Supporting Information

Parallel implementations of docking programs SOLGRID & SOL

Due to large computational demands for virtual screening of molecular databases the implementation of parallel multi-processor facilities into docking tools is extremely required. Programs GOLD[1], FlexX & FlexE[2] can use PVM (Parallel Virtual Machine) for simultaneous docking of many ligands on UNIX-clusters. In this case parallel computations are based on master-slave scheme, where one (indivisible) task for one machine is docking of one ligand from beginning to the end. Program ICM[3] uses PBS (Portable Batch System) for analogous purposes. Program DOCK[4] can use MPI through MPICH and MPICH2 libraries[5]. In this case parallel computations again based on master-slave scheme with the elemental task of one ligand docking. Virtual screening with DOCK can be started also with grid technology[6].

AutoDock can be used in various grid systems, such as FightAIDS@Home[7] and Discover Dengue Drugs–Together[8]. DOVIS[9,10], GriDock[11] and other add-ons[12] can be used with AutoDock for Linux-clusters; VSDocker[13] can be used with AutoDock for Windows-clusters. These systems are created for maximal automation of docking, including preparing of input data (ligands and proteins) and sorting output data. AutoDock in these systems is used as executable unit without parallelization inside it. Program AutoDock Vina can use multicore processors, simultaneously performing optimization of different ligand poses[14,15]. Parallel docking on clusters with AutoDock was implemented in ref.[16], through the simultaneous execution of independent runs of Lamarckian GA. Acceleration is close to linear when the number of processors is in the range 1-96. Program AutoGrid for grid potentials construction was not parallelized in this work. Parallel version of AutoDock for CUDA also exists[17], however practically significant results have not been published yet.

We have made MPI parallel implementations for one ligand docking by the program SOL as well as for SOLGRID generating grids of protein-ligand interaction potentials. Our tests were carried out on cluster supercomputer Chebyshev, where each node had two 4-cores processors

Intel Xeon E5472 3.0 GHz and 8 GB of RAM and InfiniBand DDR (Mellanox ConnectX) with latency of 1.3-1.95 microsecond and bandwidth of 1540 MB per second was used as MPI network.

Parallel implementation of SOLGRID

Potentials in each point are calculated independently. There are 101*101*101 = 1030301 space points, where potentials have to be calculated, so it is excellent opportunity for parallel calculations. One process is master and does not calculate potentials. Other processes are slaves. Master process divides grid to many slices, and dynamically distributes these slices one by one to slaves. Number of slices is much greater than number of slaves. But time of calculation potentials in one slice is much greater than MPI latency. Each slave calculates potentials in its area, then sends results to master process, and then receives new area until all potentials are calculated.

Here and below we characterize parallel implementation in practical aspect by acceleration $a(N)$ and efficiency $e(N)$. Acceleration $a(N)=t_1/t_N$, where t_1 is calculation time with one calculating process and t_N is calculation time with N calculating processes. Efficiency $e(N)=a(N)/N$. Ideally, the efficiency is equal to 1.

Figure 1 shows the acceleration and efficiency of the program SOLGRID on the number of processes in the range of 1-128.

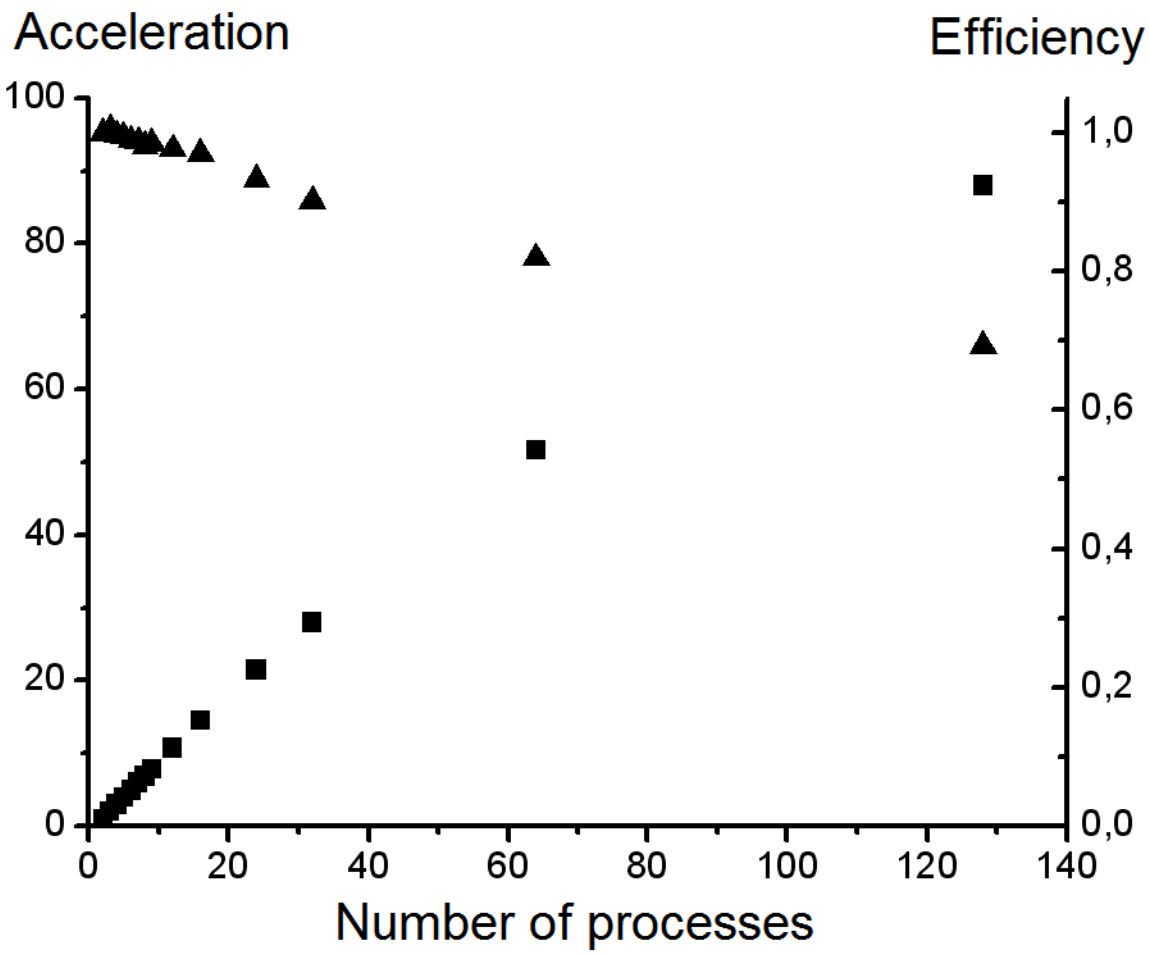

Figure 1. Dependence of acceleration (square points) and efficiency (triangular point) on the number of processes (N) of the program SOLGRID

As can be seen from Figure 1, acceleration of SOLGRID remains almost linear up to 128 processes. Thus, the calculation of the grid potentials on 100 cores will take a few minutes instead of hours required to calculate the grid potentials on one core. Moreover, the SOLGRID parallel version allows creating grid potentials directly on a cluster within a few minutes instead of transferring it through, usually slow, Internet connection.

Parallel implementation of SOL

Independent runs of genetic algorithm (GA) can be performed simultaneously. But we also implemented parallel execution of one run of GA. The basic idea in parallel execution of one GA run is that different ligand poses can be processed independently. There is an array of ligand

poses on each GA iteration, these poses are modifications of previous successful poses; this array is called “population”. There are about 30000 ligand poses in the population. Processing of the population is about 80-90% of the overall time of GA performance.

So, independent GA runs are simultaneously performed by groups of processes, and processes in each group simultaneously treat the population. We have suggested two approaches to parallel processing of the population: first approach with distributed storage of the population in processes, second approach with centralized storage of the population in memory of one process.

Parallel implementation of SOL with distributed storage of the population

The population is equally divided between the processes of one group. Each process handles only its slice of population, but synchronization is required to choose best poses in the entire population. It is a bottleneck of this approach. Also processing of independent GA runs cannot be easily switched between groups of processes. So it is necessary to organize execution of independent GA runs in such a way that it will be done by the same time. For example, if we have 50 independent GA runs and 30 processes, the best solution is to simultaneously execute 30 GA runs (with 30/30=1 process per run), then simultaneously execute 15 GA runs (with 30/15=2 processes per run), then simultaneously execute remaining 5 GA runs (with 30/5=6 processes per run). Such planning is performed by searching combination with lowest estimating time.

Figure 2 shows the acceleration and efficiency of the program SOL with distributed storage of population on the number of processes in the range of 1-1024.

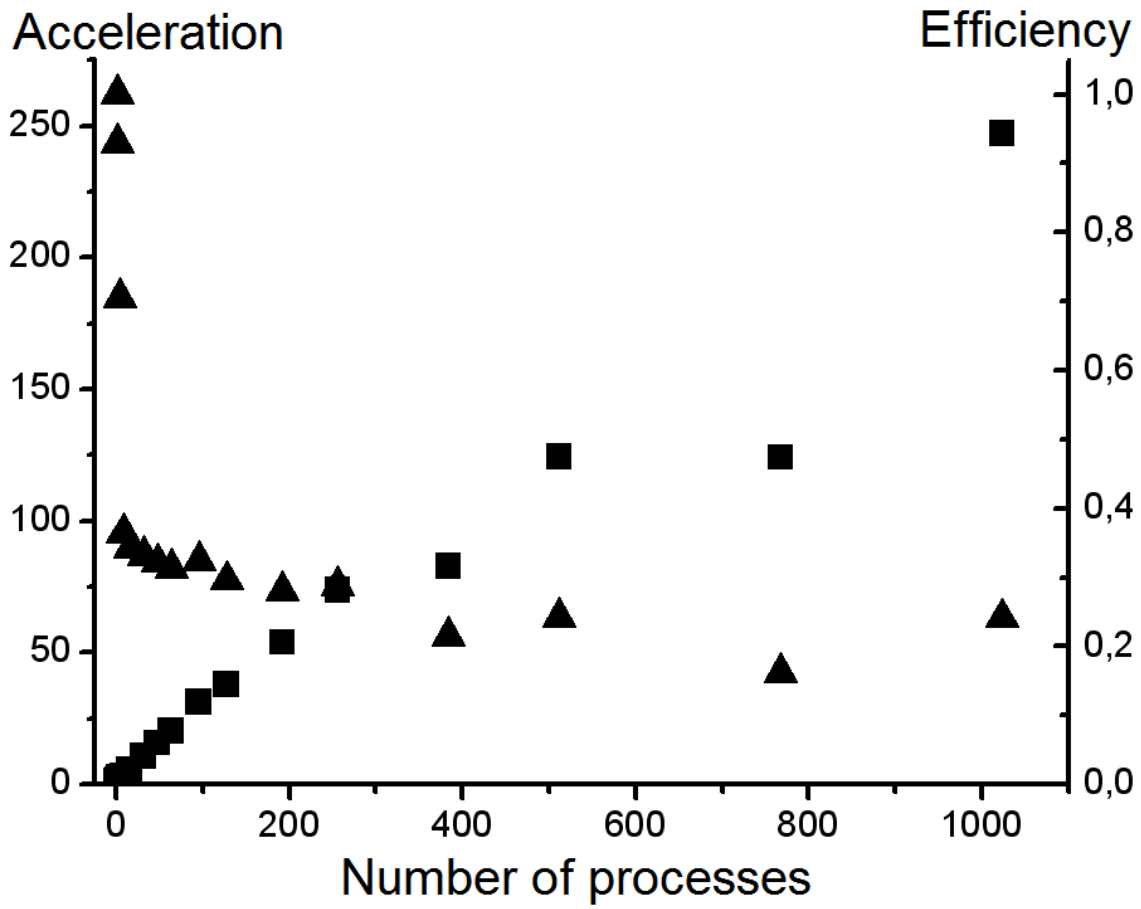

Figure 2. Dependence of acceleration (square points) and efficiency (triangular point) on the number of processes (*N*) of program SOL with distributed storage of population.

We can see that efficiency decreases rapidly from 100% to 40% when number of processes changes from 1 to 16, then efficiency slowly decreases from 40% to 20% when number of processes changes from 8 to 1024.

Parallel implementation of SOL with centralized **storage of the population**

In this realization population is stored only in one process in group (main process). This process alone finds best ligand poses, then transfer this poses to other processes in group. Then processes are continuously making slices of populations and send them to main process. Main process is also making slices of population. When the new population is fully filled by these slices, the iteration is repeated.

This approach allows switching different GA runs between different groups of processes. There is master process, containing all best poses from all independent GA runs, and this process

from time to time exchanging poses between independent GA runs. So number of groups of processes is not necessary equal to number of independent GA runs.

Figure 3 shows the acceleration and efficiency of the program SOL with local storage of population on the number of processes in the range of 1-1024.

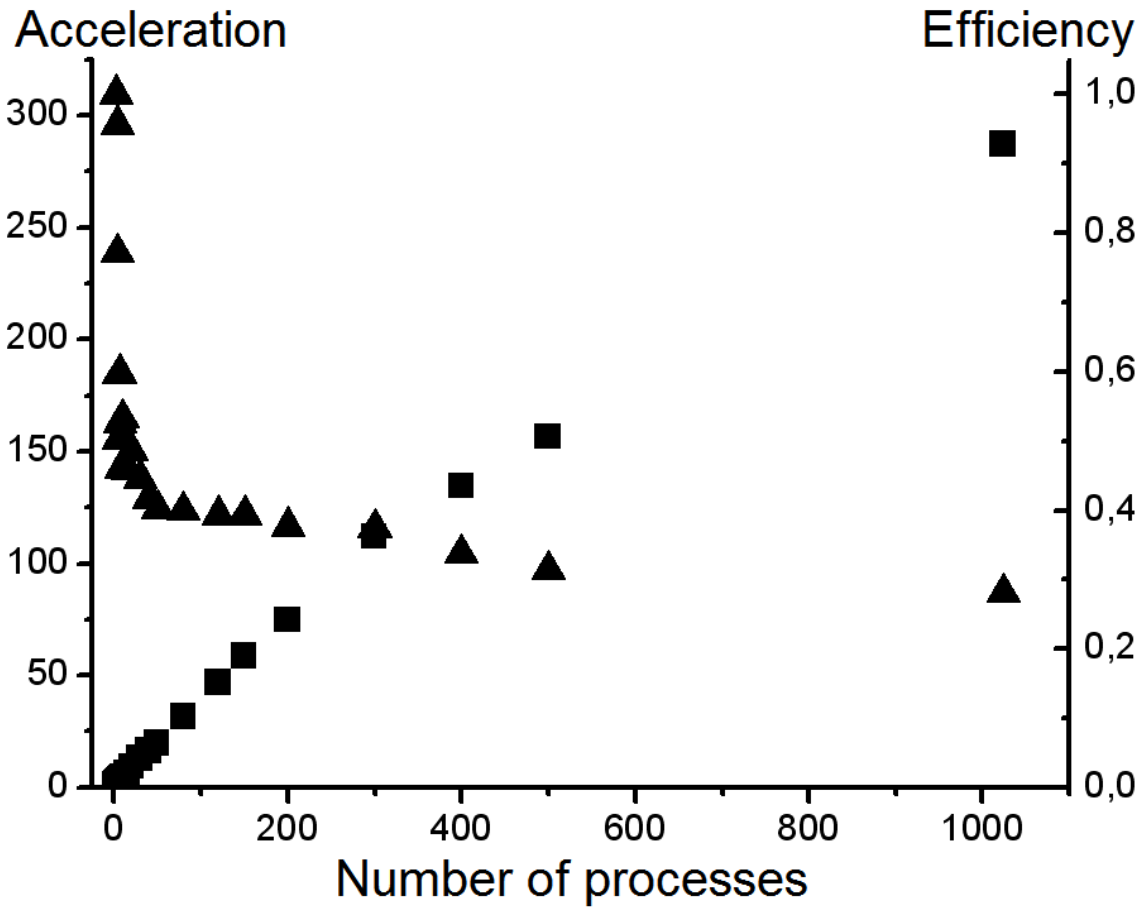

Figure 3. Dependence of acceleration (square points) and efficiency (triangular point) on the number of processes (N) of program SOL with centralized storage of the population.

We can see that efficiency again decreases rapidly from 100% to 40% when number of processes changes from 1 to 16, then efficiency slowly decreases from 40% to 30% when number of processes changes from 8 to 1024.

Thereby for both parallel versions of SOL near linear scalability is observed in the range of 64-1024 processes with efficiency about 20-40%. For mass screening it is better to execute many serial docking programs on the cluster. However these parallel versions of SOL can be useful for

choosing docking parameters before mass screening or for analyzing particular ligand-protein complexes.

REFERENCES

(1) Verdonk, M.L.; Cole, J.C.; Hartshorn, M.J.; Murray, C.W.; Taylor, R.D. Improved protein-ligand docking using GOLD. *Proteins*, **2003**, *52*, 609–623.

(2) Claussen, H.; Buning, C.; Rarey, M.; Lengauer, T. FlexE: efficient molecular docking considering protein structure variations. *J. Mol. Biol*., **2001**, 308, 377–395.

(3) Abagyan, R.; Totrov, M.; Kuznetsov, D. ICM - a new method for protein modeling and design: applications to docking and structure prediction from the distorted native conformation. *J. Comp.Chem*., **1994**, *15*, 488–506.

(4) Ewing, T.J.; Makino, S.; Skillman, A.G.; Kuntz, I.D. DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases. *J. Comput. Aided Mol. Des*., **2001**, *15*, 411–428.

(5) Moustakas, D.T.; Lang, P.T.; Pegg, S.; Pettersen, E.; Kuntz, I.D.; Brooijmans, N.; Rizzo, R.C. Development and validation of a modular, extensible docking program: DOCK 5. *J. Comput. Aided Mol. Des.*, **2006**, *20*(10-11), 601–619.

(6) Levesque, M.J.; Ichikawa, K.; Date, S.; Haga, J.H. Design of a grid service-based platform for in silico protein-ligand screenings. *Comput. Methods Programs Biomed*., **2009**, *93*(1), 73–82.

(7) The Scripps Research Institute: FightAIDS@Home. http://fightaidsathome.scripps.edu

(8) World Community Grid: Discover Dengue Drugs–Together. http://www.worldcommunitygrid.org/research/dddt/overview.do

(9) Zhang, Sh.; Kumar, K.; Jiang, X.; Wallqvist, A.; Reifman, J. DOVIS: an implementation for high-throughput virtual screening using AutoDock. *BMC Bionformatics*, **2008**, *8*, 126–129.

(10) Zhang, Sh.; Kumar, K.; Hu X.; Wallqvist, A.; Reifman, J. DOVIS 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0. *Chem. Central J.*, **2008**, *2*, 18–24.

(11) Vistoli, G.; Pedretti, A.; Mazzolari, A.; Testa, B. Homology modeling and metabolism prediction of human carboxylesterase-2 using docking analyses by GriDock: a parallelized tool based on AutoDock 4.0. *J. Comput. Aided Mol. Des.*, **2010**, *24*(9), 771–787.

(12) Morris, G.M.; Goodsell, D.S.; Huey, R.; Olson, A.J. Distributed automated docking of flexible ligands to proteins: parallel applications of AutoDock 2.4. *J. Comput. Aided Mol. Des*., **1996**, *10*(4), 293–304.

(13) Prakhov, N.D.; Chernorudskiy, A.L.; Gainullin, M.R. VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters. *Bioinformatics*, **2010**, *26*, 1374–1375.

(14) Trott, O.; Olson, A.J. AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *J. Comp. Chem*., **2010**, *31*(2), 455–461.

(15) Chang, M.W.; Ayeni, C.; Breuer, S.; Torbett, B.E. Virtual screening for HIV protease inhibitors: a comparison of AutoDock 4 and Vina. PLoS One, **2010**, 5(8): e11955. doi:10.1371/journal.pone.0011955. .

(16) Khodade, P.; Prabhu, R.; Chandra, N.; Rahab, S.; Govindarajanb, R. Parallel implementation of AutoDock. *J. Appl. Cryst*., **2007**, *40*, 598–599.

(17) AutoDock Software in Parallel with GPUs: Project Web Hosting - Open Source Software. http://gpuautodock.sourceforge.net/