. The job pool receives docking jobs from inter-node scheduler. Similar to the inter-node scheduler, the intra-node scheduler also monitors the utilization of CPU and MIC and distributes jobs accordingly. The goal of scheduling at intra-node level is to assign more jobs to CPU, since the frequency of CPU is nearly twice as MIC. Carefully managing the load between CPU and MIC is essential to balance the computation throughout the docking simulation.

B. Pipelined Workflow

Original Vina only docks one pair of molecular in a single run, which is unsustainable to screen large database with tens of thousands of moleculars. Moreover, as presented in section II, thread-level load imbalance negatively affects the performance. To deal with these problems, we change the docking process into pipelined workflow, consisting two kinds of threads, producer and consumer.

Producer threads continually read input files of different jobs from the scheduler and parse them into self-defined structures. In addition to parsing molecular files, producer threads prepare data for the conformation search. The prepared data is encapsulated into the data structure of task, and then the task is pushed into task queue, which can be concurrently accessed by consumer threads.

Consumer threads perform conformation search by popping task from task queue. When a search task is done, the consumer thread tests whether it is the final task of a particular job. If it is, the consumer thread executes subsequent stage, merging output from other search tasks that belong to the same job. If it is not, the consumer thread fetches another task from task queue. The design of pipelined workflow eliminates the load imbalance at thread level by interleaving the search tasks from different docking jobs. The pipelined workflow is executed in the same way on both CPU and MIC.

C. Offloading Infrastructure

Considering the advantage of MIC to support native x86 codes, we choose to offload the docking process to MIC, similar to its execution on the CPU side. In order to manage the offload execution and corporation with CPU, we design a send-and-wait mechanism to control and monitor docking on CPU and MIC using Intel Coprocessor Offload Infrastructure (COI). CPU is the manager to require jobs from the job pool and send jobs to MIC. To control the number of jobs sent to MIC, CPU side keeps a slot queue, representing how many jobs can be sent to MIC. Every time the scheduler tries to send a job to MIC, it requires a job from the job pool and pop a slot from the slot queue. After that, the main docking process on MIC is the same as on CPU.

To minimize the communication overhead when sending jobs from CPU to MIC, instead of transferring the actual input data associated with the job, we send the job index to MIC since the input data is available to both CPU and MIC through NFS. To ensure the sequential order of sending job index and waiting for job completion, we design a two pipeline mechanism to perform offloading, including one send pipeline and one wait pipeline. Therefore, the send and wait operations do not interfere with each other, avoiding the potential deadlock. Offloading computation to MIC with two-pipeline implementation effectively coordinates the docking simulation between host CPU and MIC.

D. Dynamic Scheduling

In this section, we propose the dynamic scheduling mechanism to distribute jobs within individual node and across multiple nodes in order to achieve process-level load balance. To increase the utilization of fast CPU cores while saturate slow MIC cores as much as possible, we use the concept of remaining number of ligands (RNL) to indicate the number of ligands reserved for CPU. When the number of jobs in the job pool is less than RNL, the scheduler prevents MIC from getting new jobs. Note that RNL is dynamically adjusted by the scheduler during runtime. Every time a job is finished (whether on CPU or MIC), the scheduler calculates the ratio of execution time of CPU and MIC job and update the RNL according to Equation 1. Therefore, RNL reflects the computation capability of CPU and MIC in real time.

$$RNL = R \times MAX(1, \frac{\#\ CPU\ cores}{\#\ tasks\ per\ ligand}) \qquad (1)$$

where

$$R = \frac{Docking\ time\ on\ MIC}{Docking\ time\ on\ CPU} \times 0.5 + R \times 0.5 \qquad (2)$$

As shown in Equation 1, RNL is the number of CPU jobs a cluster can run at one time multiplied by the computing capability ratio of CPU and MIC (indicated by R). In other word, RNL reveals the number of jobs CPU can finish during the time MIC completes one job. It is easy to infer that the ideal RNL guarantees the computation on MIC always finishes no later than CPU. In order to eliminate the load fluctuation, R is updated based on history and latest job execution time as shown in Equation 2. Taking history data into account is useful to prevent R from oscillation that leads to unnecessary idleness of MIC.

In ideal case, RNL can accurately identify jobs that need to remain for CPU. However, due to the error of measurement, randomness of the algorithms and different computing complexity between molecular, ideal RNL can hardly be reached. Therefore, we introduce a constant TUNER in Equation 3 to compensate the inaccuracy. Based on our empirical study, assigning more jobs than ideal for CPU would not degrade the performance significantly. Thus we can adjust the TUNER to determine the actual value of RNL.

$$RNL = R \times MAX(1, \frac{\#\ CPU\ cores}{\#\ tasks\ per\ ligand}) \times TUNER \qquad (3)$$

E. Other Optimizations

In addition to the fine-grained scheduling framework, we implement several other optimizations on VinaSC to further improve the performance.

1) Input parsing parallelization: We introduce Intel Cilk Plus to parallelize the process of input parsing. We take the advantage of *clik_for* to execute loops in parallel. In order to ensure correctness, we modify shared and local variables to eliminate data dependency during loop iteration.

2) Vectorization: Vectorization transforms target code from SISD to SIMD so that it enables manipulation on massive data simultaneously. We leverage the optimizations from Intel compiler, which includes auto-vectorization using compile option *-vec-report** as well as annotation pragma *ivdep* and *vector always* to guide compiler vectorization.

IV. Evaluation

A. Experiment Setup

We use PDBBIND [10], which is a widely used small molecule database containing over 330,000 molecular as ligand. As for receptor, we use 1IEP cancer cell. We randomly choose ligands from the database as the test dataset. We evaluate our framework by docking the test dataset with 1IEP.

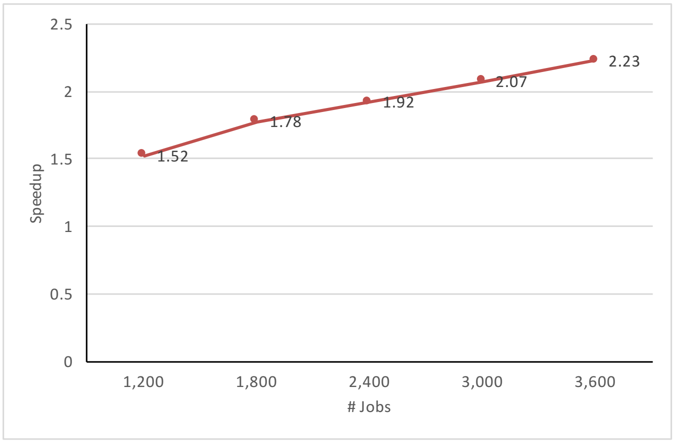

Fig. 2: Performance Speedup of VinaSC under different # jobs.

Our experiments are conducted on a cluster with 6 nodes. Each node consists of one Intel Xeon E5-2697 v2 CPU, one Xeon Phi 7120A coprocessor and 64GB main memory. The cluster uses Infiniband TrueScale QDR40 with 40 Gbps bandwidth to connect each node. The software stack at each node includes Redhat OS v6.2, Intel parallel studio XE 2015 update 3 and Intel MPI 4.1.0.24.

B. Speedup

We evaluate the performance of VinaSC on 6 nodes, varying the total number of docking jobs. We set the TUNER in Equation 3 to 1.5 in the following experiments. Since the original Autodock Vina can only perform docking on a single node at a time, it cannot be used directly as the performance baseline. Thus, we implement another framework with process-level scheduler. This scheduler distributes original Vina instances only on host CPUs among nodes. We use the intuitive process-scheduler as a baseline to show the performance speedup achieved by VinaSC.

As shown in Figure 2, VinaSC achieves scalable performance speedup when the number of jobs increases from 1,200 to 3,600. We set 12 search threads for each job. The best performance speedup is achieved by 2.23x compared to the baseline when the number of jobs reaches 3,600. The reason for the linear performance scalability of VinaSC is due to better scheduling decisions for load balance between CPU and MIC, since the scheduler manipulates more jobs (more information) to adjust RNL.

C. Correctness

Since we made modifications to original Vina, it is necessary to verify the correctness of docking results. Due to the randomness of the algorithm, the results of the same simulation varies across runs, which makes it hard for validation. After consulting domain experts, we set the tolerable variation to be 5%. Thus, it is acceptable if the affinity result (kcal/mol) calculated using our framework varies within 5% compared to original Vina. Since results with higher affinity scores affect the docking quality significantly, we only verify the top three docking conformation.

Based on the above criterion, we randomly choose 50 ligands from PDBBIND, docking them with randomly selected proteins using our framework and original Autodock Vina. Each molecular pair is docked repeatedly for 10 times. For each docking pair, the number of search threads is set to 24 (*exhaustiveness* = 24). Across all experiments, the average affinity difference is 0.82% and max affinity difference is less than 4.8%. The results demonstrate the correctness of our implementation of VinaSC.

V. Summary

In this paper, we propose VinaSC, a fine-grained scheduling framework to scale out docking simulation in cluster of heterogeneous computing resources such as CPU and MIC. Our experimental results demonstrate significant performance improvement over original Autodock Vina by more than 2.3x. Moreover, the performance speedup scales linearly when the number of the docking jobs increases.

VI. Acknowledgement

This research is supported by the National Natural Science Foundation of China (Grant No. 61133004, 61502019). We thank Dr. Kunqian Yu from Shanghai Institute of Materia Medica, Chinese Academy of Sciences who gives us advice on biological algorithms and result validation.

References

[1] "Autodock Vina Manual," http://vina.scripps.edu/manual.html.
[2] "Intel Many Integrated Core Architecture," http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html.
[3] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. A. Jarvis, "Exploring simd for molecular dynamics, using intel® xeon® processors and intel® xeon phi coprocessors," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 1085–1097.
[4] O. Trott and A. J. Olson, "Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *Journal of computational chemistry*, vol. 31, no. 2, pp. 455–461, 2010.
[5] C. J. Newburn, S. Dmitriev, R. Narayanaswamy, J. Wiegert, R. Murty, F. Chinchilla, R. Deodhar, and R. McGuire, "Offload Compiler Runtime for the Intel® Xeon Phi Coprocessor," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 1213–1225.
[6] J. Jeffers and J. Reinders, *Intel Xeon Phi coprocessor high-performance programming*. Newnes, 2013.
[7] R. Rahman, *Intel® Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress, 2013.
[8] G. Chrysos, "Intel® xeon phi coprocessor-the architecture," *Intel Whitepaper*, 2014.
[9] H. Peréz-Sánchez, A. Fassihi, J. M. Cecilia, H. H. Ali, and M. Cannataro, "Applications of high performance computing in bioinformatics, computational biology and computational chemistry," in *International Conference on Bioinformatics and Biomedical Engineering*. Springer, 2015, pp. 527–541.
[10] R. Wang, X. Fang, Y. Lu, and S. Wang, "The pdbbind database: collection of binding affinities for protein-ligand complexes with known three-dimensional structures," *Journal of medicinal chemistry*, vol. 47, no. 12, pp. 2977–2980, 2004.