# Acceleration of protein-ligand docking simulation using graphics processing units

Takahiro Sasaki
Department of Computer Science
Tokyo Institute of Technology

Takurou Udagawa
Department of Computer Science
Graduate School of Information
Science and engineering
Tokyo Institute of Technology

Masakazu Sekijima
Department of Computer Science
Graduate School of Information
Science and engineering
Tokyo Institute of Technology
Global Scientific Information and Computing Center
Tokyo Institute of Technology

***Abstract*—The abstract goes here.**

In modern drug development, it is important to obtain ligand (small molecule) which binds specifically to a target protein related to disease. A common structure-based screening approach is to use molecular docking simulation to identify the docking between a target protein and ligand. This approach must simulate a docking to predict whether a target protein binds to ligand for tens of millions compounds. Screening a library of compounds is enormously costly and time-consuming. When three-dimensional (3-D) structure of the target protein is already known, molecular docking simulation between a ligand and a protein can be performed in order to search large ligand libraries for drug-candidates: ligands that bind to this protein with high affinity. In this paper, we show how AutoDock, which is software for simulating the docking between a target protein and ligand using a genetic algorithm, can calculate the genetic algorithms scoring function in parallel. The methods presented for parallelizing the workload result in an average speedup of 3.3 times on a comparative CUDA enabled graphics processing unit (GPU)(CUDA: Compute Unified Device Architecture).

## I. Introduction

In modern drug discovery, the identification of small molecules that especially bind to a target protein is often used in searching for drug candidates. Most proteins bind to another molecule and then express their functions[1]. Recently, virtual screening has been used to search for leading-candidate compounds by screening a large library of compounds with a computer quickly and comprehensively. The main objective of virtual screening is to create useful compound library for molecular design by screening likely lead compounds in massive compound libraries.

Two commonly used approaches to virtual screening are ligand-based screening and structure-based screening[2], [3], [4]. Ligand-based screening searches for ligands that have similar chemical properties and structures to ligands that is known to bind to a target protein. In structure-based screening, searching for lead compounds is based on structure of a target protein. Docking simulation is commonly used docking simulation between target protein and ligand as structure-based screening. Docking simulation is costly and time consuming because tens of millions of compounds must be examined whether they bind to a target protein[5], [6], [7].

Docking simulation software such as DOCK[8], AutoDock[9], or myPresto sievgene[10] is commonly used to predict the binding mode between a protein and a ligand. DOCK is the oldest docking simulation software. Searching for binding mode is done to evaluate the energy between the protein and ligand by a function based on the force field of DOCK by assigning protein's atom to a steric lattice (grid). In AutoDock, the energy between the protein and ligand is calculated by a scoring function based on the AMBER force field[11]. A optimization of the energy between the target protein and ligand is computed using a genetic algorithm. This enables flexible docking to take account of the flexibility of ligands. A method of myPresto sievgene is to calculate the grid potential of a target protein and, then screen the search space with rigid docking of the ligand. The structure that has the best energy is selected to optimize the energy. Finally, the score of the binding mode is evaluated strictly In this paper, we present a method for calculating the scoring function in parallel on graphics processing units (GPUs) with AutoDock. The scoring function can be computed in parallel, so we present a method for calculating the scoring function in parallel using GPUs.

June 20, 2013

## II. AutoDock

AutoDock[12], [13], [14], [15] was developed by and is maintained by The Scripps Research Institute Olson Laboratory. It can take full ligand flexibility into account while treating the protein as a rigid compound during docking simulation.

Prior to a docking calculation in AutoDock, a grid calculation is computed. This grid calculation involves computing the energy of the target protein by assigning its atom to grid points. In the docking calculation, the ligand is set to the energy grid that represents the target protein: then search heuristically the optimal solutions when the energy between protein and ligand is minimum . A heuristic search algorithm such as simulated annealing[16], hill climbing, swarm intelligence[17], or a genetic algorithm[18] is used commonly to sample a large solution space in AutoDock. Solutions produced by the genetic algorithm execution are some-times analysed using clustering.

### A. Grid Algorithms

Most types of molecular docking simulation software use a grid algorithm. First, a target protein is divided into a 3-D cubic lattice. Second, the atoms of the target protein are assigned to grid points in order to compute the energy value by running a probe atom (protein atom assigned to a grid point). In a docking calculation, AutoDock computes the sum of the energy of the grid points which overlaps with ligand atoms. This enables a docking calculation to be performed by the grid algorithm without taking into account large binding conformation.

### B. Genetic Algorithm

A genetic algorithm (GA) is a heuristic search algorithm inspired by the process of natural evolution. A genetic algorithm efficiently searches for optimal solutions from a large search space by iterating genetic operations such as selection, crossover, and mutation. Selection is performed according to the fitness of individuals that have data (candidate solutions) represented by genes (chromosome). Fitness is calculated using scoring function. The advantages of genetic algorithm are they can obtain multiple solutions and are applicable to NP-hard problems if the manner or representing genes is altered[19], [20].

In AutoDock, the gene of each individual is represented by a vector that describes the binding mode. It has a 3-D translation (location of ligand in the energy grids) and orientation (quaternion of real numbers) and torsions (angle of ligand's rotatable bond).

AutoDock's genetic algorithm starts with a population of solutions encoded in one of many ways. Its flowchart is as follows:

1) Generate an initial population at random.
2) Select individuals according to fitness by calculating the scoring function.
3) Produce the next generation from the selected individuals.
4) Crossover pair of individuals in this generation.
5) Mutate some genes of individuals in this generation.
6) If the termination condition is satisfied, finish; else go to step 2.

AutoDock's genetic algorithm's termination condition is as follows:

1) The difference between the energies of individuals in a generation is less than threshold value.
2) The number of generations have reached a maximum.

The computation cost of a genetic algorithms is high because this algorithm requires a large amount of iterative calculation. To solve this problem, parallelized model of genetic algorithms (parallel genetic algorithm) has been studied[21], [22], [23]. The genetic operations in a genetic algorithm have a high degree of parallelism. The basic view of parallel genetic algorithm is to divide populations into some sub-populations and compute a search calculation on each processor. However this method must communicate among the divided sub-populations: this data communication among processes is called migration. Island model is often used as communication model of parallel genetic algorithm. How to communicate among processes is actively studied.

## III. Parallel computation using GPUs

This section describes a method for calculation of scoring function on GPUs. In genetic algorithms, the computation time of the scoring function is important because this function is computed in every generation. A profile of AutoDock computation times shows that the computation time for GAs accounts for most of the calculation: over 94% of the total. Furthermore, it is clear that the scoring function is needed for time consumption (about 70%) of the GA calculation in AutoDock. The time consumed by each step of the GA in AutoDock is shown in Table I. The figure in parentheses is the percentage of each step in the genetic algorithm computation time and the figure on the left of the parentheses is the percentage of each step of the genetic algorithm of the whole docking simulation's computation time in occupation of the column in TableI.

We used CUDA[26] which is an integrated develop environment from NVIDIA to implement the parallelizing computation on GPUs. NVIDIA GPU hardware has many streaming multiprocessor. A streaming multiprocessor is consisted of some streaming processors. A streaming processor is device that can calculate floating-point arithmetic in single precision and integer arithmetic. Fermi architecture GPU device has up to 16 streaming multiprocessors. In Fermi architecture, a streaming multiprocessor is called CUDA core.

The CUDA programming model assumes that the all threads execute on a physically separate GPU from CPU running the application. CUDA-enabled GPUs have both on-chip and on-board memory. Global memory lise in on-board and shares in all the data of CUDA cores and memory access of global memory is slow. How to efficiently access global memory (coaleacing) is an essential requirement to CUDA programming. Shared memory can be either 16 KB or 48 KB per SM arranged in 32 banks that are 32 bits wide. Shared memory is not as fast as register memory, but faster than global memory beacause shared memory is on-chip memory.

Threads are managed hierarchically. Grid is consisted of some blocks and thread block is a collection of multiple threads. Thread block has identifier in grid and thread has id in thread block. A thread scheduler assigns threads to streaming processor. Each streaming processor in a streaming multiprocessor executes the same instruction for different data (Single-Instruction Multiple-Data).

In the scoring function's parallelizing computation, we assigned individuals in a generation to a thread block and each ligand atom to a thread in order to calculate energy. Ligand atom data is transferred into global memory for a while. Each thread calculates the energy using ligand atom data which is transferred from global memory into shared memory. When the computation of all the threads has been completed, the reduction in energy is calculated and stored results in global memory.

The scoring function can be computed in parallel for individuals. We implement the scoring function to calculate it on GPUs.

TABLE I. PROFILE OF AUTODOCK

| Step | Time (sec) | Occupation (%) |
|---|---|---|
| AutoDock preprocessing | 6.1 | 4.9 |
| Genetic Algorithm | 116.4 | 94.9 |
| Scoring function | 85.7 | 69.9 (73.6) |
| Selection | 3.3 | 2.7 (2.8) |
| Crossover | 2.3 | 1.8 (2.0) |
| Mutation | 0.6 | 0.0 (0.0) |
| Sorting population | 14.9 | 12.1 (12.8) |
| Others | 9.2 | 7.5 (7.9) |
| Analysis | 0.1 | 0.2 |
| Total | 122.6 | 100 |

### A. Scoring Functions

The scoring function is computed using two kernels. The atom coordinates necessary for the scoring calculation are loaded from global memory and subsequently transferred into shared memory. For two kernels of an execution configuration, a thread block is assigned to individuals in the population. The first kernel calculates the sum of the energy between a ligand and a protein from energy grids using the trilinear interpolation algorithm. All threads synchronously compute the reduction in energy in each block. The energy computed by this kernel is stored in global memory.

The second kernel is used to compute the intramolecular energy between non-bonded ligand atoms. Each thread is assigned to a non-bonded atom pair and computes the Coulomb potential, van der Waals potential, and hydrogen bonding energy. Finally, all the threads compute the reduction in energy and store the results in global memory.

### B. Data Copy Hiding

Since the CPU memory space is different from the GPU memory space, data must be transferred between the CPU and the GPU. A memory copy is generally performed synchronously; namely, during a memory copy between CPU and GPU, GPUs can not calculate until data transmission is complete. In the case of an asynchronous memory copy, while data is being copied, the kernel is executed on GPUs.

## IV. RESULTS AND DISCUSSION

We performed experiments to compare the existing method (performing the calculation in a CPU) and our new method (using a GPU). In the case that the number of individuals in the genetic algorithm was 32 and 16,384, we compared the time to compute the whole scoring function for all the dataset. For the dataset known as 1HSG, we compared the speedup in performance, i.e., the reduction in time taken to compute the whole scoring function, when the number of individuals was changed from 32 to 16,384.

### A. Datasets

We used the datasets, listed in Table II, which were downloaded from Protein Data Bank[27] to perform all the comparative experiments.

### B. Experiment Environment

These comparative experiments were performed on TSUBAME 2.0 supercomputer at Tokyo Institute of Technology. The computation environment is described in Table III.

TABLE II. DATASET

| PDB ID | 1HSG | 1ADB | 3PTB | 7ABP |
|---|---|---|---|---|
| Molecular weight | 22221.4 | 81387.8 | 23484.7 | 33543.5 |
| Number of residues | 99 | 374 | 223 | 306 |
| Number of ligand atoms | 49 | 52 | 15 | 13 |

TABLE III. COMPUTATION ENVIRONMENT

| | CPU | GPU |
|---|---|---|
| Model | Intel Xeon X5670 | NVIDIA Tesla M2050 |
| Clock rate | 2.93 GHz | 1.14 GHz |
| Memory | 56 GB | 3 GB |
| Compiler | gcc 4.3.4 | nvcc 4.1 |
| OS | SUSE Linux Enterprise Server 11 SP1 | |

### C. Measurement method

We use the gettimeofday() function to measure the computation time, which is the difference between the start time and end time. The number of genetic algorithm executions in AutoDock was 10. The computation time of scoring function is defined as the time to compute the whole scoring function.

### D. Computation Time

TABLE IV. TIME CONSUMPTION OF SCORING FUNCTION; NUMBER OF INDIVIDUALS 32

| PDB ID | CPU (sec) | GPU (sec) | Speedup |
|---|---|---|---|
| 3PTB | 337 | 6,758 | 0.05x |
| 7ABP | 673 | 5,471 | 0.12x |
| 1HSG | 2,792 | 6,758 | 0.41x |
| 1ADB | 5,380 | 7,378 | 0.73x |

TABLE V. TIME CONSUMPTION OF SCORING FUNCTION; NUMBER OF INDIVIDUALS 16384

| PDB ID | CPU (sec) | GPU (sec) | Speedup |
|---|---|---|---|
| 3PTB | 680 | 431 | 1.58x |
| 7ABP | 1,228 | 861 | 1.43x |
| 1HSG | 5,372 | 1,631 | 3.29x |
| 1ADB | 6,537 | 2,256 | 2.90x |

The computation times of the existing method (CPU) and proposed method (GPU) when the number of individuals was 32 are shown in Table IV. For all the datasets, our method was slower than the existing method. By contrast, when the number of individuals was 16,384, out method was quicker than the existing method, as shown in Table V. The speedup rate for 1HSG was about x3.3 it was the most quickly computed among all of the datasets. Table VI compares speedup of the scoring function computation time achieved by the existing and proposed methods when the number of individuals was changed from 32 to 16,384. When the number of individuals was 32 and 64, out method was slower than the existing method, but when the number of individuals was 128 or more, our method was quicker. Furthermore, when the number of individuals was 2,048 or more, the speedup rate was about 3.3 times, which represent saturation.

## V. CONCLUSION

In this paper, we presented a method for calculating the scoring function in genetic algorithms with AutoDock on GPUs to reduce the computation time of AutoDock. The scoring function is computed using two kernels. In both kernels, atom coordinate data are loaded from global memory.

TABLE VI. TIME CONSUMPTION OF SCORING FUNCTION; NUMBER OF INDIVIDUALS 16384

| Number of individuals | Speedup |
|---|---|
| 32 | 0.41 |
| 64 | 0.72 |
| 128 | 1.42 |
| 256 | 2.21 |
| 512 | 2.57 |
| 1024 | 2.98 |
| 2048 | 3.20 |
| 4096 | 3.22 |
| 8192 | 3.33 |
| 16384 | 3.33 |

When each kernel is computed, the atom coordinate data are transferred from global memory into shared memory. Asynchronous data transfer enables the data transfer between CPU and GPU to be hidden by overlapping the kernel calculation and data transfer.

We performed experiments to compare the time consumption of AutoDock in CPU to that in the GPU. The experimental results show small speedup when the number of individuals was small but a larger speedup when the number of individuals was large. We achieved an average speedup of 3.3 times with 1HSG as the dataset.

## REFERENCES

[1] Marcotte, Edward M., Matteo Pellegrini, Ho-Leung Ng, Danny W. Rice, Todd O. Yeates, and David Eisenberg. "Detecting protein function and protein-protein interactions from genome sequences." Science 285, no. 5428 (1999): 751-753.

[2] Walters, W. Patrick, Matthew T. Stahl, and Mark A. Murcko. "Virtual screening-an overview." Drug Discovery Today 3, no. 4 (1998): 160-178.

[3] Lyne, Paul D. "Structure-based virtual screening: an overview." Drug Discovery Today 7, no. 20 (2002): 1047-1055.

[4] Walters, W. Patrick, Matthew T. Stahl, and Mark A. Murcko. "Virtual screening-an overview." Drug Discovery Today 3, no. 4 (1998): 160-178.

[5] Shoichet, Brian K. "Virtual screening of chemical libraries." Nature 432, no. 7019 (2004): 862-865.

[6] Bissantz, Caterina, Gerd Folkers, and Didier Rognan. "Protein-based virtual screening of chemical databases. 1. Evaluation of different docking/scoring combinations." Journal of medicinal chemistry 43, no. 25 (2000): 4759-4767.

[7] DiMasi, Joseph A., Ronald W. Hansen, and Henry G. Grabowski. "The price of innovation: new estimates of drug development costs." Journal of health economics 22, no. 2 (2003): 151-186.

[8] Ewing, Todd JA, Shingo Makino, A. Geoffrey Skillman, and Irwin D. Kuntz. "DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases." Journal of computer-aided molecular design 15, no. 5 (2001): 411-428.

[9] Morris, Garrett M., Ruth Huey, William Lindstrom, Michel F. Sanner, Richard K. Belew, David S. Goodsell, and Arthur J. Olson. "AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility." Journal of computational chemistry 30, no. 16 (2009): 2785-2791.

[10] Omagari, Katsumi, Daisuke Mitomo, Satoru Kubota, Haruki Nakamura, and Yoshifumi Fukunishi. "A method to enhance the hit ratio by a combination of structure-based drug screening and ligand-based screening." Advances and applications in bioinformatics and chemistry: AABC 1 (2008): 19.

[11] Jayaram, B., D. Sprous, and D. L. Beveridge. "Solvation free energy of biomacromolecules: Parameters for a modified generalized Born model consistent with the AMBER force field." The Journal of Physical Chemistry B 102, no. 47 (1998): 9571-9576.

[12] Morris, Garrett M., David S. Goodsell, Ruth Huey, and Arthur J. Olson. "Distributed automated docking of flexible ligands to proteins: parallel applications of AutoDock 2.4." Journal of computer-aided molecular design 10, no. 4 (1996): 293-304.

[13] Morris, Garrett M., David S. Goodsell, Robert S. Halliday, Ruth Huey, William E. Hart, Richard K. Belew, and Arthur J. Olson. "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function." Journal of computational chemistry 19, no. 14 (1998): 1639-1662.

[14] Huey, Ruth, Garrett M. Morris, Arthur J. Olson, and David S. Goodsell. "A semiempirical free energy force field with chargebased desolvation." Journal of computational chemistry 28, no. 6 (2007): 1145-1152.

[15] Goodsell, David S., Garrett M. Morris, and Arthur J. Olson. "Automated docking of flexible ligands: applications of AutoDock." Journal of Molecular Recognition 9, no. 1 (1998): 1-5.

[16] Aarts, Emile HL, Jan HM Korst, and Peter JM Van Laarhoven. "Simulated annealing." Local search in combinatorial optimization (1997): 91-120.

[17] Kennedy, James. "Swarm intelligence." Handbook of nature-inspired and innovative computing (2006): 187-219.

[18] Holland, John H. "Adaptation In Natural And Artificial Systems: An Introductory Analysis With Applications To Biology, Control, And Artific." (1992): 211.

[19] De Jong, Kenneth A., and William M. Spears. "Using genetic algorithms to solve NP-complete problems." In Proceedings of the third international conference on Genetic algorithms, vol. 124, p. 132. Morgan Kaufmann, San Mateo, CA, 1989.

[20] Joines, Jeffrey A., and Christopher R. Houck. "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's." In Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, pp. 579-584. IEEE, 1994.

[21] Muhlenbein, Heinz. "Evolution in time and space-the parallel genetic algorithm." In Foundations of genetic algorithms. 1991.

[22] Mühlenbein, Heinz, M. Schomisch, and Joachim Born. "The parallel genetic algorithm as function optimizer." Parallel computing 17, no. 6 (1991): 619-632.

[23] Cantú-Paz, Erick. "A summary of research on parallel genetic algorithms." (1995).

[24] Posp chal, Petr, Jiri Jaros, and Josef Schwarz. "Parallel genetic algorithm on the cuda architecture." Applications of Evolutionary Computation (2010): 442-451.

[25] Li, Jian-Ming, Xiao-Jing Wang, Rong-Sheng He, and Zhong-Xian Chi. "An efficient fine-grained parallel genetic algorithm based on gpu-accelerated." In Network and parallel computing workshops, 2007. NPC workshops. IFIP international conference on, pp. 855-862. IEEE, 2007.

[26] "NVIDIA CUDA DEVELOPER ZONE" https://developer.nvidia.com/

[27] Protein Data Bank http://www.rcsb.org/