Proceedings of Department of Defense High Performance Computing Modernization Program Users Group Conference 2007, June 18-22, 2007, Pittsburgh, PA

DOVIS: A Tool for High-throughput Virtual Screening

Xiaohui Jiang, Kamal Kumar, Anders Wallqvist, and Jaques Reifman
US Army Medical Research and Materiel Command (MRMC), Biotechnology HPC Software Applications Institute (BHSAI), Telemedicine and Advanced Technology Research Center, Ft. Detrick, MD
xjiang@bioanalysis.org

Abstract

We developed a DOcking-based VIrtual Screening (DOVIS) pipeline to predict how small molecules may interact with a given protein, so that we can rank a large database of molecules based on their predicted affinities to the protein. We used the program AutoDock as the docking engine and OpenBabel as the molecular data model. We designed and implemented a novel parallelization scheme using a file-based inter-process communication protocol to control parallel jobs. A Web-server/Web-page architecture was developed based on the User Interface Toolkit (UIT) to manage DOVIS jobs on high performance computing (HPC) platforms and provide a Web-page-based graphical user interface (GUI). The DOVIS pipeline is running on JVN at the Army Research Laboratory (ARL) Major Shared Resource Center (MSRC). Scientists at several Department of Defense (DoD) labs are currently using the DOVIS pipeline in their research projects.

1. Introduction

A key factor in the development of small-molecule therapeutics against biological threats is the ability to identify initial lead compounds that can be used by medicinal chemists as a starting point for developing therapeutic drugs. Experimentally screening a large chemical database of compounds can be time consuming and expensive. Molecular docking is a computational technique to predict how a small molecule may interact with a protein. Docking-based virtual high-throughput screening is an *in silico* approach to rank order a large database of molecules against a given protein target. It saves time and resources by focusing and reducing the number of compounds to be experimentally tested. In the last several years, virtual screening has become an accepted tool in drug discovery. It has been successfully applied in a number of therapeutic programs, in particular, at the lead discovery stage (Ghosh, et al., 2006). With the explosive growth of publicly available chemical databases, molecular docking techniques need to be capable of screening millions of compounds. Since docking one molecule may take 1 to 5 minutes, screening millions of molecules requires the use of high performance computing.

Several commercial docking programs, such as Glide (Friesner, et al., 2004), LigandFit (Venkatachalam, et al., 2003) and FlexX (Rarey, et al., 1995), can distribute docking jobs to computers on a network. However, protocols that can seamlessly dock millions of compounds and capture the top percentage of high-scoring ligands are not standard. There are two major requirements for such a protocol running on a shared Linux cluster: (1) the ability to launch parallel docking jobs through a queuing system; and (2) the ability to process millions of compounds in a reasonable time. Since the latter requirement may call for hundreds of CPUs working simultaneously, the protocol should effectively handle the associated data flow through the file system without affecting the performance of the cluster. In this paper, we describe a Linux cluster-based protocol using AutoDock (Morris, et al., 1998) as the docking engine.

AutoDock is a broadly used docking program developed at the Scripps Research Institute. Application of AutoDock requires several separate pre-docking steps (e.g., ligand preparation, receptor preparation, and grid map calculations, before the actual docking process can take place). Existing tools, such as AutoDock Tools (ADT) and Blind Docking Tool (BDT) (Vaque, et al., 2006), integrate individual AutoDock steps within a graphical user interface (GUI), but they do not contain the automated capability to effectively process millions of compounds in a single execution. Another limitation of AutoDock is the output file format, which is not commonly used by other molecular modeling programs.

Here, we describe a DOcking-based VIrtual Screening (DOVIS) pipeline, with AutoDock as the docking engine and OpenBabel (http://openbabel.

0-7695-3088-5/07 $25.00 © 2007 IEEE

sourceforge.net) as the molecular data model. This implementation has the following advantages: (1) a dynamic scalable parallelization scheme for AutoDock integrated with queuing systems, such as the Load Sharing Facilities (LSF) from Platform Computing Inc. and the Portable Batch System (PBS) from Altair Grid Technologies; (2) a Web-page-based GUI for users to specify docking parameters, submit docking jobs, and query/visualize docked ligands; (3) an industry standard input/output file interface, in particular the Molecular Data Limited (MDL) structured data (sd) format for the output results; (4) a general interface to use third-party scoring functions and a protocol to retain a user-specified top number of docked ligands based on their scores; and (5) a collection of pre-processed compounds from the ZINC database (Irwin, et al., 2005) in sd format embedded with molecular properties and purchase information.

2. Methodology

2.1. Parallel Docking

Docking a large number of compounds in a database against a given protein target fits the single program multiple data (SPMD) model, where each CPU works on a different part of the database. There is neither overlap of data nor dependency among CPUs. However, there are situations to be considered when running such processes on shared HPC resources:

1. To dock one-million compounds in a reasonable time (~1 week) requires 200+ CPUs. On a shared HPC platform, what's the best way to request CPUs at this scale?
2. Compounds in a database are usually different and so is the time required to dock each of them. Hence, load balancing among CPUs is important.
3. Usually, there is a runtime limit on shared resources. How does one efficiently use the CPUs obtained within the allowed runtime and how to automatically "restart" the process for unfinished jobs?

Taking these situations into consideration, we designed a flexible parallelization scheme to maximize the usage and efficiency of the system:

1. The compounds in a database are grouped into tasks with N compounds in each task. Every CPU works on one task at a time.
2. A file-based inter-process communication (IPC) protocol is used to control the distribution of tasks. This protocol ensures that only one CPU can access the task file at any particular time.
3. Job-array, a queuing method, is used to launch the parallel job.
4. When a CPU is running, it uses the IPC protocol to check-out a task and to run the required computations.
5. Once a task is completed by a CPU, the CPU uses the IPC protocol to check-in the task and upload the results to a central location. Then, the CPU checks whether there is enough runtime left to do another task. If so, it goes back to the previous step; otherwise it exits the process.
6. After all CPUs exited, a separate process, which has a dependency on the job-array processes, starts to match the check-out and check-in task lists and the status of the original task list. If there are unfinished tasks remaining on the list or there are tasks checked out but never checked in, it updates the original task list and restarts the parallel job; otherwise the whole job is completed.

This parallelization scheme has several advantages. First, the process is self controlled without a dedicated control process. The IPC is only used to manage the task list and update results. The data volume involved in IPC is very light. With a typical task runtime of about 5 hours, the waiting time for the IPC protocol is minimal. In addition, the number of CPUs managed by the scheme is dynamically changeable. Users can add or remove a number of CPUs working on the job at runtime. Job-array is one convenient way to launch parallel jobs, but other queuing methods can also be used to launch jobs. Finally, the scheme can automatically re-launch the parallel job depending on the status of the overall tasks.

2.2. Graphic User Interface

In order to provide easy access to non-expert users to the DOVIS pipeline, we developed a GUI using the User Interface Toolkit (UIT). A Web-server was setup at the BHSAI to control DOVIS jobs on HPC resources through the UIT Web-services provided by the Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC). In addition, we used UIT to transfer results (docked molecules) back to the Web-server and visualize those using JMol applets. Therefore, users can access the DOVIS pipeline, submit jobs to HPC resources and visualize results on their Web-browser.

3. Implementation

3.1. Parallel Docking

In the DOVIS pipeline, we used OpenBabel application programming interfaces (APIs) for file import/export and format conversions. In addition, we parsed the AutoDock results to create docked molecules

in MDL sd format with embedded docking scores. We used C++ and Perl scripts to drive all AutoDock executables.

A task list is created right before the start of a parallel job. The input molecules are assigned to tasks. Each task contains a user-specified number of molecules, usually 50 to 100 molecules. For the IPC protocol, we use the system command "lockfile" as a Network File system (NFS) safe file locking mechanism to control access to task files. We implemented two file locks: one for check-out task, and the other for check-in task and update of the saved results. Since a user may wish to save a limited number of top scored molecules, we set up a central directory to hold the sorted results. Each CPU uses the check-in file lock to upload results, which are qualified for the final list.

In order to improve the prediction accuracy, we developed a general interface to run third-party scoring programs within the DOVIS pipeline. From the pipeline, we pass the file names of the protein program database (pdb) file, the docked molecule file, the binding site file, and the scored molecule file to a Perl-wrapper script along with user-specified options for the third-party scoring program. The Perl wrapper is a specific script to drive a particular scoring executable. It also parses the results from the scoring program to embed the scores into the output sd file. Using this approach, the interface parameters between the pipeline and the Perl wrapper are fixed. For a new scoring program, only a new wrapper script is needed to link it to the pipeline, no change is required in the pipeline.

Another concern is that the AutoDock executable docks one molecule at a time, employing file-based I/O data volume around 20 Mb. Most of these data consist of energy grid files. When loading the data from a NFS location with a large number of CPUs running concurrently, the data flow can degrade the performance of the entire Linux cluster. We modified the source code of AutoDock so that after loading the energy grids once, multiple molecules can be docked. This change reduces the I/O data volume by more than 90% when 50 molecules are docked at a time. We tested the scenario with 256 CPUs running concurrently using a NFS, and did not observe any performance degradation of the file system.

Currently, we use the job-array function in LSF or PBS to launch parallel jobs. We also programmed a Perl script, which has a dependency on the parallel job to check the status of the task list. If there is any unfinished task, the script will automatically re-launch the parallel job.

3.2. Graphic User Interface

The DOVIS pipeline Web-server was implemented using Java Servlets/Java Server Pages. It uses the Java version of APIs in UIT to provide Web-services. In addition, it incorporates the usage of JMol applets in Web-pages to display the results. This scheme enables molecular visualization at a user's local machine, which significantly enhances the visualization performance as compared to an X-window-based approach.

3.3. Pre-Docking Steps

Before starting the parallel docking jobs, there are two pre-docking steps. One is to pre-compute energy grids for all possible atom types around the binding site of a target protein. A Perl script was used to drive the AutoGrid executable for the energy grid calculations. The other pre-docking step is to divide the input set of small molecules into partitions (e.g., 10,000 molecules/partition), and to embed a DOVIS ID and other available information (e.g., molecular descriptors and purchase information from the ZINC database) into the partitioned sd file. Using partitioned, smaller input files improves the efficiency and reduces input/output overhead of the parallel docking job since each CPU only needs to copy the partition with the molecules in the current task to its working directory instead of the whole database. In addition, since the name of a molecule may not be unique in a database, a serial DOVIS ID is created to uniquely label every entry in a database.

4. Results

In order to build a compound database for testing purposes and for scientific research, we pre-processed the ZINC database (version 6, 2.3 million compounds) to partition and embed information. Using the pre-processed ZINC database, we tested how many ligands can be docked to a receptor per day with varying numbers of CPUs. We performed tests with up to 128 CPUs on a Linux cluster and obtained a near-linear speedup as a function of the number of CPUs, for the ZINC database. This indicates that our implementation achieved near-optimal performance, and that the system is able to dock about 700 ligands/CPU/day.

In addition, we used the thymidine kinase test case to validate the DOVIS pipeline, where nine known inhibitors (positive controls) are mixed with random compounds to form a database of 1,000 compounds. Using the DOVIS pipeline with the best scoring function, we can find the nine known inhibitors in the top 20% of compounds retrieved from the database known (Figure 1). Although it is difficult to predict which scoring function performs

best for a particular system, if positive controls are available, users can run a small scale screening first to determine the best scoring function before conducting screening of a large database.

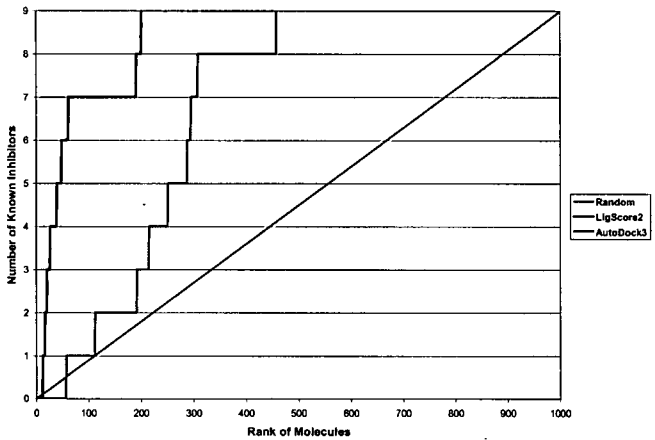

Figure 1. Recovery of know inhibitors of thymidine kinase from a virtual screening exercise

Furthermore, US Army users at US Army Medical Research Institute for Infectious Diseases (USAMRIID) used the DOVIS pipeline to screen 2.3 million compounds from the ZINC database against the ricin A chain protein—a toxin that could be used in bioterrorism. The run was fully automated using 256 CPUs and consumed a total of approximately 77,000 CPU hours.

5. Summary

Using AutoDock as its docking engine, DOVIS helps Department of Defense (DoD) scientists identify potential therapeutics against biological threats for force protection. The DOVIS pipeline provides an automated parallel docking package that is integrated with a queuing system. This application is suitable for conducting large-scale high-throughput virtual screening on Linux cluster platforms.

Disclaimer

The opinions or assertions contained herein are the private views of the authors and are not to be construed as official or as reflecting the views of the US Army or the US DoD. This paper has been approved for public release; distribution is unlimited.

Acknowledgements

This work was sponsored by the US DoD High Performance Computing Modernization Program under the High Performance Computing Software Applications Institutes initiative.

The authors are thankful for Gary Kedziora from the User Productivity Enhancement and Technology Transfer for his help on the modification of the AutoDock source code, and Michael Lee and Mark Olson from USAMRIID for general discussion.

References

Friesner, R.A., J.L. Banks, R.B. Murphy, T.A. Halgren, J.J. Klicic, D.T. Mainz, M.P. Repasky, E.H. Knoll, M. Shelley, J.K. Perry, D.E. Shaw, P. Francis, and P.S. Shenkin, "Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy." *J. Med. Chem.*, 47, pp. 1739–1749, 2004.

Ghosh, S., A.H. Nie, J. An, and Z. Huang, "Structure-based virtual screening for drug discovery." *Curr. Opin. Chem. Biol.*, 10, pp. 194–202, 2006.

Irwin, J.J. and B.K. Shoichet, "ZINC--a free database of commercially available compounds for virtual screening." *J. Chem. Inf. Model.*, 45, pp. 177--182, 2005.

Morris, G.M., D.S. Goodsell, R.S. Halliday, R. Huey, W.E. Hart, R.K. Belew, and A.J. Olson, "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function." *J. Comput. Chem.*, 19, pp. 1639–1662, 1998.

Rarey, M., B. Kramer, T. Lengauer, "Time-efficient docking of flexible ligands into active sites of proteins." *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, 3, pp. 300–308, 1995.

Venkatachalam, C.M., X. Jiang, T. Oldfield, and M. Waldman, "LigandFit: a novel method for the shape-directed rapid docking of ligands to protein active sites." *J. Mol. Graph. Model.*, 21, pp. 289–307, 2003.

Vaque, M., A. Arola, C. Aliagas, and G. Pujadas, "BDT: an easy-to-use front-end application for automation of massive docking tasks and complex docking strategies with AutoDock." *Bioinformatics,* 22, pp. 1803–1804, 2006.