HHS Public Access

Author manuscript

J Chem Theory Comput. Author manuscript; available in PMC 2022 February 09.

Published in final edited form as:

J Chem Theory Comput. 2021 February 09; 17(2): 1060–1073. doi:10.1021/acs.jctc.0c01006.

Accelerating AutoDock4 with GPUs and Gradient-Based Local Search

Diogo Santos-Martins[1,a], **Leonardo Solis-Vasquez**[1,b], **Andreas F Tillack**[a], **Michel F Sanner**[a], **Andreas Koch**[b], **Stefano Forli**[a]

[a]Department of Integrative Structural and Computational Biology, The Scripps Research Institute, La Jolla, California, USA

[b]Embedded Systems and Applications Group, Technical University of Darmstadt, Germany

Abstract

AutoDock4 is a widely used program for docking small molecules to macromolecular targets. It describes ligand-receptor interactions using a physics-inspired scoring function that has been proven useful in a variety of drug discovery projects. However, compared to more modern and recent software, AutoDock4 has longer execution times, limiting its applicability to large scale dockings. To address this problem, we describe an OpenCL implementation of AutoDock4, called AutoDock-GPU, that leverages the highly parallel architecture of GPU hardware to reduce docking runtime by up to 350-fold with respect to a single-threaded process. Moreover, we introduce the gradient-based local search method ADADELTA, as well as an improved version of the Solis-Wets random optimizer from AutoDock4. These efficient local search algorithms significantly reduce the number of calls to the scoring function that are needed to produce good results. The improvements reported here, both in terms of docking throughput and search efficiency, facilitate the use of the AutoDock4 scoring function in large scale virtual screening.

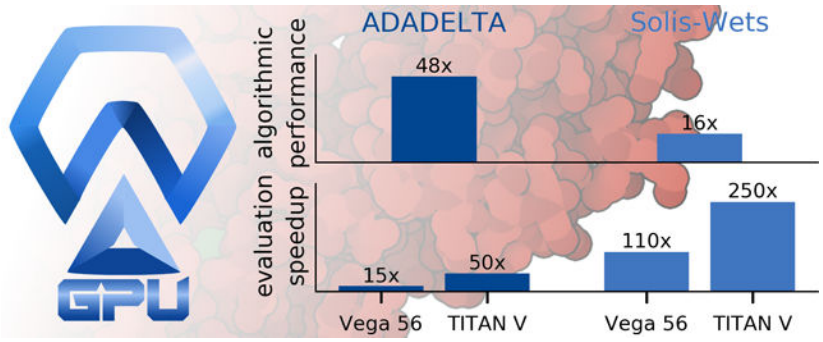
Graphical Abstract

AUTODOCK-GPU is open source under a GPL licence. Its source code as well as its documentation is available at: https://github.com/ccsb-scripps/AutoDock-GPU The experiments reported in this paper were performed with AUTODOCK-GPU v1.2.

Correspondence: koch@esa.tu-darmstadt.de forli@scripps.edu.

[1]These authors contributed equally to this work.

ASSOCIATED CONTENT. The supporting information file (.pdf) introduces the OpenCL programing model, describes the code architecture of AUTODOCK-GPU, discusses porting to FPGAs, highlights differences from AUTODOCK4, discusses the memory requirements, analyzes the cost efficiency on cloud computing, describes the tuning of ADADELTA parameters, and compares the algorithmic efficiency of of Solis-Wets in AUTODOCK4 and AUTODOCK-GPU. This information is available free of charge via the Internet at http://pubs.acs.org



Keywords

Molecular Docking; AutoDock; High Performance Computing; GPU

INTRODUCTION

The identification of new molecules with a given biological activity is a non-trivial task, and a key step in the drug discovery process. Structure-based virtual screening in which small organic molecules are docked to biologically and therapeutically relevant targets is a very popular computational technique for the identification of new chemical scaffolds with biological activity.

Over the years, this technique has provided numerous positive results competitive with experimental screenings[1–5]. The key to successful dockings is the use of energy estimate methods (or scoring functions) that are sufficiently accurate to distinguish (to a certain extent) between binders and non-binders, while being sufficiently fast to be applicable on very large scales. Typical docking virtual screenings now can easily involve virtual libraries from few thousand to hundreds of millions of molecules[6] usually against rigid target structures. Depending on the search algorithms, this translates into 10^{10} to 10^{16} calls to the scoring function per screening, but this number can further grow by orders of magnitude if conformational ensembles[7] are used to represent receptor conformational variability.

From a computational perspective, molecular docking is well suited for parallel computing given the parallel nature of the tasks involved in evaluating multiple solutions for a single ligand, as well as docking large libraries of ligands. Scoring function calls can be distributed across compute units, either being CPU cores on the same machine[8], many CPU nodes across multiple machines in high-performance computing environments[9], or even distributed computing resources[10,11]. From that perspective, GPU computing represents a hybrid scenario in which a large number of compute units can be accessed on a single machine. GPU cards with thousands of cores are a powerful resource that is already being exploited to accelerate a number of molecular modeling computations[12–14]. GPU computing is worth pursuing even considering that the distribution of work by multiple computing units, data transfer and the coordination of the different operations can create unexpected bottlenecks that make the parallelization process non-trivial.

The AutoDock suite consists of widely used molecular docking programs[15,16], which include a number of docking engines derived from the original AutoDock code[17], and specialize in different aspects of the docking simulations. AutoDock Vina implements a Monte-Carlo-based iterated local search method, designed to provide speed performance exploiting CPU parallelism for virtual screenings[8]. AutoDockFR focuses on peptide docking and on simulating various degrees of receptor flexibility[18]. AutoDock4 implements a single-threaded Lamarckian Genetic Algorithm (LGA)[19] search engine, which is usually parallelized by submitting multiple individual docking jobs on separate CPUs or cores. While being slower than AutoDock Vina, AutoDock4 has been reported to be more effective in some cases[20], especially at reproducing experimental binding affinities[21]. Moreover, AutoDock4 is more versatile and easier to customize without modifying the source code, allowing us and others to modify the force field[22–29] and grid interaction maps[30] to add new custom potentials, and implement new docking protocols[31–34].

In the past, there have been attempts to exploit GPUs for accelerating different steps of the AutoDock4 docking protocol with various degrees of success[35–37]. By exploiting to different extents the embarrassingly parallel nature of docking and the underlying algorithms, its efficient adaption to hardware accelerators through parallel programming frameworks[35,38–40] can result in significant performance improvements. Among these programming frameworks, the Open Computing Language (OpenCL) provides an open standard that is portable across hybrid platforms such as CPUs and GPUs.

In this work, we describe our OpenCL implementation of AutoDock to explore the performance improvements resulting from adapting the AutoDock4 docking engine to exploit both GPU and CPU parallel architectures. This program uses a minor variation of the AutoDock LGA algorithm, and introduces derivatives for translation, rotation, and torsion variables. The local search is performed using either the original random optimizer Solis-Wets (SW)[41], or the newly implemented ADADELTA[42] gradient-based local search. The program generates output docking log files (*DLG*) files compatible with AutoDockTools 19. Using the single-threaded AutoDock4.2.6 as baseline, we characterize the performance of the new OpenCL implementation on both CPU and GPU platforms.

METHODOLOGY

AutoDock docking method.

In AutoDock, molecular interactions are modeled by a scoring function f that quantifies the free energy ΔG of a given binding pose[43]:

$$\begin{aligned}\Delta G = {} & \Delta H_{vdW} \quad W_{vdW} \sum_{i,j} \left(\frac{A_{ij}}{s(r_{ij})^{12}} - \frac{B_{ij}}{s(r_{ij})^{6}} \right) + \Delta H_{hbond} \quad W_{hb} \sum_{i,j} E(t \\ &)\left(\frac{C_{ij}}{s(r_{ij})^{12}} - \frac{D_{ij}}{s(r_{ij})^{10}} \right) + \Delta H_{elec} \quad W_{el} \sum_{i,j} \left(\frac{q_i q_j}{\varepsilon(r_{ij}) r_{ij}} \right) \\ & + \Delta G_{desolv} \quad W_{desolv} \sum_{i,j} (S_i V_j + S_j V_i) e^{\left(-r_{ij}^2 / 2\sigma^2\right)} + \Delta S_{tor} \quad W N_{tor}\end{aligned} \tag{1}$$

The AutoDock scoring function (Equation 1) is a semi-empirical free-energy force field (kcal/mol) composed of four pairwise energetic terms (dispersion/repulsion ΔH_{vdW}, hydrogen bonding ΔH_{hbond}, electrostatics ΔH_{elec}, and desolvation ΔG_{desolv}), and an additional term that predicts the unfavorable loss of ligand entropy upon binding (ΔS_{tor}). For each term, a dimensionless weighting constant was empirically determined using linear regression on a set of ligand-receptor complexes with known binding constants[44]. The following constants depend on the atom types: A_{ij} (kcal/mol/Å^{12}) and B_{ij} (kcal/mol/Å^{6}) correspond to a modified Lennard-Jones (12–6) potential between atoms i and j; C_{ij} (kcal/mol/Å^{12}) and D_{ij} (kcal/mol/Å^{10}) correspond to the hydrogen bonding (12–10) potential between hydrogen-bond acceptor and donor atoms i and j; S and V are the solvation parameter and atom volume, respectively, and σ is set to 3.5 Å. The $s(r_{ij})$ function is used to broaden the attractive wells in 12–6 and 12–10 potentials and is referred to as *smoothing* in the AutoDock documentation. The $E(t)$ function provides directionality to the hydrogen bond term based on the t angle. Additionally, q_i and q_j are atomic charges, while $\varepsilon(r_{ij})$ is a dielectric function of r_{ij}, the interatomic distance between atoms i and j. The last term, proportional to the number of torsions N_{rot}, measures the unfavorable loss of ligand entropy due to the restriction of conformational degrees of freedom in the bound state.

Interactions between the ligand and the receptor are precalculated and cached using the AutoGrid program, which calculates interaction energy maps for ligand atoms, with a default resolution of 0.375 Å. During docking, these three-dimensional maps are used to lookup interaction energies using trilinear interpolation, speeding up intermolecular interaction estimates compared to pairwise methods.

A Lamarckian Genetic Algorithm (LGA) is used to generate ligand poses to explore the energy landscape described by the scoring function[44]. The LGA combines a genetic algorithm (GA) global search with local search (LS) iterations to refine poses identified by the GA. Poses that are improved by the LS method are re-introduced into the GA population (hence the Lamarckian denomination). The collection of poses forms the GA population. Each pose is represented by a vector of genes, referred to as *genotype*. Each gene is a variable representing ligand position, orientation, and torsional conformation. At each GA iteration, new individuals are generated from ancestors chosen with a proportional selection scheme, while a population subset (default: 6%) is subjected to the LS optimization. The execution of each independent LGA run stops when either the number of score evaluations or the number of GA generations is reached, whichever comes first. These numbers are specified by the user. The pose with the best score is returned by the LGA.

Furthermore, a docking job consists of several independent LGA runs (50 by default in AutoDock4). Each LGA run optimizes the sum of intermolecular (ligand-receptor) and intramolecular (ligand-ligand) interactions described by the scoring function f, and returns the best pose (lowest score) found. After all LGA runs are completed, the returned poses are clustered using the root mean square deviation (RMSD) as distance metric. The size of the cluster containing the pose with the best score is commonly used as a proxy to identify convergence of the LGA runs towards a particular solution (pose) within the search space. As a general rule, the search effort is considered sufficient if the first cluster contains at least 20% of the poses returned by LGA runs.

OpenCL implementation of AutoDock.

A single docking job requires a large number of scoring function evaluations, usually between 10^6 and 10^8. Thus, scoring function evaluations become the bottleneck, accounting for more than 90% of the runtime[45]. Therefore, offloading these steps onto accelerators, such as GPUs, can benefit significantly from the speedup provided by such specialized hardware.

The improvements of the AutoDock search algorithm introduced in this paper were built on top of our previous work[45], released as the open-source software project *OpenCL Accelerated Molecular Docking*, hereafter simply referred to as AutoDock-GPU[46]. The code has been implemented using OpenCL v.1.2. OpenCL provides an open and royalty-free standard for writing parallel programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other hardware accelerators, providing software developers with a portable abstraction for accelerating computationally-intensive tasks[47]. As reported in a previous study[48], data parallelism of AutoDock is exploited at three different processing levels: *high*, *medium*, and *low*. First, based on the fact that a docking job consists of several independent LGA runs, the high-level parallelization consists of trivially performing these runs in parallel (similarly to what is done automatically in AutoDock Vina 8). Then, medium-level parallelization is achieved by processing individuals from a single genetic generation within an LGA run independently. Finally, the low-level strategy consists of the parallel execution of fine-grained tasks that pertain to a single individual, such as rotating the pose or evaluating the scoring function.

Our implementation (Figure 1) combines the high- and medium-level strategies first. A docking job is composed of R LGA runs (Run_{ID}: 0, 1, 2, …, $R-1$), each run processing a population of P individuals (Ind_{ID}: 0, 1, 2, …, $P-1$). The execution of such runs, and the processing of their individuals, are controlled by nested loops in the serial implementation, but can be merged into a single loop for optimal parallelism. By doing so, individuals from different docking runs can be processed simultaneously, each as an OpenCL work-group identified with a WG_{ID} obtained as follows:

$$\text{WG}_{\text{ID}} = \text{Run}_{\text{ID}} \times P + \text{Ind}_{\text{ID}} \tag{2}$$

The entire set of $P \times R$ work-groups is distributed by the GPU runtime scheduler over the available compute units (CU). Each CU executes a WG associated with a single individual, achieving high- and medium-level parallelization simultaneously. Finally, the processing an individual, which involves fine-grain tasks (genotype generation, ligand rotation, intermolecular and pairwise interactions), can be assigned to OpenCL work-items, thus achieving low-level parallelization.

The overall workflow of AutoDock-GPU is depicted in Figure 2. This consists of a sequence of functions executed either on the host (H) or on the device (D). After the application inputs are parsed in H1, the populations of all docking runs are initialized with random values in H2. Then, the OpenCL setup takes place in H3, which comprises the identification and selection of platform and device, and the creation of the kernels to be executed on the device.

In order to guarantee that the limited memory capacity of a GPU card (i.e., compared to most multi-core CPU servers) does not negatively impact the practical usage of AutoDock-GPU, we analyzed the memory footprint required to hold the processing data.

First, for the released version we defined hard-coded limits for AutoDock-GPU the maximum values allowed for the following runtime parameters: ligand atomic types, ligand atoms, rotatable bonds, pairwise contributors, rotations, population size, docking runs, and grid points (see Table S2 for the values). While such constraints might limit the upper-bound size of systems that can be modeled with AutoDock-GPU, they guarantee that data allocation is within ~ 4 GB. These values can be easily modified when compiling the code, if more performant hardware is available. Further details on the software architecture and memory requirement analysis are provided in the Supplementary Information.

Gradient of the scoring function.

The process of docking a ligand is implemented as an optimization problem in which the collection of variables subject to optimization define the conformation, orientation and position of the ligand. In the GA metaphor, individual variables are called *genes*, and the collection of genes defining a solution constitute a *genotype* Ω:

$$\Omega = \left\{x, y, z, \phi, \theta, \alpha, \psi_1, \ldots, \psi_{N_{\text{rot}}}\right\} \tag{3}$$

The first three variables control translation of the ligand in x, y, and z directions; the next three variables – ϕ, θ, and α – correspond to the rotation of the ligand as a rigid body, and the remaining ψ variables are associated with N_{rot} rotatable bonds. Each genotype corresponds to a different pose of the ligand. The quality of a pose, from the energetic standpoint, is evaluated by the scoring function in Equation 1, which is expressed in terms of the optimization variables (genes).

$$f\left(x, y, z, \phi, \theta, \alpha, \psi_1, \ldots, \psi_{N_{\text{rot}}}\right) \tag{4}$$

Docking a ligand consists of finding the genotype Ω corresponding to the lowest possible value of the scoring function f. Here we introduce in the AutoDock4 a new gradient-based optimizers which uses the gradient g of the scoring function f, with respect to the genotype Ω:

$$g = \nabla f(\Omega) = \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \phi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \alpha}, \frac{\partial f}{\partial \psi_1}, \ldots, \frac{\partial f}{\partial \psi_{N_{\text{rot}}}}, \tag{5}$$

ADADELTA local search.

ADADELTA[42] is a gradient-based first-order local optimization algorithm, which is used to update the genotype Ω at each iteration t:

$$\Omega_{t+1} = \Omega_t + \Delta\Omega_t \tag{6}$$

where $\Delta\Omega_t$ is the update vector, which depends not only on the gradient g, but also on the history of past gradients and past update vectors:

$$\Delta\Omega_t = -\frac{\sqrt{E\left[\Delta\Omega^2\right]_{t-1} + \varepsilon}}{\sqrt{E\left[g^2\right]_t + \varepsilon}} g_t \tag{7}$$

where $E[\Delta\Omega^2]$ is a running average of squared updates, and $E[g^2]$ is a running average of squared gradients:

$$E\left[\Delta\Omega^2\right]_t = \rho E\left[\Delta\Omega^2\right]_{t-1} + (1-\rho)\Delta\Omega_t^2 \tag{8}$$

$$E\left[g^2\right]_t = \rho E\left[g^2\right]_{t-1} + (1-\rho) g_t^2 \tag{9}$$

where ρ and ε are ADADELTA hyperparameters. The constant ε prevents the denominator in Equation 7 from becoming zero, providing a non-zero update at the first iteration ($t = 0$). The running averages $E[g^2]$ and $E[\Delta\Omega^2]$ are vectors of length equal to the number of *genetic* variables. The update rule is applied to each variable separately, implementing a different learning rate for each dimension. The hyperparameter ρ controls the amount of *memory* of preceeding iterations for $E[g^2]$ and $E[\Delta\Omega^2]$. Smaller ρ make the running averages more sensitive to the current iteration.

ADADELTA belongs to the class of first-order algorithms, which include Adam[49] and FIRE[50]. One of the advantages of first-order algorithms is their simpler implementation in comparison to Quasi-Newton methods, such as BFGS[51].

Among the first-order algorithms, we chose ADADELTA because it requires only two hyperparameters (ρ and ε) (compared to Adam and FIRE, which use four and five, respectively). The optimization of ADADELTA's ρ and ε for AutoDock-GPU is discussed in the Supplementary Information. We found that $\rho = 0.8$ and $\varepsilon = 0.01$ provide the best search performance (Fig. S3).

Dataset.

To compare the performance of the OpenCL implementation with AutoDock4, we compiled a dataset of 140 diverse protein-ligand complexes covering a wide range of properties for both ligands (rotatable bonds, atom count, etc.) and targets (shape of binding site). The dataset was built including complexes from two well-established sets for assessing docking methodologies for drug discovery purposes, the Astex Diversity Set[52] and CASF-2013[53].

In particular, the Astex Diversity Set, which is comprised of 85 complexes mostly with up to 10 rotatable bonds, was extended with complexes from the CASF-2013 to increase the number of ligands with 11 to 20 rotatable bonds. Finally, 20 additional complexes selected from the Protein Data Bank[54] were included to add ligands with up to 32 rotatable bonds, and with diverse binding site shape (i.e., wide, narrow, deep, shallow sites). The dataset is available at https://doi.org/10.5281/zenodo.4031961. Ligand and receptor input files were

prepared with OpenBabel (version 2.4.1)[55] after removing all water molecules and ions according to the standard AutoDock protocol[9].

Docking calculations.—Reference dockings for the standard single-threaded CPU baseline were performed using the latest stable version of AutoDock (v.4.2.6). For the OpenCL implementation, calculations on the GPU were performed using the CUDA v10.0 and AMDAPPSDK v3.0 packages for NVIDIA and AMD architectures, respectively. The size of the work group for GPU platforms was 64. GNU parallel[56] was used to run on machines with multiple GPUs. To test performance on multicore CPU machines, the Intel SDK-2017 driver was used, with a work group size of 16.

To account for the stochastic nature of the LGA search, four replicates of 100 LGA runs were performed for each ligand-receptor complex. Generally, a smaller number of LGA runs (e.g., 20) is sufficient with AutoDock-GPU, but we used 100 to have better statistics. Each of the four replicates uses different ligand input coordinates, one with the experimental coordinates of the ligand in the complex to check for input biases, and three with randomized poses. The top pose from each LGA run was considered, totaling 400 poses per ligand-receptor complex (Figure 3). A statistical measure of success was then calculated on this large pool of docked poses. The length of each LGA run was defined by the number of evaluations (i.e., calls to the scoring function), which ranged between 32,000 and 8,192,000. To guarantee that the desired number of score evaluations is performed, we set the maximum number of generations to a very high value (99999). The maximum number of local search iterations was set to 300.

Global minimum identification.

The identification of true global minima for each system is a prohibitive task requiring systematic scanning of all degrees of freedom. Therefore, we designed a protocol to identify *bona fide* global minima by analyzing the distribution of scores produced by LGA runs (Figure 3). For each protein-ligand complex, first we performed 4 docking replicates of 100 LGA runs each (400 total poses), starting from different input conformations, with 8,192,000 maximum energy evaluations. From each replicate, we retrieved the 5 top scores creating a pool of 20 poses. Then, the procedure was repeated with 4 docking replicates with 4,096,000 evaluations, and the best pose from each was added to the poses pool, generating 24 poses in total. If the difference between the best and worst scores within the 24 poses in the pool was ≤ 0.1 kcal/mol, the best score found was considered to be the *bona fide* global minimum.

We found the global minimum for 106 out of 140 complexes. Figure 3 illustrates a system for which the global minimum was found (PDB ID: 1hvy) and a system for which it wasn't (PDB ID: 7cpa). For 1hvy, the scores converge towards a lower bound (depicted by the dashed line).

Measuring the time per score evaluation.

To calculate the time required for a single evaluation (i.e., a call to the scoring function), we run dockings with different numbers of evaluations from 32,000 to 2,048,000 evaluations, in

exponential increments of 2-fold, providing 7 data points that we used to fit a simple linear model. Four replicates were performed for each docking, and considering either minimum or average runtimes led to nearly identical results. We excluded cases for which the coefficient of determination R^2 is less than 0.99. In the Results and Discussion section, we report the time per evaluation for all 140 complexes in our dataset, running AutoDock-GPU on different accelerators, and using either Solis-Wets or ADADELTA as local-search methods.

RESULTS AND DISCUSSION

Validation of the scoring function implementation.

In order to validate the accuracy of the OpenCL implementation of the scoring function, a total of 1015 docked poses generated with AutoDock-GPU from 10 complexes were re-scored using AutoDock4.2.6 (using the keyword *epdb*). For docking poses in the favorable energy range (i.e., negative energies) we observed differences in the order of 0.01 kcal/mol for grid-interpolated interactions, and 0.02 kcal/mol for ligand-ligand (pairwise) interactions. For docked poses with large positive energies, which are typically discarded because they have unfavorable interactions, the differences between AutoDock4.2.6 and AutoDock-GPU can reach the order of 1 kcal/mol. These discrepancies occur because AutoDock4 interpolates values from look-up tables, while AutoDock-GPU calculates exact values by evaluating the analytical form of the scoring function f (Equation 1). Due to the exponential repulsive terms in f, the error in the interpolated values can be large, ultimately showing that in this regard AutoDock-GPU is a more accurate implementation of the AutoDock scoring function (Eq. 1).

Runtime performance on GPUs.

The runtime speedups of AutoDock-GPU achieved on various GPUs was measured with respect to AutoDock4 running on a single CPU core. Runtime is understood as the total *wall clock* time spent by the docking program until completion. We selected five different ligand-receptor complexes covering a wide range of number of atoms (N_{atom}) and rotatable bonds (N_{rot}), as displayed in Figure 4. The number of LGA runs was 100, and the number of evaluations per LGA run was 2,048,000.

Runtime speedups are reported in Figure 4. Overall, when using Solis-Wets for local search, speedups range from about 30x to 350x, depending on the GPU and the system being docked. When using ADADELTA as local-search, the speedups range from about 2x to 80x. ADADELTA is slower than Solis-Wets because the calculation of gradients is computationally expensive and difficult to parallelize. For Solis-Wets, ligands with more atoms yield higher speedups, while the opposite trend is observed for ADADELTA. The higher-end GPU card (TITAN V) achieved about 10x larger runtime speedups than the lower end M2000, in agreement with their characteristics (Table 3).

Performance scaling on multicore CPUs.

A key feature of OpenCL is its portability. This allows running applications on different devices with none or minor recoding effort. In this study, this feature was leveraged to target multicore CPUs as execution platform. Our implementation ensured that the porting effort

from GPUs to CPUs was reduced to re-compiling the application targeting the different device architecture, without code changes.

Performance scaling refers to the performance enhancement due to an increase of computing resources. This is typically translated into shorter runtimes, which in turn can result into lower costs for such resources, when accessed on a per-time basis. In order to assess the resource investment in terms of the returned performance gains, we discuss the performance scaling behavior of AutoDock-GPU on multicore CPUs.

To do that, we selected AWS CPU instances equipped with different numbers of cores: c5.4xlarge (8 cores), c5.9xlarge (16 cores), and c5.18xlarge (36 cores). This analysis was performed on the same complexes used for reporting GPU runtime speedups (Figure 4), and the baseline for speedups in the runtime of AutoDock4.2.6 on a single CPU core.

In contrast to GPUs, where the overall performance when running Solis-Wets increases with N_{atom} (Figure 4), multi-core CPUs yield lower runtime performance for larger N_{atom} (Figure 5). For instance, on an 8-core instance, Solis-Wets achieves 15x of runtime speedup with the smallest complex (PDB ID: 5tim), while only reaching 5.3x with the largest one (PDB ID: 3er5). We attribute the different speedup dependencies on N_{atom} and N_{rot} values between multicore CPU and GPU to the considerably larger number of fine-grained computing cores on GPUs (e.g., 3584 CUDA cores on a GTX 1080 Ti GPU) that more efficiently leverage the larger fine-grained parallelism – provided by more atoms and rotatable bonds – than by using fewer CPU cores (e.g., a maximum of 36 cores on the AWS c5.18xlarge instance).

The data in Figure 5 shows that speedups increase with a factor of ~2x when e.g., migrating from an instance with eight cores (c5.4xlarge) to another with 16 cores (c5.9xlarge). Slightly higher runtime speedup gains are observed when migrating to the largest c5.18xlarge instance, which can be simply explained by the upgrade provided by having more than double the number of cores (= 36). A similar speedup scaling behavior was observed when running ADADELTA. Additionally, Figure 5 shows resource-utilization efficiency, calculated as the ratio between the runtime speedup and the number of CPU cores present in the employed AWS CPU instances. For both LS methods, the resource-utilization efficiencies are higher on smaller instances (fewer cores). Moreover, higher efficiencies are obtained when no rotatable bonds are present (5tim), e.g., ~1.9x (Solis-Wets) and ~0.73x (ADADELTA) on the c5.4xlarge. For the other complexes (having more than seven rotatable bonds), such efficiencies become lower but more stable, i.e., ~0.7x and 0.3x when using Solis-Wets and ADADELTA, respectively. In the supporting information we provide a cost-efficiency analysis of several accelerators, including CPU and GPU, based on the hourly prices of Amazon Web Services (AWS), showing that GPUs are more cost-effective (Figure S2).

Time spent per evaluation of the scoring function.

Docking a single ligand can imply hundreds of millions of calls to the scoring function (i.e., evaluations). For example, performing 100 LGA runs with the standard amount of evaluations per run (2.5 million) results in 250 million total evaluations being executed in a

single docking. For this reason, the average time spent per evaluation almost always dictates docking runtime.

The most important factors affecting the time per evaluation are the accelerator platform, the choice of LS method and the number of ligand atoms (Table 2). The time spent per evaluation increases with the number of ligand atoms, and that held true across all platforms tested (Figure 6). Among the GPU cards in Table 3, AutoDock-GPU achieves the fastest evaluations on the TITAN V, spending between 0.01 – 0.67 μs per evaluation, depending on the protein-ligand complex.

For Solis-Wets, the time per evaluation increases in a nearly linear fashion with the number of ligand atoms (N_{atom}), while for ADADELTA it increases closer to a quadratic fashion (Figure 6 and Table 2). Consequently, ADADELTA becomes progressively slower than Solis-Wets for larger N_{atom} values. For example, ADADELTA is about 4x slower than Solis-Wets for ligands with ~20 atoms, and about 16x slower for ligands with ~100 atoms.

To calculate speedups provided by GPU cards we established a performance baseline by running the single-threaded AutoDock4.2.6 on a recent compute instance (Amazon AWS c5.18xlarge) featuring an Intel Xeon Platinum 8124M CPU @ 3.00GHz (Table 1). Speedups were then calculated by dividing the time per evaluation of AutoDock4 by that of AutoDock-GPU. Figure 7 shows speedup over single-threaded AutoDock4.2.6. Interestingly, for dockings using Solis-Wets, evaluation rate speedups increase with the number of ligand atoms, while the opposite behavior occurs for ADADELTA. Time per evaluation speedups reach 200x faster evaluations when using Solis-Wets on a GTX 1080 Ti for many of the systems in our dataset.

Algorithmic performance of local search methods.

We assessed the efficiency of search algorithms by measuring the quality of docking results as a function of the number of calls to the scoring function (i.e., evaluations). Naturally, better search algorithms are more economic and spend fewer evaluations to produce docking results that meet a certain quality standard. We evaluated the algorithmic performance of Solis-Wets and ADADELTA as implemented in AutoDock-GPU as well as the Solis-Wets implementation in AutoDock4.2.6.

Docking runs success criteria.—Our assessment of search performance starts by determining if individual LGA runs are successful or not. We use two criteria to define success, one based on the docking score, and another based on the root mean square deviation (RMSD) from the native pose determined by X-ray crystallography.

According to the score criterion, an LGA run is successful if the returned pose has a score within 1.0 kcal/mol from the best possible score for a given protein-ligand complex (the global minimum). According to the RMSD criterion, an LGA run is successful if the returned pose is within 2 Å from the native pose.

The protocol for finding global minima in our dataset is described in the Methodology section. We found the global minimum for 106 out of 140 complexes, while for the rest that

was not possible due to the presence of a large number of rotatable bonds, particularly for ligands containing more than 19 of these.

We only use the RMSD criterion when the global minimum of the scoring function is below or equal to 2 Å RMSD of the native pose, which is true for 78 out of 106 complexes (73.58%). We do this because the job of the search algorithm is to find the global minimum of the scoring function. If that global minimum corresponds to an incorrect pose, that is a problem with the scoring function, not the search algorithm. In fact, for 27 out of 106 complexes the global minimum corresponds to a pose with a RMSD greater than 2 Å. If we had used the RMSD criterion with these 27 complexes, finding the global minimum would be incorrectly considered as unsuccessful.

Success rate as a function of the number of evaluations.—The fraction of successful LGA runs increases with the number of evaluations, and its corresponding success-rate curve typically follows a sigmoidal profile (Figure 8). An insufficient number of evaluations results in a low (possibly zero) probability of finding either the native pose or the global minimum. After a certain number of evaluations, the success probability will asymptotically approach 100%.

Half-maximal success rate evaluations (E_{50}).—Success probability curves (Figure 8) can be then used to estimate the *half-maximal success rate evaluations* (E_{50}), which is the number of evaluations required to achieve a 50% probability of success. E_{50} values are calculated by fitting the following sigmoid function to the success rate curves depicted in Figure 8:

$$\text{Fraction of successful LGA runs} = \frac{1}{1 + e^{-\beta(x - E_{50})}} \tag{10}$$

where x is the number of score evaluations, and β and E_{50} are variable parameters optimized during sigmoid fitting.

The values of E_{50} depend on both the intrinsic system complexity (e.g., binding site characteristics, ligand complexity), and the algorithmic efficiency of the search methods. If the global minimum of a given protein-ligand complex is easy to find, high-success probabilities are achieved with a relatively small number of evaluations. Analogously, more efficient search algorithms will require smaller E_{50}, as illustrated in Figure 8, where the top and bottom panels correspond to an easy and moderately difficult case, respectively. The comparison of the success rate curves for the moderately difficult complex (PDB ID: 1y6b) shows that ADADELTA outperforms Solis-Wets, requiring ~4.5x fewer evaluations to achieve a comparable success rate.

Evaluation of ADADELTA and Solis-Wets using E_{50}.—The algorithmic performance of ADADELTA and Solis-Wets was compared by calculating E_{50} values by varying the fraction of the LGA population to which the LS methods were applied during docking: 6% (the AutoDock4.2.6 default), 25%, and 100% (Figure 9). The analysis of the overall results allowed us to build the regression model:

$$E_{50} = 2^{a \times N_{\text{rot}} + b} \quad (11)$$

that fits E_{50} as a function of the number of rotatable bonds (N_{rot}), a proxy for search complexity. The value of the a coefficient can be then used to estimate how E_{50} escalates with the number of rotatable bonds.

For both local search methods, and using either score or RMSD criteria, we observed that higher LS rates result in lower a coefficients, hence reducing the number of evaluations required to achieve better performance for ligands with many rotatable bonds. Therefore, all the subsequent calculations were performed using 100% LS rate, and we recommend using a high local search rate with AutoDock-GPU.

According to the scoring criterion, the regression models predicted E_{50} of $\sim 2^{0.68 N_{\text{rot}} + 5.79}$ for Solis-Wets, and $\sim 2^{0.45 N_{\text{rot}} + 5.88}$ for ADADELTA. For both LS methods, the increase of rotatable bonds results in a shift of E_{50} toward higher values, however that increase is clearly larger for Solis-Wets. For example, for a ligand with zero rotatable bonds, the estimated E_{50} values for Solis-Wets and ADADELTA would be 55×10^3 and 60×10^3 evaluations, respectively. However, when the number of rotatable bonds increases to 12, Solis-Wets is estimated to require 16×10^6 evaluations to reach a 50% success rate, while ADADELTA would require only 2.5×10^6 (a 6x reduction compared to Solis-Wets). Extrapolating for 20 rotatable bonds, this factor would increase to ~23x, as ADADELTA would need 30×10^6 evaluations, while Solis-Wets is predicted to require 680×10^6.

The trend shows that Solis-Wets can be competitive for relatively rigid molecules, but the gradient-based ADADELTA is clearly preferable for ligands with more rotatable bonds. Figure 10 shows the relative performance of the two methods (considering complexes for which E_{50} could be calculated for both). Ligands with three or fewer rotatable bonds (light blue circles) tend to be located below the equality line, showing lower E_{50} (i.e., higher algorithmic performance) for Solis-Wets. On the other hand, ligands with four or more rotatable bonds have lower E_{50} values using ADADELTA.

In addition to comparing Solis-Wets and ADADELTA in AutoDock-GPU, we also observed that the Solis-Wets implementation in AutoDock-GPU significantly outperforms that of AutoDock4.2.6 (Figure S4). The implementations differ in the genotype update. In AutoDock4.2.6, all genes are changed at every Solis-Wets iteration, while AutoDock-GPU implements a variable probability of changing each gene. That probability decreases with the number of genes, preventing the simultaneous modification of a large number of genes. Furthermore, in AutoDock-GPU, the magnitude of the changes to each gene change is smaller than in AutoDock4.2.6. Overall, the less abrupt modifications in the Solis-Wets of AutoDock-GPU confer significant advantage, with many systems showing 16x reductions in E_{50} (Figure S4).

Balancing algorithmic and evaluation rate performance.

On one hand, ADADELTA's search efficiency is higher than that of Solis-Wets, requiring fewer energy evaluations, e.g., 6× fewer for ligands with 12 rotatable bonds). On the other

hand, ADADELTA is computationally more expensive, requiring up to 16× more time for ligands containing ~100 atoms. Therefore, it is essential to identify the optimal balance at which the reduction of number of evaluations outweighs the increase in time spent per evaluation.

To address this question, we defined the time-to-E_{50}, which is the time required to perform E_{50} evaluations with each method. It is simply the product of E_{50} by the time per evaluation for each system. Time-to-E_{50} was calculated on the whole dataset using evaluation times obtained with the AMD Vega 56, which has intermediate performance in the range of GPU cards we tested. For ligands with 7 or fewer rotatable bonds, Solis-Wets performs E_{50} evals faster than ADADELTA (Figure 11). However, for ligands with 8 or more rotatable bonds, both methods take about the same time to do E_{50} evaluations.

CONCLUSIONS

This work describes AutoDock-GPU, an OpenCL-accelerated version of AutoDock4 running on GPUs, enhanced with the ADADELTA gradient-based method used for local search. The energies calculated by AutoDock-GPU agree within 0.01/0.02 kcal/mol to those of AutoDock4.2.6. We observed better algorithmic performance and a significant runtime speedup.

Algorithmic performance refers to the number of score evaluations required to achieve a pre-defined level of solution quality. The quality is measured by the score defined by the AutoDock4 scoring function, together with the RMSD from the native pose. Based on our experiments, the algorithmic improvement of ADADELTA over Solis-Wets increases with the number of rotatable bonds. For a ligand with three or four rotatable bonds, the algorithmic performance of both local-search methods is similar. For a ligand with 12 rotatable bonds, ADADELTA finds correct solutions with 6-fold fewer evaluations than Solis-Wets. For a ligand with 20 rotatable bonds, we estimate this reduction to be 23-fold.

On GPUs, the time spent in a single ADADELTA evaluation is 4x – 16x larger than in one Solis-Wets evaluation. This is because ADADELTA requires the calculation of gradients, which is computationally more expensive and more difficult to parallelize. Therefore, for ligands with a small number of rotatable bonds, the algorithmic advantage of ADADELTA might not outweigh its increased computational cost. Taking into account both the time spent per evaluation, as well as the number of evaluations required to produce correct solutions, we estimate that Solis-Wets is better for ligands with seven or fewer rotatable bonds, while either method performs equally well for ligands with eight or more rotatable bonds. We anticipate that the time spent per evaluation by both ADADELTA and Solis-Wets will improve over time reducing the gap between the two, therefore preference for either local-search method will then need to be reevaluated. Further development work is on-going to improve the parallelization of the gradient calculation on GPU platforms, with contributions from NVIDIA and Oak Ridge National Laboratory.

While previous attempts have been made to port the AutoDock4 engine to different accelerators, the level of speedup reported here is unprecedented. In order to benefit from

highly parallel architectures, such as that of GPU accelerators, it was necessary to perform major refactoring of the data workflow and processing, which where essentially re-designed from scratch in order to exploit multiple levels of paralellism. The resulting docking engine can achieve speedups ranging between ~2x and ~350x over single-threaded AutoDock4, depending on the hardware specifications of the accelerator platform, and the local-search method. Factors such as the number of ligand atoms also affects runtime speedups, with larger ligands yielding higher speedups for Solis-Wets, but lower speedups for ADADELTA. Interestingly, it was found that on accelerator platforms such as multicore CPUs, speedups were reduced by the ligand size (i.e. ligand atoms), but not by the local search method used. This is probably due to less effective parallelization of fine-grained tasks on CPUs, with their reduced number (compared to GPUs) of available processing elements (cores). We found that such tasks can be parallelized more effectively on GPUs, with their many (in the order of thousands) fine-grained processing elements.

The accelerator platforms tested in this study were commercial GPUs from different vendors (NVIDIA and AMD) that ranged from low- to high-end devices. Using GPUs with such different computing capabilities allows potential users to estimate the performance of AutoDock-GPU on systems similar to those available at their sites. The performance is expected to improve when new and more powerful hardware will be made available. Furthermore, we leveraged the portability of OpenCL by using AutoDock-GPU on general-purpose multicore CPUs and showing that it is still able to provide scalable performance gains. This could be beneficial in research settings where high-end GPUs are not available.

The improvements reported here are significant in terms of computational efficiency. Recently, we used an earlier version of AutoDock-GPU to predict the binding mode of BACE-1 inhibitors, while sampling macrocycle conformations *on-the-fly* during docking, as part of the fourth D3R Grand Challenge[59,60]. The improved search algorithms and GPU parallelization were important to tackle the complexity of BACE-1 inhibitors.

The AutoDock-GPU engine is being ported to the IBM's World Community Grid[11], where it will be powering the OpenPandemics - COVID-19 project[61] to search for inhibitors of the SARS-CoV2 viral proteins. The engine will then be available also to the other projects that use the current AutoDock4 engine for their virtual screenings. The engine has been used also on another COVID-19 project, in collaboration with NVIDIA and Oak Ridge National Laboratory[62] to screen an ultra-large library of one billion compounds against SARS-CoV2 targets in less than 24 hours. For consumer GPUs, such as the NVIDIA GTX 1080, we estimate that docking one molecule takes between 2 and 40 seconds per GPU card, translating into 2 to 40 thousand ligands per day per GPU.

Ultimately, AutoDock-GPU enables fast calculations with the AutoDock4 scoring function, which facilitates the docking of very large libraries of molecules, but also modeling of molecules of increasing size and complexity.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

ACKNOWLEDGMENTS.

This work was supported by the National Institutes of Health GM069832 (to SF), the AWS Cloud Credits for Research program, and by the ALEPRONA funding program #57186883 from the German Academic Exchange Service (DAAD) and the Peruvian National Program for Scholarships and Educational Loans (PRONABEC).

We thank JC Ducom and the HPC facility of The Scripps Research Institute, as well as AMD Inc. for technical support. We acknowledge the use of GNU Parallel[56], Matplotlib[63] and Seaborn[64].

This is manuscript 30040 of The Scripps Research Institute.

REFERENCES

1. Shoichet BK Virtual screening of chemical libraries. Nature. 2004, 432, 862–865. [PubMed: 15602552]
2. Irwin JJ; Shoichet BK ZINC - A Free Database of Commercially Available Compounds for Virtual Screening. J. Chem. Inf. Model 2005, 45, 177–182. [PubMed: 15667143]
3. Shoichet BK Screening in a spirit haunted world. Drug Discovery Today. 2006, 11, 607–615. [PubMed: 16793529]
4. Shoichet BK; Kobilka BK Structure-based drug screening for G-protein-coupled receptors. Trends Pharmacol. Sci 2012, 33, 268–272. [PubMed: 22503476]
5. Kincaid VA; London N; Wangkanont K; Wesener DA; Marcus SA; Héroux A; Nedyalkova L; Talaat AM; Forest KT; Shoichet BK; Kiessling LL Virtual Screening for UDP-Galactopyranose Mutase Ligands Identifies a New Class of Antimycobacterial Agents. ACS Chem. Biol 2015, 10, 2209–2218. [PubMed: 26214585]
6. Lyu J; Wang S; Balius TE; Singh I; Levit A; Moroz YS; O'Meara MJ; Che T; Algaa E; Tolmachova K; Tolmachev AA; Shoichet BK; Roth BL; Irwin JJ Ultra-large library docking for discovering new chemotypes. Nature. 2019, 566, 224–229. [PubMed: 30728502]
7. Lin J-H; Perryman AL; Schames JR; McCammon JA The relaxed complex method: Accommodating receptor flexibility for drug design with an improved scoring scheme. Biopolymers. 2003, 68, 47–62. [PubMed: 12579579]
8. Trott O; Olson AJ AUTODOCK Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. J. Comput. Chem 2010, 31, 455–461. [PubMed: 19499576]
9. Forli S; Huey R; Pique ME; Sanner MF; Goodsell DS; Olson AJ Computational protein-ligand docking and virtual drug screening with the AUTODOCK suite. Nat. Protoc 2016, 11, 905–919. [PubMed: 27077332]
10. Anderson DP BOINC: A System for Public-Resource Computing and Storage. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. Washington, DC, USA, 2004.
11. World Community Grid. https://www.worldcommunitygrid.org, accessed January 30, 2019.
12. Mermelstein DJ; Lin C; Nelson G; Kretsch R; McCammon JA; Walker RC Fast and flexible gpu accelerated binding free energy calculations within the amber molecular dynamics package. J. Comput. Chem 2018, 39, 1354–1358. [PubMed: 29532496]
13. Stone JE; Hardy DJ; Saam J; Vandivort KL; Schulten K GPU Computing Gems Emerald Edition; Applications of GPU Computing Series; Morgan Kaufmann, 2011; pp 5–18.
14. Stone JE; Hynninen A-P; Phillips JC; Schulten K Early Experiences Porting the NAMD and VMD Molecular Simulation and Analysis Software to GPU-Accelerated Open-POWER Platforms. High Performance Computing. Cham, 2016.
15. Sousa SF; Fernandes PA; Ramos MJ Protein-ligand docking: Current status and future challenges. Proteins: Struct., Funct., Bioinf 2006, 65, 15–26.
16. Sousa SF; Ribeiro AJ; Coimbra J; Neves R; Martins S; Moorthy N; Fernandes P; Ramos M Protein-ligand docking in the new millennium–a retrospective of 10 years in the field. Curr. Med. Chem 2013, 20, 2296–2314. [PubMed: 23531220]
17. Goodsell DS; Olson AJ Automated docking of substrates to proteins by simulated annealing. Proteins: Struct., Funct., Bioinf 1990, 8, 195–202.

18. Ravindranath PA; Forli S; Goodsell DS; Olson AJ; Sanner MF AutoDockFR: Advances in Protein-Ligand Docking with Explicitly Specified Binding Site Flexibility. PLoS Comput. Biol 2015, 11, 1–28.
19. Morris GM; Huey R; Lindstrom W; Sanner MF; Belew RK; Goodsell DS; Olson AJ AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. J. Comput. Chem 2009, 30, 2785–2791. [PubMed: 19399780]
20. Vieira TF; Sousa SF Comparing AutoDock and Vina in Ligand/Decoy Discrimination for Virtual Screening. Appl. Sci 2019, 9, 4538.
21. Nguyen NT; Nguyen TH; Pham TNH; Huy NT; Bay MV; Pham MQ; Nam PC; Vu VV; Ngo ST AutoDock Vina Adopts More Accurate Binding Poses but AutoDock4 Forms Better Binding Affinity. J. Chem. Inf. Model 2020, 60, 204–211. [PubMed: 31887035]
22. Bikadi Z; Hazai E Application of the PM6 semi-empirical method to modeling proteins enhances docking accuracy of AutoDock. J. Cheminf 2009, 1, 15.
23. Hetényi C; Paragi G; Maran U; Timár Z; Karelson M; Penke B Combination of a Modified Scoring Function with Two-Dimensional Descriptors for Calculation of Binding Affinities of Bulky, Flexible Ligands to Proteins. J. Am. Chem. Soc 2006, 128, 1233–1239. [PubMed: 16433540]
24. Adeniyi AA; Ajibade PA Comparing the Suitability of AutoDock, Gold and Glide for the Docking and Predicting the Possible Targets of Ru(II)-Based Complexes as Anticancer Agents. Molecules 2013, 18, 3760–3778. [PubMed: 23529035]
25. Tanchuk VY; Tanin VO; Vovk AI; Poda G A New, Improved Hybrid Scoring Function for Molecular Docking and Scoring Based on AutoDock and AutoDock Vina. Chem. Biol. Drug Des 2016, 87, 618–625. [PubMed: 26643167]
26. Buzko OV; Bishop AC; Shokat KM Modified AutoDock for accurate docking of protein kinase inhibitors. J. Comput.-Aided Mol. Des 2002, 16, 113–127. [PubMed: 12188021]
27. Santos-Martins D; Forli S; Ramos MJ; Olson AJ AutoDock4Zn: An Improved AutoDock Force Field for Small-Molecule Docking to Zinc Metalloproteins. J. Chem. Inf. Model 2014, 54, 2371–2379. [PubMed: 24931227]
28. Forli S; Olson AJ A Force Field with Discrete Displaceable Waters and Desolvation Entropy for Hydrated Ligand Docking. J. Med. Chem 2012, 55, 623–638. [PubMed: 22148468]
29. Forli S; Botta M Lennard-Jones Potential and Dummy Atom Settings to Overcome the AutoDock Limitation in Treating Flexible Ring Systems. J. Chem. Inf. Model 2007, 47, 1481–1492. [PubMed: 17585754]
30. Bianco G; Forli S; Goodsell DS; Olson AJ Covalent docking using AutoDock: Two-point attractor and flexible side chain methods. Protein Sci. 2016, 25, 295–301. [PubMed: 26103917]
31. Dhanik A; McMurray JS; Kavraki LE DINC: A new AutoDock-based protocol for docking large ligands. BMC Struct. Biol 2013, 13, S11. [PubMed: 24564952]
32. Serrano P; Aubol BE; Keshwani MM; Forli S; Ma C-T; Dutta SK; Geralt M; Wüthrich K; Adams JA Directional Phosphorylation and Nuclear Transport of the Splicing Factor SRSF1 Is Regulated by an RNA Recognition Motif. J. Mol. Biol 2016, 428, 2430–2445. [PubMed: 27091468]
33. Arcon JP; Modenutti CP; Avendaño D; Lopez ED; Defelipe LA; Ambrosio FA; Turjanski AG; Forli S; Marti MA AutoDock Bias: improving binding mode prediction and virtual screening using known protein-ligand interactions. Bioinformatics. 2019, 35, 3836–3838. [PubMed: 30825370]
34. Backus KM; Correia BE; Lum KM; Forli S; Horning BD; González-Páez GE; Chatterjee S; Lanning BR; Teijaro JR; Olson AJ; Wolan DW; Cravatt BF Proteome-wide covalent ligand discovery in native biological systems. Nature. 2016, 534, 570–574. [PubMed: 27309814]
35. Kannan S; Ganji R Porting AutoDock to CUDA. IEEE Congress on Evolutionary Computation. 2010; pp 1–8, ISSN: 1941–0026.
36. Olšák M; Filipovič J; Prokop M FastGrid – The Accelerated AutoGrid Potential Maps Generation for Molecular Docking. Computing and Informatics 2012, 29, 1325–1336–1336.
37. Mendonça E; Barreto M; Guimarães V; Santos N; Pita S; Boratto M Accelerating Docking Simulation Using Multicore and GPU Systems. Computational Science and Its Applications - ICCSA 2017. pp 439–451.

38. Mittal S; Vetter JS A Survey of CPU-GPU Heterogeneous Computing Techniques. ACM Comput. Surv 2015, 47, 69:1–69:35.
39. Belikov E; Deligiannis P; Totoo P; Aljabri M; Loidl H A Survey of High-Level Parallel Programming Models. 2013.
40. Diaz J; Muñoz-Caro C; Niño A A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era. IEEE Transactions on Parallel and Distributed Systems 2012, 23, 1369–1386.
41. Solis FJ; Wets RJ B. Minimization by Random Search Techniques. Math. Oper. Res 1981, 6, 19–30.
42. Zeiler MD ADADELTA: An Adaptive Learning Rate Method. arXiv.org. 2012, abs/1212.5701.
43. Huey R; Morris GM; Olson AJ; Goodsell DS A semiempirical free energy force field with charge-based desolvation. J. Comput. Chem 2007, 28, 1145–1152. [PubMed: 17274016]
44. Morris GM; Goodsell DS; Halliday RS; Huey R; Hart WE; Belew RK; Olson AJ Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. J. Comput. Chem 1998, 19, 1639–1662.
45. Solis-Vasquez L; Koch A A Performance and Energy Evaluation of OpenCL-accelerated Molecular Docking. Proceedings of the 5th International Workshop on OpenCL. New York, NY, USA, 2017.
46. OCLADock - OpenCL Accelerated Molecular Docking. https://git.esa.informatik.tu-darmstadt.de/docking/ocladock, accessed October 10, 2018.
47. The OpenCL Specification. https://www.khronos.org/opencl, accessed October 10, 2018.
48. Pechan I; Fehér B Molecular Docking on FPGA and GPU Platforms. Proceedings of the 21st International Conference on Field Programmable Logic and Applications. 2011.
49. Kingma DP; Ba J Adam: A Method for Stochastic Optimization. arXiv.org. 2014, abs/1412.6980.
50. Bitzek E; Koskinen P; Gähler F; Moseler M; Gumbsch P Structural relaxation made simple. Phys. Rev. Lett 2006, 97, 170201. [PubMed: 17155444]
51. Nocedal J; Wright SJ Numerical Optimization, 2nd ed.; Springer, 2006.
52. Hartshorn MJ; Verdonk ML; Chessari G; Brewerton SC; Mooij WT; Mortenson PN; Murray CW Diverse, high-quality test set for the validation of protein- ligand docking performance. J. Med. Chem 2007, 50, 726–741. [PubMed: 17300160]
53. Li Y; Han L; Liu Z; Wang R Comparative assessment of scoring functions on an updated benchmark: 2. Evaluation methods and general results. J. Chem. Inf. Model 2014, 54, 1717–1736. [PubMed: 24708446]
54. Berman HM; Westbrook J; Feng Z; Gilliland G; Bhat TN; Weissig H; Shindyalov IN; Bourne PE The Protein Data Bank. Nucleic Acids Res. 2000, 28, 235–242. [PubMed: 10592235]
55. O'Boyle NM; Banck M; James CA; Morley C; Vandermeersch T; Hutchison GR Open Babel: An open chemical toolbox. J. Cheminf 2011, 3, 33.
56. Tange O GNU Parallel - The Command-Line Power Tool. ;login: The USENIX Magazine 2011, 36, 42–47.
57. Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/on-demand/, accessed January 30, 2019.
58. Amazon Elastic Compute Cloud, User Guide for Linux Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-optimize-cpu.html, accessed January 30, 2019.
59. Santos-Martins D; Eberhardt J; Bianco G; Solis-Vasquez L; Ambrosio FA; Koch A; Forli S D3R Grand Challenge 4: prospective pose prediction of BACE1 ligands with AutoDock-GPU. J. Comput.-Aided Mol. Des 2019, 33, 1071–1081. [PubMed: 31691920]
60. El Khoury L; Santos-Martins D; Sasmal S; Eberhardt J; Bianco G; Ambrosio FA; Solis-Vasquez L; Koch A; Forli S; Mobley DL Comparison of affinity ranking using AutoDock-GPU and MM-GBSA scores for BACE-1 inhibitors in the D3R Grand Challenge 4. J. Comput.-Aided Mol. Des 2019, 33, 1011–1020. [PubMed: 31691919]
61. OpenPandemics - COVID-19 | Research | World Community Grid https://www.worldcommunitygrid.org/research/opn1/overview.do, accessed November 16, 2020.
62. LeGrand S; Scheinberg A; Tillack AF; Thavappiragasam M; Vermaas JV; Agarwal R; Larkin J; Poole D; Santos-Martins D; Solis-Vasquez L; Koch A; Forli S; Hernandez O; Smith JC; Sedova A GPU-Accelerated Drug Discovery with Docking on the Summit Supercomputer: Porting,

Optimization, and Application to COVID-19 Research. Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics. New York, NY, USA, 2020.

63. Hunter JD Matplotlib: A 2D graphics environment. Comput. Sci. Eng 2007, 9, 90–95.
64. Waskom M Seaborn. 2020; 10.5281/zenodo.592845.
65. Jääskeläinen P; de La Lama CS; Schnetter E; Raiskila K; Takala J; Berg H pocl: A Performance-Portable OpenCL Implementation. Int. J. Parallel Prog 2015, 43, 752–785.
66. Solis-Vasquez L; Koch A A Case Study in Using OpenCL on FPGAs: Creating an Open-Source Accelerator of the AutoDock Molecular Docking Software. Proceedings of the 5th International Workshop on FPGAs for Software Programmers. Berlin, Germany, 2018.
67. Zhong J; Hu X; Zhang J; Gu M Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms. Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce. 2005.
68. Razali NM; Geraghty J Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. World Congress on Engineering. 2011.
69. Pechan I; Fehér B; Bérces A FPGA-based acceleration of the AutoDock molecular docking software. Proceedings of the 6th Conference on Ph.D. Research in Microelectronics Electronics. 2010.
70. OpenCL 2.0 Reference Pages https://www.khronos.org/registry/OpenCL/sdk/2.0/docs/man/xhtml, accessed October 10, 2018.

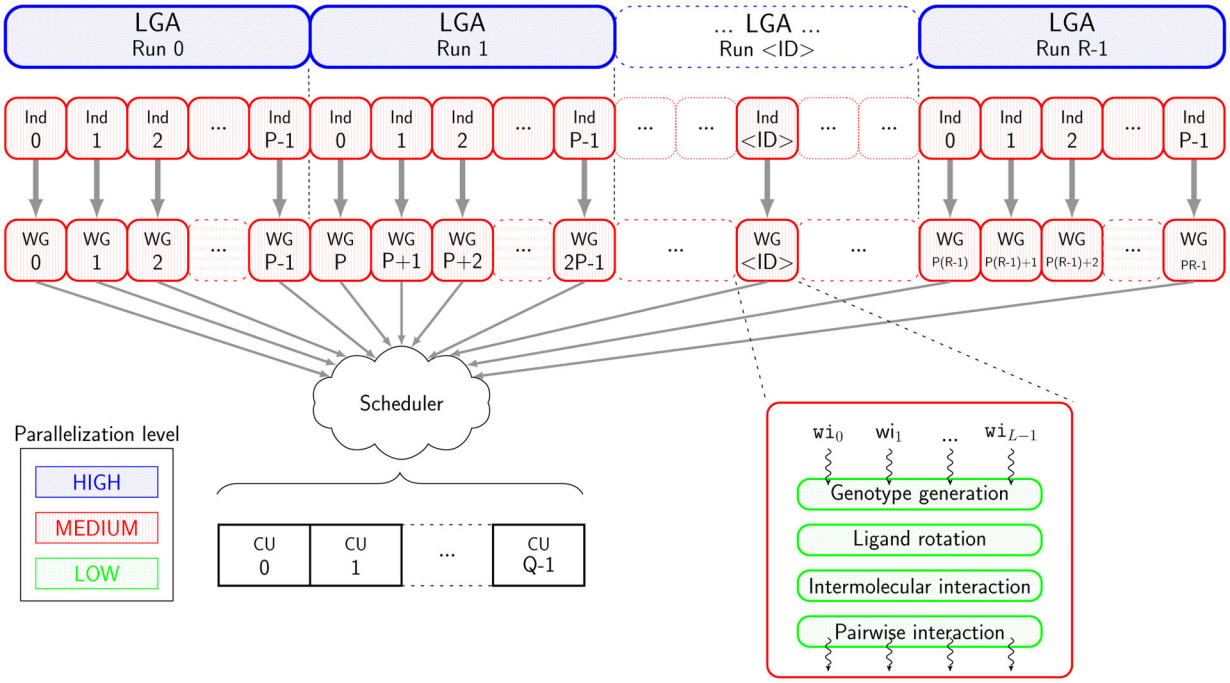


Fig. 1.

A population processed by a LGA run (Run_{ID}) can be decomposed into their individuals, and each individual (Ind_{ID}) can be mapped onto a work-group (WG_{ID}). The entire set of work-groups is distributed by the GPU runtime scheduler over the available Q compute units (CUs). A CU is a multi-threaded hardware unit capable of processing one work-group (composed of L work-items) at a time. The runs, individuals, and fine-grain tasks are colored according to their associated level of parallelism: high (blue), medium (red), and low (green).

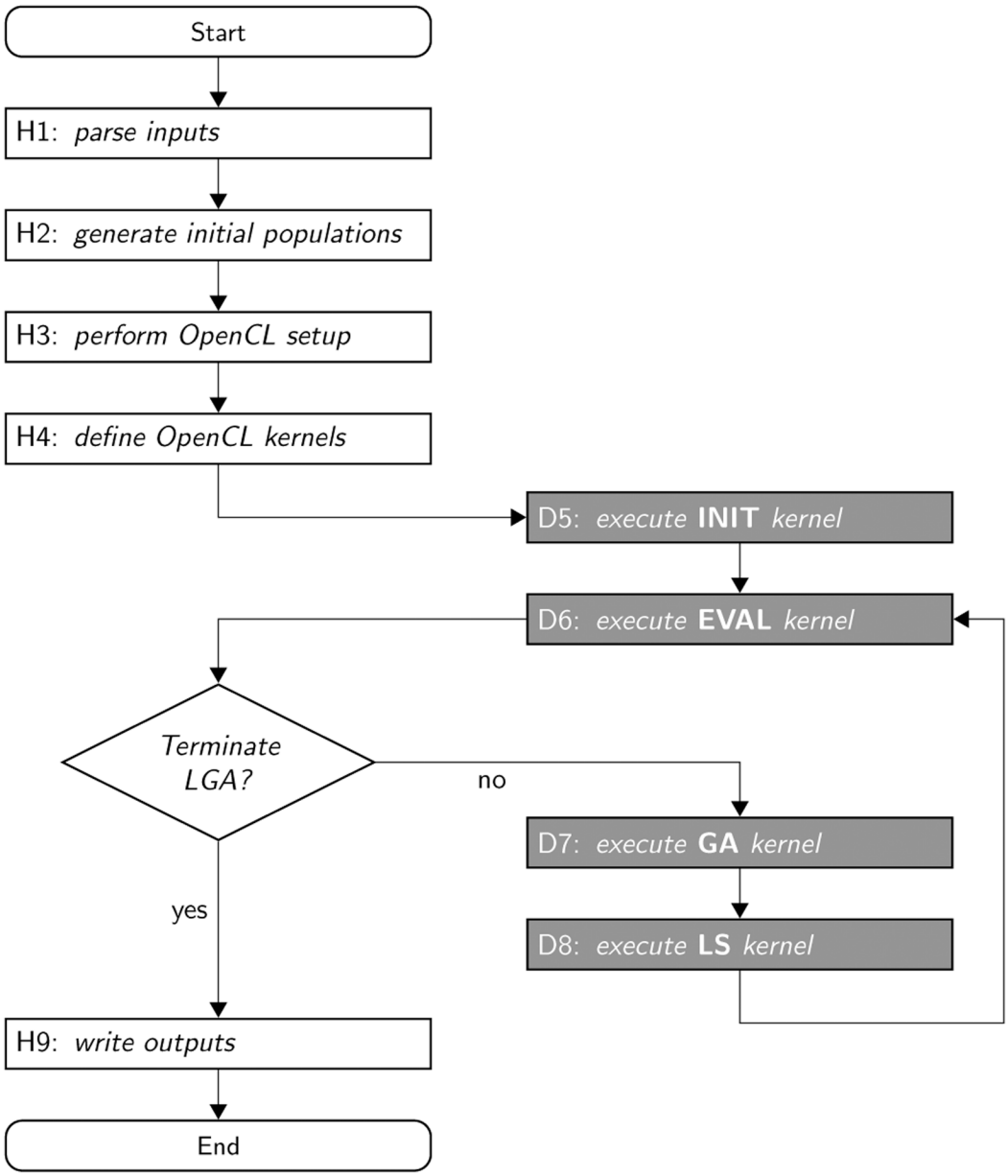


Fig. 2.

The overall AutoDock-GPU workflow consists of a sequence of host (H) and device (D) functions. Program execution always starts and finishes in host functions (depicted at the left side). Time-consuming functions, i.e., kernels, are executed iteratively on the device (depicted at the right side), while their termination is controlled by the host.

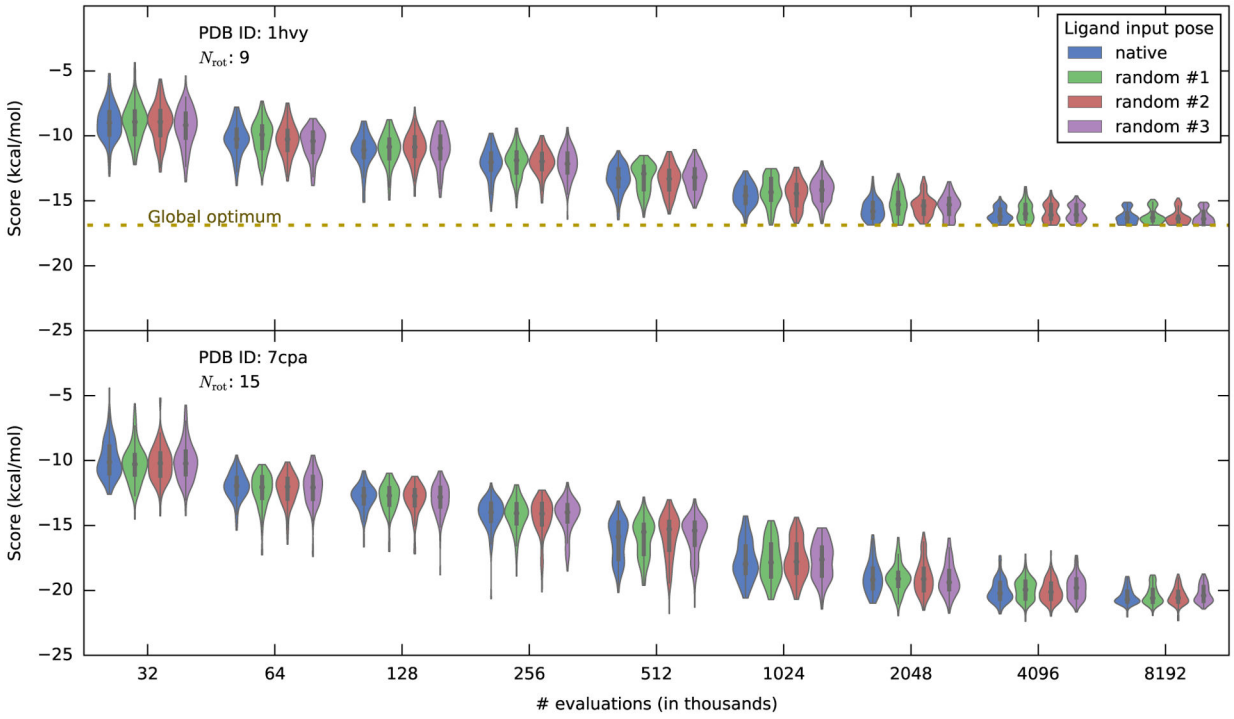


Fig. 3.

Distribution of scores returned by LGA runs with increasing number of evaluations (local-search: ADADELTA; local-search rate: 100%). Each violin plot represents scores from 100 LGA runs, and is colored by the input conformation of the ligand used. The global minimum was identified for the protein-ligand complex represented in the upper panel (PDB ID: 1hvy), but not for the complex in the bottom panel (PDB ID: 7cpa) because the distribution of scores did not converge towards a lower bound.

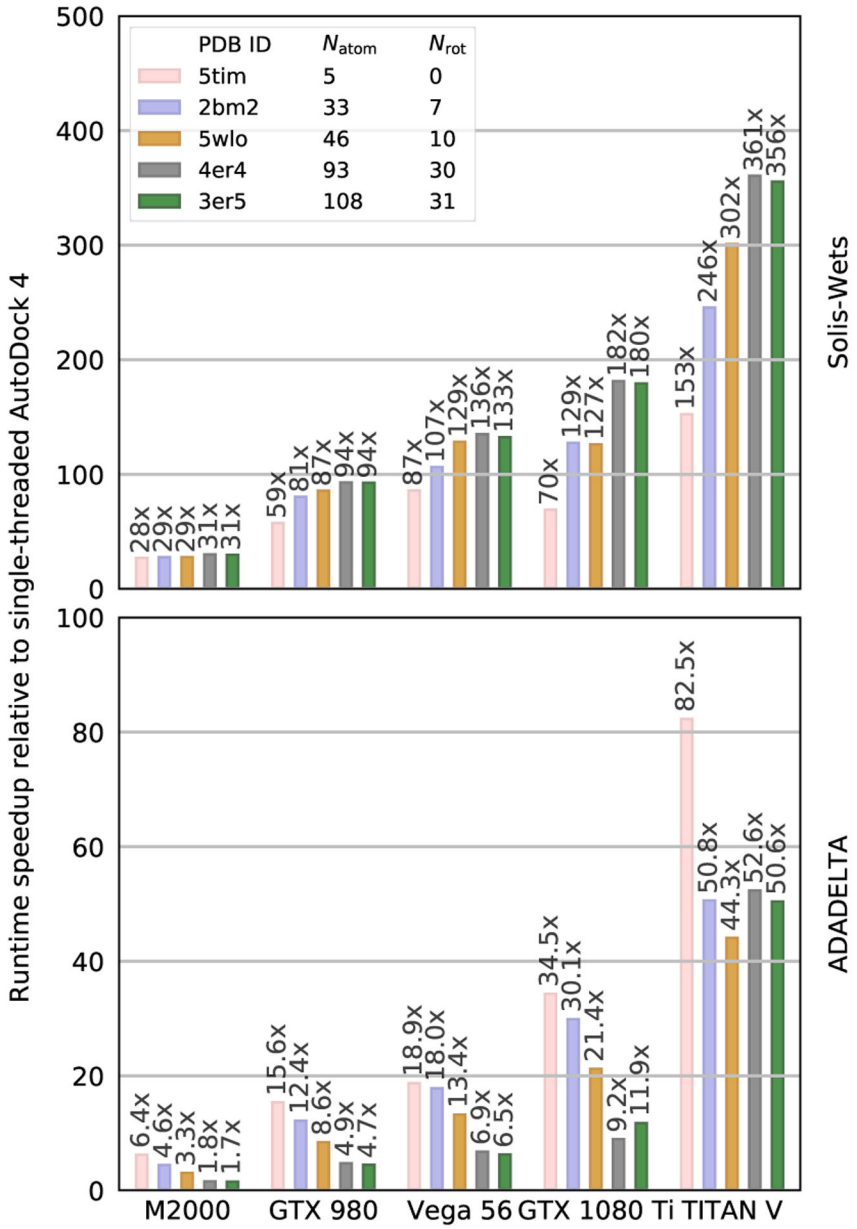


Fig. 4.

Runtime speedups of AutoDock-GPU (on various GPUs) with respect to AutoDock4 on a single CPU core. The number of LGA runs is 100, and the number of evaluations is 2,048,000.

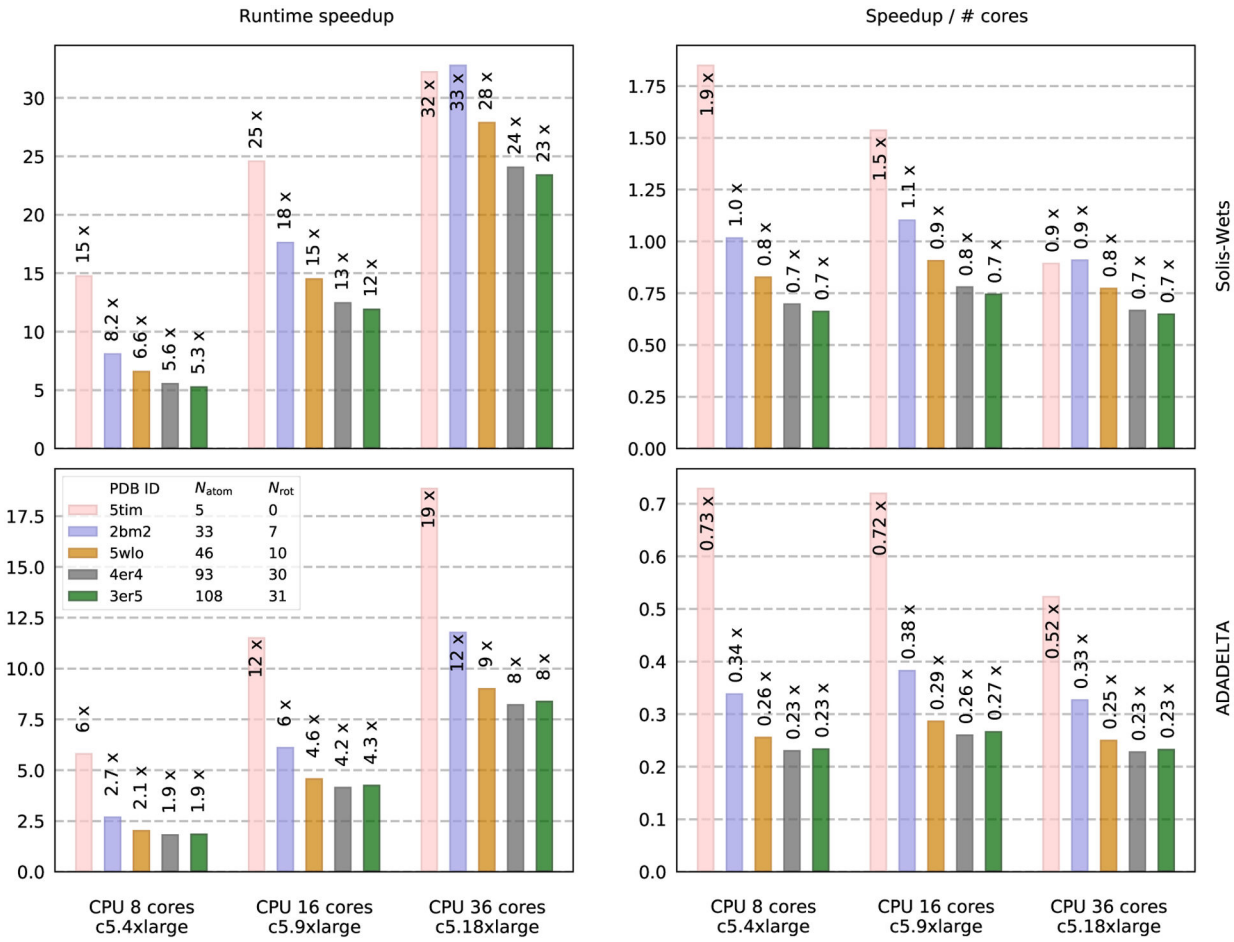


Fig. 5.

Performance scalability (left) and resource-utilization efficiency (right), both in terms of the number of CPU cores. The number of LGA runs is 100, and the number of evaluations is 2,048,000. For each ligand-receptor complex, same program commands were executed on all three CPU instances.

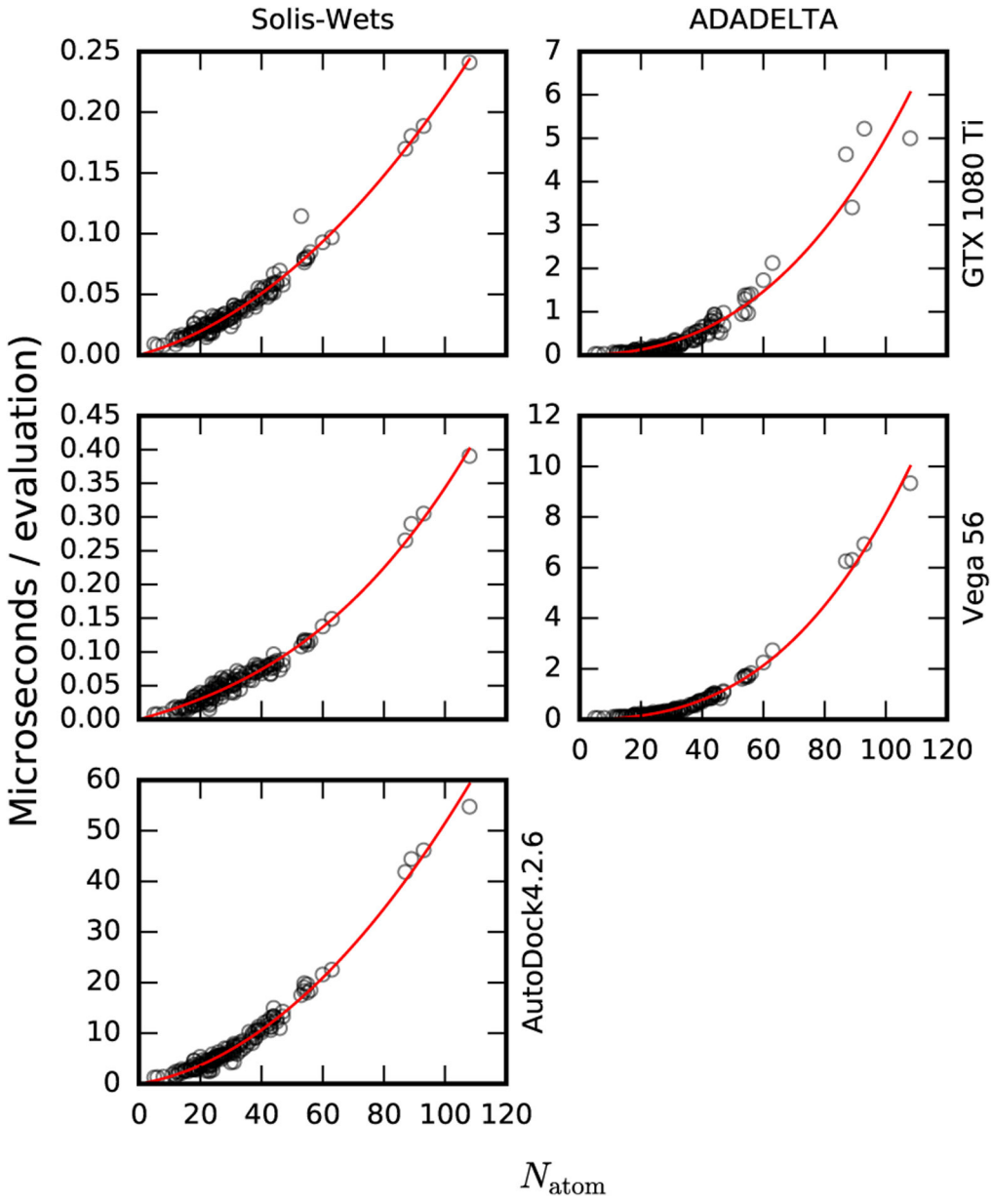


Fig. 6.

Time per evaluation in AutoDock-GPU and AutoDock4. Note that each subplot has a different scale for the y-axis. The red lines are third-degree polynomial fits. These fits were used to interpolate the time per evaluation reported in Table 2.

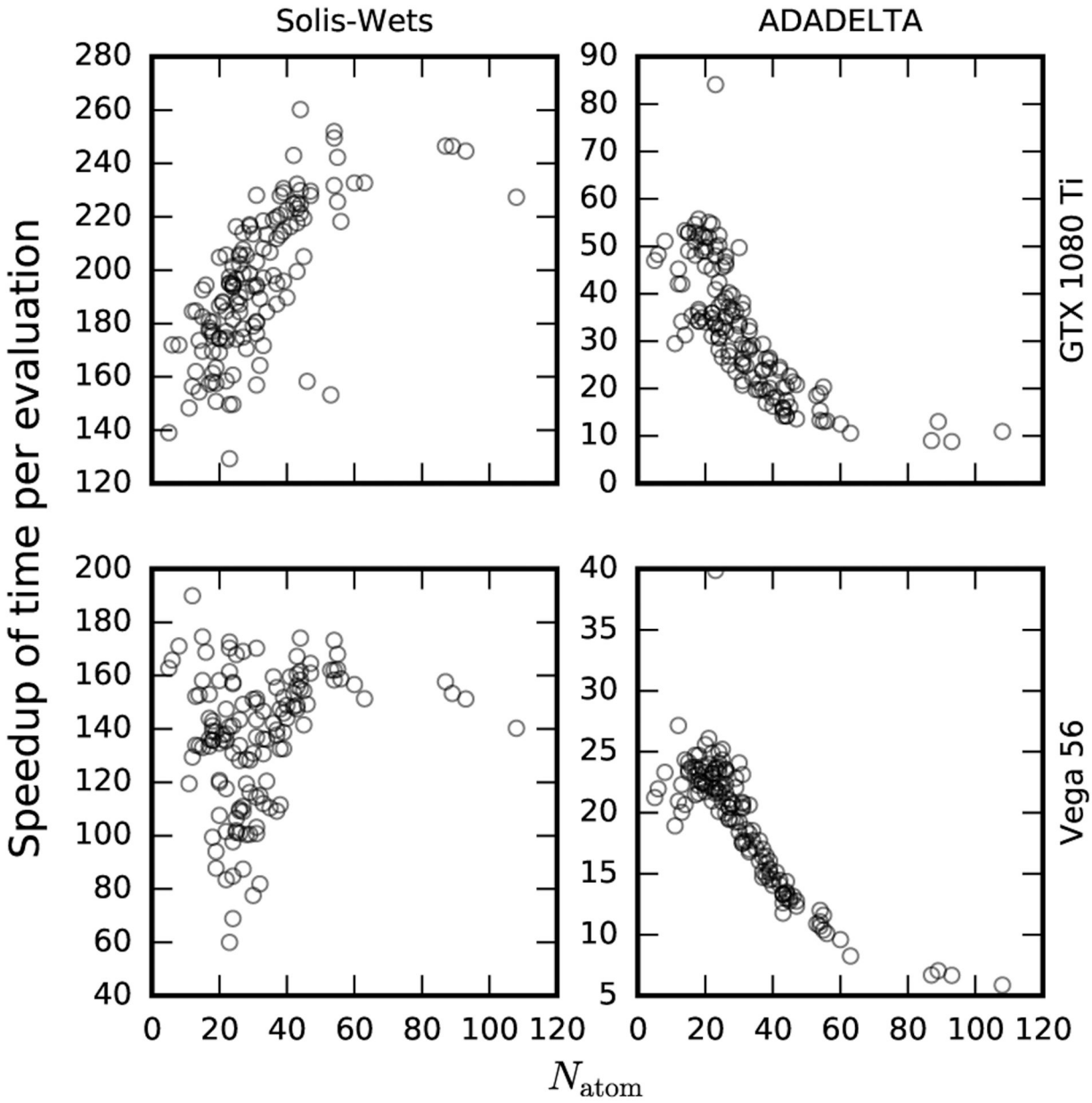


Fig. 7.

Time per evaluation speedup of AutoDock-GPU (on GTX 1080 Ti and Vega 56 GPUs) over AutoDock4 (on a single CPU core of an AWS c5.18x instance). Note that each subplot has a different scale for the y-axis.

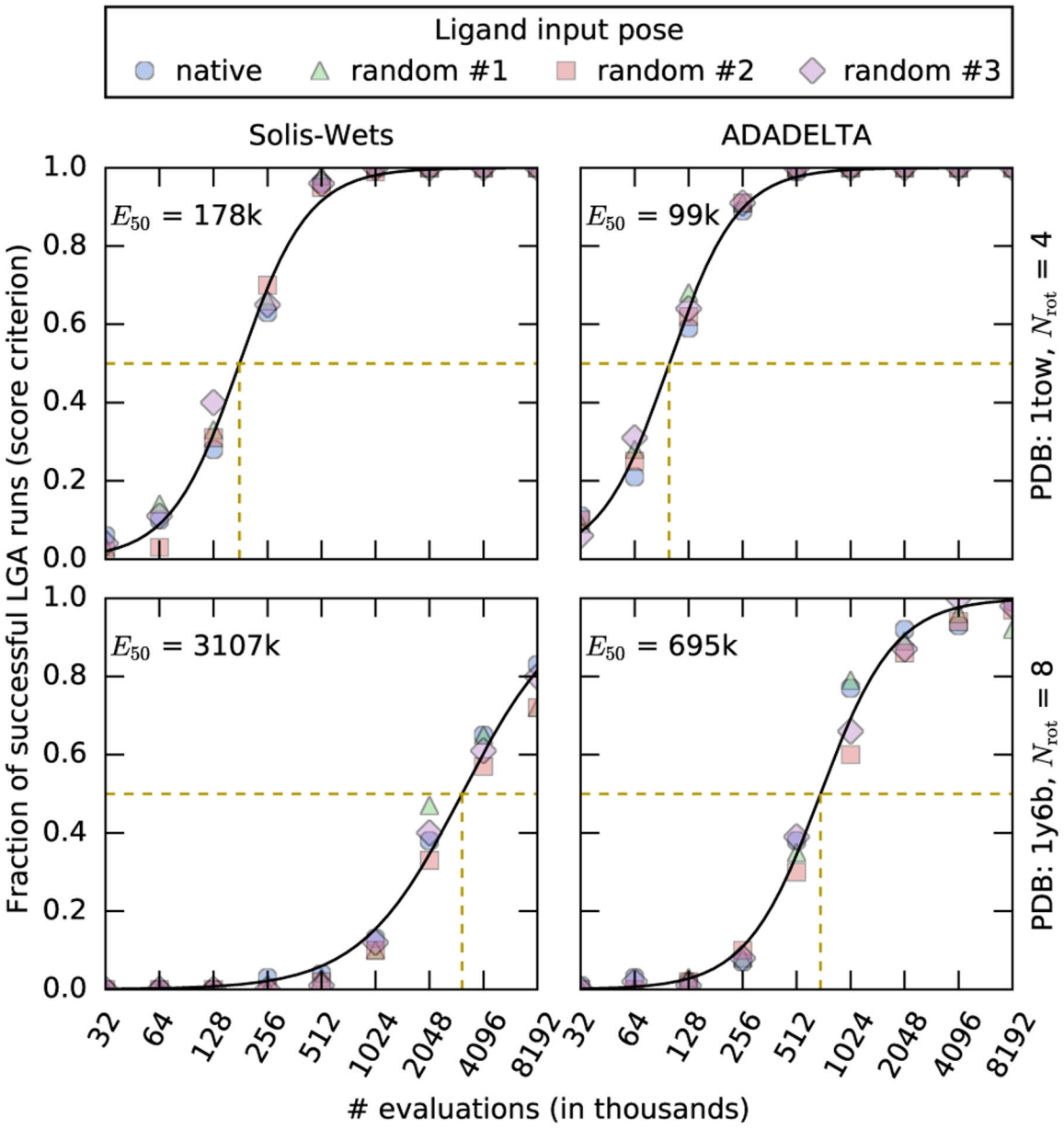


Fig. 8.

Fraction of successful LGA runs as a function of the number of score evaluations, using Solis-Wets and ADADELTA as local-search methods. The upper plots correspond to an easy search problem with only four rotatable bonds (PDB ID: 1tow), while the bottom plots correspond to a moderately difficult ligand with eight rotatable bonds (PDB ID: 1y6b). According to the score criterion, an LGA run is successful if it reports a pose within 1.0 kcal/mol from the global optimum. The black line is a fitted sigmoid curve (Equation 10) that estimates E_{50}, which is the number of evaluations at which 50% of LGA runs are successful. The dashed orange lines are a visual representation of E_{50} values.

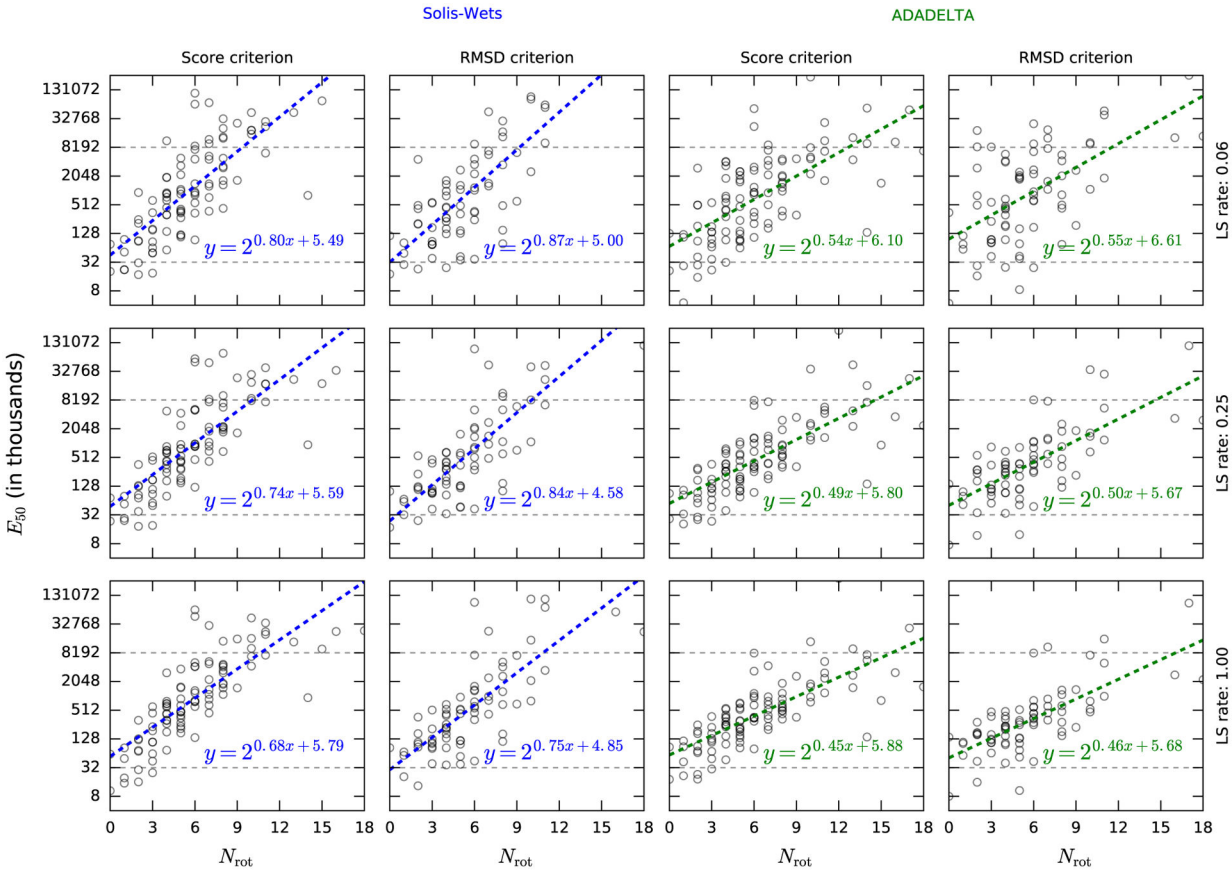


Fig. 9.

Dependency of E_{50} on the number of ligand rotatable bonds N_{rot} for Solis-Wets and ADADELTA local-search methods. The LS rate is 6% in the top row, 25% in the middle row, and 100% in the bottom row. The horizontal dashed lines correspond to the lower and upper limits of the number of evaluations used for calculating E_{50} values.

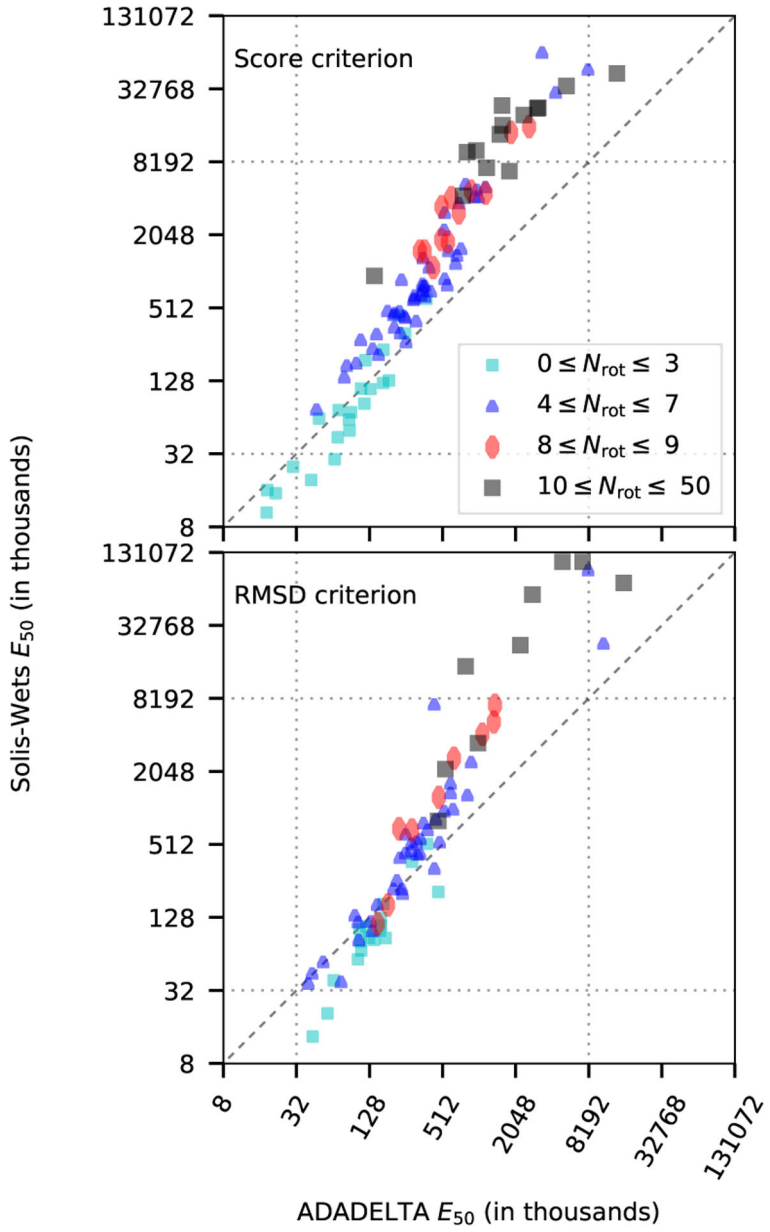


Fig. 10.

E_{50} values for ADADELTA and Solis-Wets using 100% LS rate. Each marker represents a protein-ligand complex and is color-coded by the number of rotatable bonds in the ligand. The horizontal and vertical dashed lines correspond to the lower and upper limits of the number of evaluations used for calculating E_{50} values.

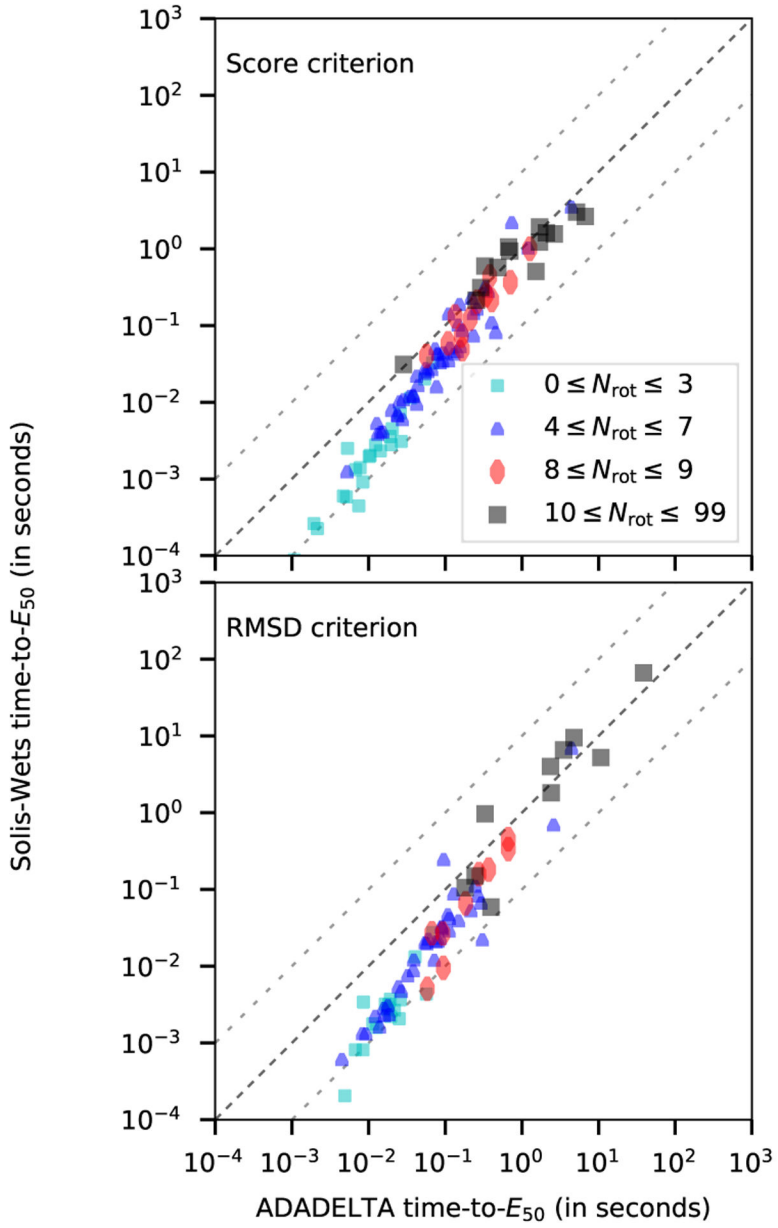


Fig. 11.

Time needed to perform E_{50} evaluations, using either ADADELTA (x-axis) or Solis-Wets (y-axis). The local search rate was 100%. For a given protein-ligand complex, time-to-E_{50} is the product of E_{50} by the corresponding time spent per evaluation. Here, evaluation rates were collected on the Vega 56 GPU platform.

Table 1.

Hardware characteristics of evaluated CPU-based platforms. Prices comprise the basic cost of compute services only[57] (i.e., underlying servers in the EU zone), and do not include charges for additional services (e.g., storage, database, migration, etc). In addition, as suggested for high-performance workloads[58], hyperthreading was disabled, allowing only a single thread per CPU core.

AWS CPU-based platform	Number of cores	Peak Memory Bandwidth (GB/s)	Peak Compute Performance FP32 (GFLOPS/s)	Total RAM Memory (GB)	On-demand EC2 price EU zone ($/hour)	Release Year
c5.4xlarge	8	42	768	30	0.776	2017
c5.9xlarge	16	85	1536	70	1.746	2017
c5.18xlarge	36	128	3456	140	3.492	2017

J Chem Theory Comput. Author manuscript; available in PMC 2022 February 09.

Table 2.

Time per evaluation (in microseconds) for different GPU platforms and varying number of ligand atoms. Data reported here is interpolated by polynomial fitting as shown in Figure 6.

	Time per evaluation (microseconds)					
	Solis-Wets			ADADELTA		
# Ligand atoms	20	40	80	20	40	80
NVIDIA M2000	0.13	0.36	1.14	0.52	3.09	20.54
NVIDIA GTX 980	0.04	0.11	0.33	0.23	1.16	6.37
AMD Vega 56	0.03	0.07	0.22	0.15	0.77	4.50
NVIDIA GTX 1080 Ti	0.02	0.05	0.15	0.13	0.58	2.90
NVIDIA TITAN V	0.01	0.02	0.07	0.06	0.19	0.67
AUTODOCK4	3.66	10.64	34.58	-	-	-

Table 3.

Comparison of selected hardware characteristics of the evaluated GPU-based accelerator cards.

GPU-based Accelerator Card	Peak Memory Bandwidth (GB/s)	Peak Compute Performance (GFLOPS/s)		On-board Memory (GB)	Release Year	Geometric Average Evaluation Rate Speedup	
		FP32	FP64			Solis-Wets	ADADELTA
NVIDIA M2000	105.8	1786	55	4	2016	28.25	4.87
NVIDIA GTX 980	224.4	4981	155	4	2014	88.13	13.22
AMD Vega 56	409.6	10566	660	8	2017	134.05	18.25
NVIDIA GTX 1080 Ti	484.4	11340	354	11	2017	193.71	29.45
NVIDIA TITAN V	652.8	14900	7450	12	2017	398.60	63.94