The International Journal of High Performance Computing Applications
():1–13
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

METADOCK: A Parallel Metaheuristic schema for Virtual Screening methods

Baldomero Imbernón[1], José M. Cecilia[1], Horacio Pérez-Sánchez[1] and Domingo Giménez[2]

Abstract

Virtual Screening through Molecular Docking can be translated into an optimization problem, which can be tackled with metaheuristic methods. The interaction between two chemical compounds (typically a protein, *enzyme* or *receptor*, and a small molecule, or *ligand*) is calculated by using highly computationally demanding scoring functions that are computed at several binding spots located throughout the protein surface. This paper introduces $METADOCK$, a novel Molecular Docking methodology based on parameterized and parallel metaheuristics and designed to leverage heterogeneous computers based on CPU-GPU architectures. The application decides the optimization technique at running time by setting a configuration schema. Our proposed solution finds a good workload balance via dynamic assignment of jobs to heterogeneous resources which perform independent metaheuristic executions when computing different molecular interactions required by the scoring functions in use. A cooperative scheduling of jobs optimizes the quality of the solution and the overall performance of the simulation, so opening a new path for further developments of Virtual Screening methods on high-performance contemporary heterogeneous platforms.

Keywords

Drug Discovery, Virtual Screening, Molecular Docking, HPC, Metaheuristics, Heterogeneous Computing

Introduction

The discovery of new drugs may benefit from using Virtual Screening (VS) methods Jorgensen (2004). These are computational techniques that analyze large libraries of small molecules (*ligands*) to search for those compounds which are most likely to bind to a drug target, typically a protein receptor or enzyme (e.g. Rester (2008); Rollinger et al. (2008)). These libraries of chemical compounds may have up to millions of ligands Irwin and Shoichet (2005). Indeed, the analysis of larger databases increases exponentially the chances of generating hits.

The computational complexity of VS methods is determined by two main parameters: the size of the database to be analyzed and the accuracy of the chosen VS method. Fast VS methods with atomic resolution require some minutes per ligand Zhou et al. (2007). In contrast, molecular dynamic approaches can require up to thousands of hours per ligand Wang et al. (2006). Therefore, the main bottleneck for the success of VS methods is the lack of computational resources or, in other words, there is a need for highly efficient applications that leverage emergent, high performance computing architectures Asanovic et al. (2006).

We are witnessing the steady transition to heterogeneous computing systems Top500 (2016) where nodes combine traditional Central Processing Units (CPUs), which may have multiple cores, and many-core systems or accelerators (like NVIDIA GPUs NVIDIA Corporation (2014) or Intel Xeon Phi Sodani et al. (2016)), that enable us to speed-up computationally demanding parts of the code. Heterogeneity limits system growth, which can not now be addressed in an incremental way. In particular, concepts like scalability, energy barrier, data management, programmability and reliability are becoming challenges for tomorrow's cyberinfrastructures Song et al. (2016). Runtime systems are still too immature to map processors and computations efficiently. In the meantime, researchers are focusing on the latest breakthroughs in high performance computing together with specific fields (metaheuristics, image processing, computational modeling, etc.). This results in a vertical approach enabling remarkable advances in computer-driven scientific simulations, the so-called hardware-software co-design De Michell and Gupta (1997).

Programmers play a fundamental role in this emerging scenario. They have to redesign, and even rethink, applications to leverage the best features of all the sides in a joint execution to maximize performance, with parallelism as a mandatory ingredient. Of particular interest to us are *metaheuristic* algorithms, especially those inspired by *natural* processes, whose computations are intrinsically massively parallel, and therefore well-suited for implementation in this area of computation Cecilia et al. (2013). There are a wide variety of metaheuristic algorithms, each with its own characteristics Blum and Roli (2003), and sometimes it is necessary to develop and experiment with

[1]Bioinformatics and High Performance Computing Research Group (BIO-HPC), Polytechnic School, Universidad Católica San Antonio of Murcia (UCAM), 30107, Murcia, Spain.
[2]Department of Computing and Systems, University of Murcia, Spain.

Corresponding author:
Baldomero Imbernón, Bioinformatics and High Performance Computing Research Group (BIO-HPC), Polytechnic School, Universidad Católica San Antonio of Murcia (UCAM), 30107, Murcia, Spain.
Email: bimbernon@ucam.edu

various soft computing methods, and tune them for each particular problem in order to obtain satisfactory results.

Metaheuristics are frequently used to solve NP-hard problems Rozenberg et al. (2011). Some of these problems are in the field of *Bioinformatics*, e.g., DNA analysis Minetti et al. (2008) or molecular docking López-Camacho et al. (2015). Many metaheuristic methods are available, including *Distributed metaheuristics* (e.g., Genetic Algorithms, Scatter Search, Ant Colony, Particle Swarm Optimization, etc.) and *Neighborhood metaheuristics* (e.g., Tabu Search, Hill Climbing, Simulated Annealing, etc.). The best metaheuristic to deal with a particular problem is not clear. Many experiments with different metaheuristics and hybridations of basic metaheuristics are needed to discover the optimum solution for a problem. Additionally, for any particular metaheuristic, a tuning process is traditionally conducted to select appropriate values of some parameters in the metaheuristic, and experimentation with several metaheuristics and their tuning process will drastically increase the computational cost.

In this paper, we introduce $METADOCK$, a virtual screening technique that uses a unified schema to generate a wide range of metaheuristics. $METADOCK$ is designed to leverage massively parallel and heterogeneous architectures, including Chip Multiprocessors and NVIDIA's GPUs. We use a molecular *docking* computational methodology that seeks to predict non-covalent binding of molecules or, more frequently, of a macromolecule (receptor) and a small molecule (ligand). The goal is to predict the bound conformations and the binding affinity; i.e., the strength of association, which is usually measured with a scoring function. These functions compute the score by calculating different conformations of the ligand at several binding spots throughout the protein surface, including computations between pairs of atoms in the protein and the ligand. For detailed reviews on recent and widely used scoring functions see Yuriev and Ramsland (2013); Li et al. (2014b,a); Lionta et al. (2014).

This paper is an extension of a previous work developed by the same authors Imbernón et al. (2016). Major contributions include:

1. A new methodology called $METADOCK$ for virtual screening, widely applied in the field of computational Drug Discovery, is introduced. It is based on a parameterized schema that is able to generate a branch of different search algorithms (Metaheuristics) by setting different input parameters.
2. $METADOCK$ leverages heterogeneous computers based on CPUs and NVIDIA GPUs to provide an agile framework to run real experiments. We assume the nodes may have NVIDIA GPUs with different compute capabilities. A load balance strategy is introduced to efficiently distribute different workloads at runtime among all GPUs in the system. This load balancing strategy is based on the application performance.
3. We analyze $METADOCK$'s prediction quality by calculating its performance on binary classification (active or not active compounds), for which we generate Receiver Operating Characteristic (ROC) curve analysis, after processing benchmarks containing experimental information about protein-ligand datasets, such as Directory of Useful Decoys (DUD). Our results place $METADOCK$ as a very competitive docking method, reporting an Area Under Curve (AUC) value of up to 0.84, in the benchmarks reported.
4. We provide an extensive performance analysis to validate our parallelization strategy for such heterogeneous environments. Our results reveal that our techniques achieve up to 50x speed-up factor compared to a multicore counterpart version. Moreover, they also provide the following conclusions: (1) homogeneous distribution is not a good load balancing strategy for systems with GPUs with different compute capabilities; (2) technical specifications are not enough to achieve peak performance, and load balancing at runtime is needed, with the workload depending on application performance; (3) all these parallelization strategies give the opportunity to improve the solution quality in our problem.

The rest of the paper is structured as follows. The next section includes the background of Virtual Screening and some relevant knowledge about metaheuristics and GPU computing. Next, our metaheuristic-based virtual screening technique and its design for heterogeneous computers are introduced before showing the experimental set-up and results. A final section summarizes the conclusions and gives some directions for future work.

Background

This Section briefly shows the main characteristics of Virtual Screening methods, metaheuristics and GPU computing for a better understanding of the rest of the paper.

GPU computing

Almost from the outset, computer architects have relied on technology scaling to provide sustainable performance. Heterogeneous architecture design is now seen as the only solution to continue scaling Moore's law through innovation Austin (2015), with systems where nodes combine traditional multicore architectures (CPUs) and accelerators (mostly GPUs Kirk and Wen-mei (2013) or Intel Xeon Phi Sodani et al. (2016)). In a few years, the GPGPU (developing General Purpose application on GPUs) field expanded and became one of the best ways of achieving high performance computing from emergent heterogeneous computers. We briefly introduce the CUDA programming model and refer the reader to NVIDIA Corporation (2014) for insights. Compute Unified Device Architecture (CUDA) is a platform for Graphics Processing Units (GPUs) that covers both hardware and software. On the hardware side, the GPU consists of N multiprocessors which are replicated within the silicon area. Each is endowed with M cores sharing the control unit, and with a shared memory (a small cache explicitly managed by the programmer). Moreover, all of these multiprocessors are connected to an off-chip memory (GPU device memory) that acts as an interface to the host CPU. Each GPU generation has increased the CUDA Compute Capabilities (CCC), as well as the number of cores, and the shared and device memory sizes (see Table 1). In conjunction with these developments, power efficiency has been reduced by a factor of 2 in each new generation.

Table 1. CUDA summary by generation, with Maxwell to increase the number of cores soon.

Hardware generation and starting year	Fermi 2010	Kepler 2012
Multiprocessors per die (up to)	16	15
Cores per multiprocessor	32	192
Total number of cores (up to)	512	2880
Shared memory size (maximum in kilobytes)	48	48
Device memory size (maximum in gigabytes)	6	12
CUDA Compute Capabilities	2.x	3.x
Peak single-precision performance (GFLOPS)	1178	4290
Performance per watt (approx. and normalized)	2	6

The CUDA software paradigm is based on a hierarchy of abstraction layers: the *thread* is the basic execution unit; threads are grouped into *blocks*; and blocks are mapped to multiprocessors. C language procedures to be ported to GPUs are transformed into CUDA *kernels*, mapped to many-cores in an SIMD (Single Instruction Multiple Data) fashion (that is, with all threads running the same code but with different IDs). The programmer deploys parallelism by declaring a *grid* composed of blocks equally distributed among all the multiprocessors. A kernel is therefore executed by a grid of thread blocks, where threads run simultaneously, grouped in batches called *warps*, which are the main scheduling units.

Metaheuristics

There are many optimization problems of high computational cost which can not be solved by evaluating all the possible solutions. Due to the high computational cost of exact methods, the optimum solutions for those NP-hard problems can be found for only very small instances, and so they are traditionally approached through heuristics and metaheuristics (general information on metaheuristics can be found, for example, in Blum et al. (2011); Dréo et al. (2005); Glover and Kochenberger (2003); Hromkovič (2003); Michalewicz and Fogel (2002)), which are tuned for the problem in question.

Metaheuristics include an abstraction layer that may provide a sufficiently good solution for an optimization problem, especially with limited computation capacity or inexact information Bianchi et al. (2009). They reduce the search space, focusing only on the most promising areas, and thus, cannot guarantee the analysis of all possible solutions, which means that they do not guarantee to find the optimal solution. There are many metaheuristic algorithms of different characteristics Blum and Roli (2003) that can provide several good solutions to the same problem. Among them, we highlight:

- *Distributed metaheuristics*, which search for solutions within the whole solution space. These work with populations or sets of elements that are combined in order to generate better solutions progressively. Some examples include *Scatter Search*, *Genetic Algorithms*, *Ant Colony* and *Particle Swarm Optimization*.
- *Neighborhood metaheuristics*, which work with an element in the solution space and search for better elements in its neighborhood. Examples include *Hill Climbing, Tabu Search, Guided Local Search, Variable Neighborhood Search, Simulated Annealing* and *GRASP*.

Virtual Screening

We draw on our description of the Virtual Screening (VS), which was first given in Guerrero et al. (2014); Sánchez-Linares et al. (2012). VS methods are computational techniques used in several scientific areas, such as catalysts and energy materials Franco (2013), and mainly drug discovery Kitchen et al. (2004), where experimental techniques can benefit from the predictions provided by simulation methods.

VS methods search for libraries of small molecules that can potentially bind to a drug target, typically a protein receptor or enzyme, with high affinity. Within VS methods, Molecular Docking techniques simulate the docking process of small molecules into the structures of macromolecular targets (see Figure 1). Moreover, they look for (i.e., score) the optimal binding sites by providing a ranking of chemical compounds according to the estimated affinity or *scoring* Schneider (2002). In general, VS methods optimize *scoring functions*, which are mathematical models used to predict the strength of the non-covalent interaction between two molecules after docking Jain (2006). In fact, these candidate molecules will continue the drug discovery process roadmap that goes from in-vitro studies, to animal investigations and, eventually, to human trials Drews (2000).

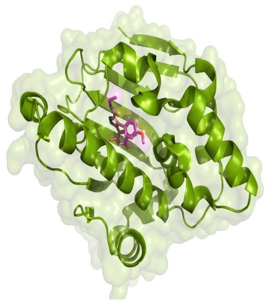

Figure 1. Crystallographic structure of 5-(5-chloro-2,4-dihydroxyphenyl)-N-ehtyl-4-(4-methoxyphenyl)-1H-pyrazole-3-carboxamide (pink color) bound to HSP90 protein (green color). Details available at PDB database with accession code 2BSM.

Although VS methods have been researched for many years and several compounds can be identified that evolve into drugs, the impact of VS has not yet fulfilled all expectations. Neither the VS methods nor the scoring functions used are sufficiently accurate to identify high-affinity ligands reliably. To deal with a large number of potential candidates (many databases comprise hundreds of thousands of ligands), VS methods must be very fast and still be able to identify "the needles in the haystacks". These techniques require hundreds of CPU hours for each ligand, and, according to Wang et al. (2006), even thousands of CPU hours for each ligand when simulation strategies are used to compute absolute binding affinities.

The relevant non-bonded potentials used in VS calculations are the Coulomb, or electrostatic, and the Lennard-Jones potentials, since they describe the most important short and long-range interactions between atoms of the protein-ligand system very accurately. The calculation of non-bonded potentials is usually the most time-consuming calculation in VS methods. For example, in Molecular Dynamics simulations, the calculation of these kernels can take up to 80% of the total execution time Kuntz et al. (2001).

Within these VS methods, of particular interest to us are protein-ligand docking techniques. Examples are given in Yuriev et al. (2011) and Huang and Zou (2010). Docking simulations are typically carried out on the protein surface using known methods, like Autodock Morris et al. (1998); Glide Friesner et al. (2004); DOCK Ewing et al. (2001); FMD Dolezal et al. (2015), which combines message passing interface (MPI) with multithreading; or BUDE McIntosh-Smith et al. (2014), a molecular docking program for hybrid computing architectures that exploits the heterogeneity with OpenCL for portability to different computer architectures. The surface region is commonly derived from the position of a particular ligand in the protein-ligand complex, or from the crystal structure of the protein without any ligands. The main problem of many docking methods is that they assume, once the binding site is specified, that all ligands will interact with the protein in the same region, and completely discard the other areas of the protein. In $BINDSURF$, Sánchez-Linares et al. (2012) uses GPUs to overcome this problem by dividing the whole protein surface into arbitrary independent regions (or spots). Using GPU parallelism, a large ligand database is screened against the target protein simultaneously over its whole surface, and docking simulations for each ligand are performed simultaneously in all the specified protein spots, resulting in new spots found after the examination of the distribution of scoring function values over the entire protein surface.

Metaheuristics for virtual screening on heterogeneous systems

Traditional parallel implementations are not always efficient when ported to heterogeneous systems. They are often inherited from scalable supercomputers, where all nodes in the cluster have the same compute capabilities, and therefore they lack the ability to distinguish computational devices with asymmetric computational power. Differences are not limited to fundamental hardware design (CPUs vs. GPUs), but also occur within the same family of processors. For example, the Kepler family (see Table 1) includes Tesla K20, K20X and K40 models, endowed with 13, 14 and 15 multiprocessors, respectively (the K80 model even reaches 30 multiprocessors split into two chips). Here, we distinguish two different aspects; the system itself, which may be heterogeneous or homogeneous, and the parallel distribution of the workload which can also be heterogeneous or homogeneous. This section shows our proposal, $METADOCK$, for metaheuristic-based virtual screening applications that leverage massively parallel and heterogeneous computers. We introduce the reader to the design of our virtual screening approach before showing the parallelization strategy for heterogeneous distribution of the workload.

METADOCK: Metaheuristics for VS methods

Our Virtual Screening technique divides the whole protein surface into arbitrary and independent regions (or spots). Spots are specified around alpha-carbons of the protein backbone, so that we can ensure a full scanning of the protein surface. All these spots are independent of each other and, thus, offer great opportunities for data-based parallelization. Docking simulations for each ligand are then performed simultaneously at every protein spot. Actually, the computation places copies of the same ligand at each of those spots. These copies (*a.k.a.* individuals or conformations) are different from each other as they have a different position and orientation with respect to each spot. Docking simulations search for an optimized *conformation* for both the protein and ligand and the relative orientation between them, such that the free energy (given by the *scoring function*) of the overall system is minimized. Therefore, $METADOCK$ uses an optimization procedure where the scoring function, that models the non-bonded interactions between protein and ligand, is minimized throughout the execution. With that in mind, we first introduce the optimization procedure used in $METADOCK$ before briefly describing the GPU implementation of the underlying scoring function. The scoring function computation represents more than 95% of the $METADOCK$ overall computation time and thus it is offloaded to the GPU to increase overall application performance.

Search method based on a parameterized metaheuristic schema Algorithm 1 shows the $METADOCK$ generic template that we use to generate several metaheuristics for the VS problem. Several authors agree (Raidl (2006); Vaessens et al. (1998)) that many metaheuristics, particularly those based on populations, share six basic functions (see Algorithm 1): *Initialize*, *End condition*, *Select*, *Combine*, *Improve* and *Include*. These functions are like algorithmic templates in which the programmer can provide different implementations, so obtaining different metaheuristics. Population-based metaheuristics maintain and improve multiple candidate solutions (S), and often use population characteristics to guide the search. Local search metaheuristics are also included in the schema, with $|S| = 1$.

Algorithm 1 $METADOCK's$ parameterized metaheuristic schema

```
Initialize(S,ParamIni)
while no End condition(S) do
    Select(S,Ssel,ParamSel)
    Combine(Ssel,Scom,ParamCom)
    Improve(Scom,ParamImp)
    Include(Scom,S,ParamInc)
end while
```

Each of the functions in the algorithm works with various sets or populations (*S*, *Ssel* and *Scom*). *S* represents the whole population of candidate solutions. In our case, a candidate solution (or individual) is a conformation. Thus, several individuals are selected (*Ssel*) to be combined, so generating a new set of elements, *Scom*. Candidate solutions can also be improved by applying a local search; i.e. moving, translating and/or rotating with respect to each spot.

A further step in developing unified metaheuristics schemes is the introduction of several parameters, i.e., *metaheuristic parameters* (ParamXXX in Algorithm 1), in each of these functions to provide a wider range of metaheuristics. Almeida et al. (2013); Cutillas-Lozano et al. (2012); Cutillas-Lozano and Giménez (2013) show that the use of a parameterized schema of metaheuristics helps to find satisfactory metaheuristics and to tune them for a particular problem. Several metaheuristics could be evaluated for the problem (each with its corresponding tuning process); and hybrid metaheuristic schemes can also be considered. As a consequence, the selection and tuning for a satisfactory metaheuristic or hybridation for a problem is a complex process, which can require large execution times.

$METADOCK$ is based on that unified parameterized metaheuristic schema and is used for docking simulations. As mentioned in the Metaheuristics Section and as shown in Algorithm 1, the schema is like a template that defines a set of functions to be implemented for a particular problem. Those functions use several parameters to provide different metaheuristic implementations. Particularly, $METADOCK$ uses up to fifteen metaheuristic parameters (see Table 2).

The schema is applied at each spot, with the same metaheuristic parameters in Table 2 (the same metaheuristic) for each spot, and with the basic functions working on different subsets. Below, we briefly summarize details about the implementation of the basic functions used in $METADOCK$.

- **Initialize** returns an initial set of solutions. $INEIni$ conformations are generated randomly for each of the m spots. Once they have been generated, a percentage ($PEIIni$) of the initial conformations of each spot is improved. The intensification of the improvement is indicated by the parameter $IIEIni$. Finally, $(PBEIni + PWEIni) * INEIni$ conformations from each spot are selected for the execution of the following functions. $PBEIni$ and $PWEIni$ represent the percentage of best and worst conformations to be included. The best conformations are those with the best value of the scoring function, and the "worst" conformations are selected from the remaining ones. Indeed, $METADOCK$ does not select only the best conformations, so as to diversify the search and avoid falling into local optima.
- **End condition** determines the stop criteria for $METADOCK$. Either, the maximum number of steps without improvement of the best solution from all the spots, $NIREnd$, or the maximum number of iterations, $MNIEnd$, is used to finish the execution.
- **Select** chooses some conformations to work with for the next phases. A percentage of the best and worst conformations relative to each spot are selected, i.e. $PBESel$ and $PWESel$. Again, to diversify the search and avoid local minima, not only "good" conformations are selected.
- **Combine** mixes conformations in pairs, depending on their scoring. Three parameters represent the percentage of best-best, worst-worst and best-worst conformations to be combined: $PBBCom$, $PWWCom$ and $PBWCom$. Combinations are performed among conformations at the same spot.
- **Improve** performs a local search within the neighborhood of some of the conformations previously generated by *Combine*. Two metaheuristic parameters are considered for each spot. First, $PEIImp$ defines the percentage of conformations that the local search will be applied to to improve those conformations. Second, $IIEImp$ establishes the number of trials for the local search. Hence, $METADOCK$ can generate hybrid metaheuristics with different degrees of intensification.
- **Include** updates the reference set for the next iteration of the schema. The parameter $PBEInc$ establishes the percentage of best conformations associated to each spot to be included in its reference set. The rest of the conformations to be included in this set are randomly selected from the remaining conformations at the spot. The inclusion of non-promising conformations contributes to diversify the search, so avoiding stalling in local minima.

GPU implementation of the scoring function The $METADOCK$ scoring function is based on the relevant non-bonded potentials typically used in VS calculations previously described in Background section. They are the Coulomb, or electrostatic, the Lennard-Jones potentials, and Hydrogen-Bounds interactions kernel. A discussion about the main terms included in the scoring function is beyond the scope of this paper, but we refer the reader to Wang et al. (2004) for insights.

Algorithm 2 Sequential baselines for the Lennard-Jones interactions between receptor and ligand.

```
for i=1 to $N\_CONFORMATION$ do
  for j=1 to $N\_ATOMS\_RECEPTOR$ do
    for k=1 to $N\_ATOMS\_LIGAND$ do
      Energy = 4*epsilon*(term12(j,k) - term6(j,k))
      Scoring += Energy
    end for
  end for
  S_energy[i] = Scoring
  Scoring = 0
end for
```

Algorithm 2 shows the sequential baselines for the Lennard-Jones potential interactions between a receptor and all conformations for a particular ligand (i.e. the set S in algorithm 1) to briefly illustrate the GPU implementation of the scoring function.

The scoring function of $METADOCK$ is implemented in a single kernel where all terms are calculated at the same time. We identify each candidate solution (i.e. conformation) to a CUDA warp, and warps are grouped into blocks depending on the CUDA thread block granularity. Algorithm

Table 2. The fifteen metaheuristic parameters used in the unified parameterized metaheuristic schema for $METADOCK$

Metaheuristic Parameters	Description
$INEIni$	Number of initial ligand conformations.
$PEIIni$	Percentage of the best conformations that are improved in the function Initialize.
$IIEIni$	The intensification of the improvement in the function Initialize.
$PBEIni$	Percentage of best conformations to be included in the initial set for the next iterations.
$PWEIni$	Percentage of worst conformations to be included in the initial set for the next iterations.
$PBESel$	Percentage of the best conformations to be selected for combination.
$PWESel$	Percentage of the worst conformations to be selected for combination.
$PBBCom$	Percentage of best-best conformations to be combined.
$PWWCom$	Percentage of worst-worst conformations to be combined.
$PBWCom$	Percentage of best-worst conformations to be combined between them.
$PEIImp$	Percentage of best conformations of the combination to be improved.
$IIEImp$	The intensification of the improvement of elements generated by combination.
$PBEInc$	Percentage of best conformations to be included in the reference set.
$NIREnd$	Maximum number of steps without improvement.
$MNIEnd$	Maximum number of iterations with or without improvement.

Algorithm 3 Method to calculate scoring on GPU

```
pos = atom_position
individual = get_individual()
for i=1 to r do
    Energy = 4*epsilon*(term12(i,pos) - term6(i,pos))
    Scoring += Energy
end for
synchronize_threads()
S_energy[individual] = Reduction_atoms_individual()
```

3 shows the CUDA kernel for the Lennard-Jones potential. Here, some performance strategies that we have applied to our codes to leverage NVIDIA GPU architectures are introduced:

- The use of shared memory facilitates the re-usability of data by threads of the same block. In our case, the compound is loaded into the shared memory so that threads within the same block can share this information, so saving costly device memory accesses. Thus, each thread calculates the scoring function corresponding to the elements that are associated to each of them, thus increasing the overall application bandwidth.
- The possibility of using shuffle instructions is available in devices with 3.X or higher compute capability, and their use can improve application performance substantially. These instructions enable information sharing within threads that belong to the same warp without using either shared or device memory.

In addition to the scoring function CUDA kernels, other kernels are also included in $METADOCK$ to implement different actions required by main schema functions shown in Algorithm 1. Among them, we may highlight the modification of ligand conformations in ($Initialize$ and $Improve$ functions). The main objective of these kernels is to keep the information in the GPU device memory, so avoiding data-movement through PCI-Express bus.

Scaling to a heterogeneous node

Algorithm 4 shows the parallelization scheme we use to leverage heterogeneous nodes with shared-memory multiprocessors and multiple GPUs. OpenMP is used to manage several CPU threads, where each thread is responsible for controlling a GPU context (we refer the reader to Chapman et al. (2008) for insights). Then, each GPU calculates the scoring function for a set of candidate solutions. In a *homogeneous distribution*, those candidate solutions are equally distributed among GPUs in the form of CUDA thread blocks. However, a HPC cluster may have different kinds of devices or even devices within the same family with different compute capabilities or improved instruction set architecture. Thus, the programmer plays a fundamental role in deciding where the code will run on each of those different architectures as long as performance is the main objective.

Algorithm 4 Scoring computation on a Parameterized Metaheuristic for multicore+multiGPU

```
omp_set_num_threads(number_GPUs)
#pragma omp parallel for
for i=1 to number_GPUs do
    Select_device(Devices[i].id)
    Host_To_GPU(Scom,Stmp)
    Conformations=Devices[i].conformations
    threads=Devices[i].Threadsblock
    stride=Devices[i].stride
    Calculate_scoring<Conformations/threads,threads>
    (Stmp+Devices[i].stride)
    GPU_To_Host(Scom,Stmp)
end for
```

With this scenario in mind, we introduce a heterogeneity distribution of the workload for our VS methodology. The execution time of each independent execution can differ, as it depends on (1) the underlying GPU each metaheuristic instance runs on, which is actually unknown at compile-time, and (2) the number of candidate solutions (the same in principle for all processors, but affected by GPU heterogeneity). Given that the slowest GPU will determine the overall execution time, our mission is to make use of the idle time offered by the most powerful GPUs. The peak performance differences shown in the last two rows of Table 1 lead us to believe that there is ample room for improvement. Nevertheless, it is worth noting that these are theoretical peak performances provided by the manufacturer, but only the different speed of the targeted GPUs is the factor for heterogeneity that should be considered to distribute the workload.

We have designed an implementation with two main focuses: (1) resources accounting through OpenMP processes and (2) performance monitoring via OpenMP threads. First, our algorithm defines a master thread which creates as many OpenMP threads as GPUs available on a node, which is easily attained by querying the GPU properties at runtime (using `cudaGetDeviceCount` from the CUDA API) and NVML (NVIDIA Management Library). Secondly, a *warm-up* phase is performed to establish performance differences among all targeted GPUs, running the scoring function for a few candidate solutions. This phase measures the execution time of a small number of iterations of the metaheuristic at run-time (once) in order to detect these differences. Importantly, at this stage, the algorithm is not trying to *solve* the docking problem in any meaningful sense, but these runs allow us to calculate the performance differences between GPUs. The execution times in this *warm-up* phase on all GPUs are reduced to obtain the maximum value using `omp reduction`. Thus, the *Percent* parameter is eventually determined, as shown in Equation (1)

$$Percent = \frac{Ex.time_{actualGPU}}{Ex.time_{slowestGPU}} \tag{1}$$

The slowest GPU will have $Percent = 1$; a GPU two times fast than the slowest GPU would have $Percent = 0.5$, and so on. Each OpenMP thread then calculates the number of conformations it is in charge of for the simulation. The optimum number of threads is also experimentally obtained at this stage to increase the level of parallelism and to maximize processor occupancy.

Experimental set-up

Experiments have been conducted in two different heterogeneous systems based on multicore+multiGPU configurations. Bellow, we show the main characteristics of these computational systems along with the metaheuristic scheme parameters we have used to generate different kinds of metaheuristics to test our implementations. Finally, a description of the target datasets is provided.

Hardware environment

Two different computational systems are used to run our experiments:

- **Jupiter**: is a system with two hexa-cores (12 cores) Intel Xeon E5-2620 at 2 GHz and 32 GB of RAM. The node has up to six GPUs. Two of them are GPUs NVIDIA Fermi Tesla C2075 with 448 CUDA cores (14 Streaming Multiprocessors and 32 Streaming Processors per Multiprocessor) running at boost clock of 1.15 GHz, giving a raw processing power of up to 1030 GFLOPS. The memory size is 5,3 GB of GDDR5. The other four GPUs are actually two NVIDIA Fermi GPUs GeForce GTX 590 that each contain two chips in turn. Both have 512 CUDA cores (16 Streaming Multiprocessors and 32 Streaming Processors per Multiprocessor) running at boost clock of 1.21 GHz, giving a raw processing power of up to 2488.3 GFLOPS.
- **Hertz**: has four Intel Xeon X7550 processors running at 2 GHz and plugged into a quad-channel motherboard endowed with 128 Gigabytes of DDR3 memory. It has two NVIDIA GPU. A GPU NVIDIA Tesla Kepler K40c with 2880 CUDA cores (15 Streaming Multiprocessors and 192 Streaming Processors per Multiprocessor) running at boost clock of 0.88 GHz, giving a raw processing power of up to 5068 GFLOPS. The memory size is 12 GB of GDDR5. A GPU NVDIA Fermi GeForce GTX 580 with 512 CUDA cores (16 Streaming Multiprocessors and 32 Streaming Processors per Multiprocessor) running at boost clock of 1.54 GHz, giving a raw processing power of up to 1581 GFLOPS

In both platforms, gcc 4.8.2 with the -O3 flag was used for compilation on the CPU, and the CUDA toolkit version 6.5 was used for compilation on the GPU.

Benchmarking

Metaheuristics The template shown in Algorithm 1 allows experimentation with several basic metaheuristics and combinations/hybridations of them. So, it can be used for the selection and tuning of satisfactory metaheuristics for the problem we are working with, and, furthermore, the parallelization of the schema facilitates the parallelization and the determination of the best parallelism configurations for different metaheuristics and combinations. In the experiments, we consider up to six metaheuristics of different characteristics for comparison purposes. The performance, efficiency and quality are evaluated on various hardware configurations.

Table 3. Parameter setting used for experimentation

	COMBINATIONS					
	M1	M2	M3	M4	M5	M6
$INEIni$	500	50	300	20	100	10
$PEIIni$	0	50	100	100	50	100
$IIEIni$	0	20	100	100	50	200
$PBEIni$	8	20	0	100	20	100
$PWEIni$	0	20	0	100	20	100
$PBESel$	100	100	0	50	100	100
$PWESel$	0	100	0	50	100	0
$PBBCom$	100	100	0	100	50	100
$PWWCom$	0	25	0	50	10	0
$PBWCom$	0	50	0	10	25	0
$PEIImp$	0	50	0	50	100	100
$IIEImp$	0	20	0	20	10	200
$PBEInc$	100	100	0	80	50	80

Table 3 shows the values of the metaheuristic parameters for the six metaheuristics considered in the experimental section. The first metaheuristic (M1) is a *Genetic Algorithm* with a population of 500 individuals ($INEIni = 500$) for each spot in the receptor at the initialization stage. Only the best 40 individuals keeps on going with the computation

($PBEIni = 8$) after initialization. After that, all best candidates are selected, combined and included for the next iteration ($\{PBESel, PBBCom, PBEInc\} = 100$), and no local search is included to improve the conformations. The second metaheuristic (M2) is also an evolutionary method but, in this case, its computation is similar to a *Scatter Search* algorithm. It works with a reference set of 50 individuals, many elements are improved after they have been generated, initially or by combination, through local search in the neighborhood of each element to obtain better solutions, and combinations between worst or best and worst elements are included. After the initialization phase, all the selected elements are combined with each other, and a further improvement is applied in half of them. The metaheuristic M3 is a neighborhood-based metaheuristic, where local searches are applied to candidate solutions for a large initial set, thus the initialization stage is the only stage executed in this metaheuristic, and it can be seen as a GRASP method. The metaheuristics M4, M5 and M6 are combinations of the above metaheuristics, changing the parameter values of combination and improvement, to try to get better results. For example, for larger sets less elements are improved or the intensification of the improvements is lower. The unified schema allows us to experiment with several configurations to determine the best metaheuristic for our problem, but this study is beyond the scope of the paper.

Accuracy of the predictions It is necessary to measure the performance of VS methods in terms of accuracy of the predictions they yield. One common approach in this research area is to process datasets of known compounds and to check the ability of VS methods in classifying them. A set of benchmark instances from the Directory of Useful Decoys (DUD) was used for this DUD (2006). The DUD dataset contains, for 40 sets of protein-targets, a set of active ligands, decoys (ligands known to be not active) and the structural information (X-ray crystallographic studies) about a ligand co-crystallized with each respective protein. The decoy compounds have similar physical properties to the active ligands but dissimilar topology, and were designed in order to make the classification task difficult. In this work, three different targets are selected from DUD (see Table 4). These targets have different numbers of atoms (13261, 7158 and 3419 respectively) that require different amount of memory in the simulation.

Performance datasets Data sets in Table 4 are also used to evaluate computational performance of our parallelization strategies. The docking simulations, in this case, are performed with the different receptors (GPB, SRC and COMT) and their corresponding crystallography ligands. They have different sizes to test scalability on the different platforms targeted. Our scoring function calculates the interaction between all atoms from the protein ($nrec$) and all atoms from the ligand ($nlig$). This is performed at each spot where a number of individuals (the same ligand with different spatial locations) runs in parallel. Therefore, a $METADOCK$ simulation performs $Spots * Individuals$ simulations at the same time, and each calculates $nrec * nlig$ interactions. For instance, M1 works with 500 conformations at the same time in the initialization stage. The number of interactions for GPB would be $13,261 * 52 * 500 = 344,786,000$ at each spot, having up to 813 spots. The number of spots, receptor atoms and crystallography ligand atoms are listed in Table 4. Indeed, memory requirements for our simulations are directly related to this formula. The number of bytes for these benchmarks is shown.

Experimental results

This section shows the experimental results for $METADOCK$ on multicore and multiGPU systems. The main objective of these experiments is two-fold. First, we analyze our load distribution strategies to improve performance on heterogeneous nodes based on CPU and MultiGPU. Second, we study the quality of the results with several chemical compounds to discuss the effectiveness of our approach.

Performance results

Given that our technique establishes the experimental set-up dynamically, the results shown below are platform-dependent. Therefore, we provide an exhaustive analysis on the two heterogeneous systems previously described. Tables 5 and 6 show the execution time (single-point precision execution) and relative speed-up factor among different implementations and metaheuristic configurations for each target dataset in both systems (see Table 4 for dataset description and Table 3 for the schema configuration). They show the execution times for OpenMP implementation on multicore CPUs as a reference for the improvements.

Table 5 shows performance numbers on the Hertz system, in which the computational capability of the GPUs available are quite different (Fermi and Kepler architecture). First, it shows the speed-up factor for a single GPU (Tesla K40c) versus multicore CPU. This is in the range of 22-29x. The implicit data parallelism in this problem benefits greatly from the GPU horsepower. Our CUDA implementations take advantage of data-locality through tilling implementation via shared memory, which benefits the receptor scalability. Then, computing with several GPUs come into the scene. First, our version using a homogeneous distribution of the workload (i.e. assigning the same amount of workload to each GPU) is in the range 26-35x speed-up factor compared to multicore CPU for small-medium molecules. This speed-up for large molecules is up to 37x. This means an additional 1.27x speed-up factor by adding a GPU in some cases. Furthermore, these data show that metaheuristic parameters are important for the performance, that is enhanced when the number of combined and improved individuals increases. This is clearly shown in metaheuristic M3 for small-medium compounds where all initial population generated is improved, and metaheuristics M3 and M6 with the largest compound. In this case, in M6 another large improve phase for all combined elements produces a significant increase in computation.

A further step to increase performance is to figure out a new distribution strategy for load balancing. A straightforward approach is to distribute the workload according to the hardware features (i.e. peak performance). The theoretical peak performance for Tesla K40c and GeForce GTX 580 are 5068 and 1581 GFLOPS respectively. This means a 3.2x speed-up factor in favor of Tesla

Table 4. Number of decoys and active ligands of the benchmark targets from DUD database. We also include the size of receptor and crystallography ligand (number of atoms) used for performance comparison, and the number of interactions for each individual

Targets	Number of spots	Decoy ligands	Active ligands	Receptor size (number of atoms)	Crystallography ligand size (number of atoms, KB)	Target memory size (KB)
GPB	813	2,139	52	13,261	(52, 1.21)	190
SRC	452	6,319	159	7,158	(67, 1.57)	293
COMT	214	468	11	3,419	(29, 0.67)	543

K40c. The strategy called hardware-feature distribution in Table 5 schedules the workload based on this idea. Some performance gains are reported compared to the homogeneous approach although they are not remarkable. Peak performance reported in technical specifications is reported under ideal conditions (e.g. the execution of only FMA instructions, not memory latencies, etc) and thus application performance will be determined at runtime. The last column in Table 5 shows the speed-up factor of our heterogeneous distribution strategy compared to multicore. The results are better than with the hardware-feature distribution, reaching up to 1.43x speed-up factor on average compared to a homogeneous approach.

Table 6 shows performance numbers in Jupiter. The GPUs available in Jupiter (up to six) are based on the same architecture (code-named Fermi) and, thus, the overall GPU heterogeneity in this system is very low. Performance numbers in this system raise similar conclusions to those in Hertz. Our heterogeneous distribution strategy of the workload here shows fewer benefits. Indeed, this means that GPU heterogeneity on a computing node makes load balancing strategy mandatory to get peak performance. Finally, Tables 5 and 6 show that the speed-up increases with the problem size, thus proving the scalability of the multiGPU versions.

We report higher speed-up ratios whenever we increase either the level of intensification in a local search or the size of the reference set. Metaheuristics M2 and M3 contain different values for local search in the neighborhood of each conformation with the same number of initial elements. In all the executions with the three compounds, more intensive searches provide higher speed-up ratios, and they are even higher in multiGPU environments. The M4 metaheuristic studies the extreme case in which only local search is applied on a very large number of elements, achieving the best speed-up ratios in comparison with the distributed metaheuristics.

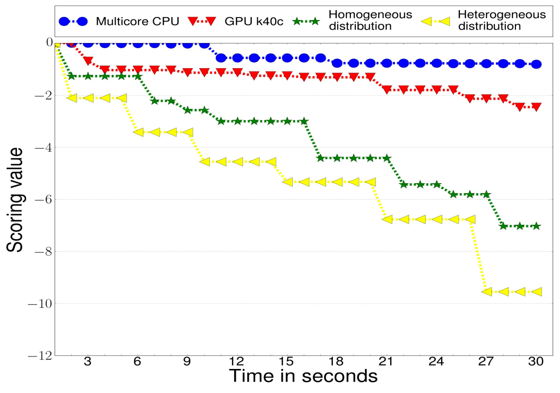

Figure 2. Scoring function evolution within 30 seconds time-frame. Metaheuristic $M6$ and target COMT on Hertz.

Finally, Figure 2 shows the scoring function evolution during a simulation of 30 seconds for a docking simulation with $METADOCK$ with $M6$ parameter configuration and the target COMT. The techniques with the highest performance obtains better quality results as it performs more computations within the same time-frame. So, the efficient exploitation of parallelism facilitates obtaining satisfactory results at shorter times.

Quality results: accuracy of the predictions

As mentioned in the benchmark section, the quality of a docking program is usually measured through its ability to differentiate between active ligands (which could evolve into drugs) and decoys.

The ROC (Receiver Operating Characteristic) curve is a binary classification model which is frequently applied for the analysis of the accuracy of virtual screening methods. This model allows us to compare how good a method is when selecting active ligands and discarding decoys. In order to validate our algorithm, Figure 3 shows the ROC curves obtained for three different targets conveniently selected from the DUD database. The R language package ROCR (published by Sing et al. (2005)) was used to perform the analyses over generated docking results. The y-axis shows the fraction of true positives (TPF), and that of false positives (FPF) is shown on the x-axis. A diagonal line would indicate that the classifier works randomly. The value of AUC (Area Under Curve) is 1.0 when TPR is always 1 and FPR equal to 0 (the ideal case). The results shown in Figure 3 have been obtained with $METADOCK$ with the combination of metaheuristic parameters M4 in Table 2, and they are representative of those obtained with the other configurations (results not shown). The AUC values obtained for the DUD datasets GPB, SRC and COMT are 0.838, 0.842 and 0.747, respectively. AUC Values greater than 0.65 can be considered to be adequate, with values closer to 1 indicating better specificity and sensitivity of the method in detecting decoys. The values obtained with $METADOCK$ are satisfactory, and the efficient exploitation of heterogeneous computing systems facilitates experimentation with moderate execution times.

Conclusions and future work

Virtual screening (VS) methods are computational techniques that aid or complement the experimental drug discovery process but they are very computationally demanding applications. This paper introduces a VS technique, called $METADOCK$, based on an unified parameterized metaheuristic schema that is able to generate a wide variety of metaheuristics, so providing a fully flexible framework

Table 5. Execution time (in seconds) and speed-up for the metaheuristics, executing the targets COMT, SRC, GPB in **Hertz**

Metaheuristics	Multicore CPU (in seconds)	Speed-up GPU K40c Vs Multicore CPU	Speed-up Homogeneous distribution Vs Multicore CPU	Speed-up Hardware-feature distribution Vs Multicore CPU	Speed-up Heterogeneous distribution Vs Multicore CPU
DUD:COMT Target					
M1	180.19	24.32	27.35	29,90	37.91
M2	223.11	23.88	28.23	29.51	36.92
M3	2,942.99	28.21	35.08	36.94	46.56
M4	284.31	23.35	26.56	29.88	35.56
M5	402.25	24.33	29.35	30.67	38.49
M6	1,758.24	24.64	30.61	31.79	39.75
DUD:SRC Target					
M1	1,025.19	23.21	26.12	28.82	37.64
M2	1,269.99	22.66	27.97	28.89	36.77
M3	15,042.33	23.03	26.66	29.46	37.98
M4	1,616.11	22.26	27.62	30.16	38.05
M5	2,287.21	22.81	23.25	28.11	37.16
M6	10,015.06	22.94	24.51	29.27	37.57
DUD:GPB Target					
M1	1,479.51	28,56	31,71	36.19	45.61
M2	1,831.24	28.94	35.78	36.88	47.29
M3	21,758.21	29.47	37.42	37.75	48.57
M4	2,335.44	29.07	36.43	37.06	47.39
M5	3,303.33	29.11	36.51	37.32	47.96
M6	14,439.52	29.34	37.07	38.26	48.45

Table 6. Execution time (in seconds) and speed-up for the metaheuristics, executing the targets COMT, SRC, GPB in **Jupiter**

Metaheuristics	Multicore CPU (in seconds)	Speed-up GPU Tesla C2075 Vs Multicore CPU	Speed-up Homogeneous distribution Vs Multicore CPU	Speed-up Heterogeneous distribution Vs Multicore CPU
DUD:COMT Target				
M1	140.48	10.42	38.72	39.35
M2	193.67	13.81	39.88	43.55
M3	1,911.52	9.62	53.25	54.16
M4	209.56	9.59	33.86	34.43
M5	262.65	8.55	34.81	35.51
M6	1,379.93	10.39	53.61	53.89
DUD:SRC Target				
M1	639.32	7.45	36.43	39.35
M2	678.41	7.86	37.89	40.97
M3	10,670.57	8.55	49.34	49.41
M4	1,150.81	8.49	40.21	43.75
M5	1,574.79	8.27	43.08	44.62
M6	7,422.66	8.93	50.08	50.27
DUD:GPB Target				
M1	910.42	9.24	45.19	46.21
M2	964.82	9.44	49.01	53.01
M3	15,050.81	10.75	60.98	62.28
M4	1,654.85	10.94	52.88	57.58
M5	2,449.27	11.47	58.71	62.47
M6	10,186.09	10.94	61.57	62.41

for drug discovery, and thus facilitating enhanced performance and prediction accuracy. $METADOCK$ is tailored for heterogeneous computers based on CPU and multiple GPUs. This heterogeneity may limit acceleration which is not acceptable in such challenging applications. In this work the heterogeneity of the system is exploited at two levels: (1) CPU-GPU heterogeneity; with an implementation in which some parts of the computation are carried out on the CPU side while the most costly parts are assigned to the GPUs, and (2) GPU-GPU heterogeneity, i.e. GPUs with different characteristics, including different architectures, numbers of cores and compute capabilities, providing a load balancing technique based on the application performance on the targeted GPUs.

The efficient exploitation of the heterogeneous system provides good speed-ups, which increase with the problem size. Our strategy is particularly useful for non-deterministic algorithms and stochastic behaviors, where real-time

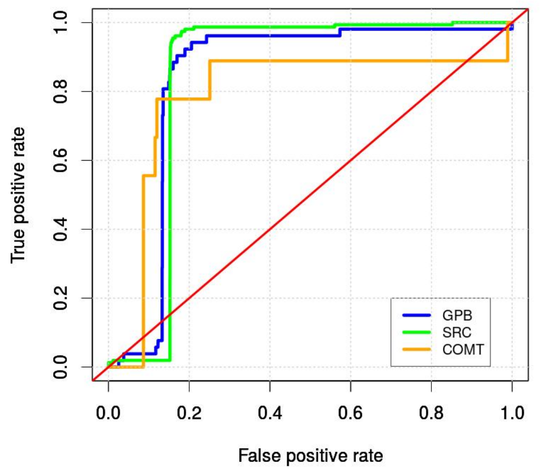

Figure 3. ROC plots for three targets of the DUD database: GPB (blue), SRC (green) and COMT (orange).

constraints must be fulfilled. Performance gains are translated into quality improvements that are a decisive factor in virtual screening. AUC results obtained with $METADOCK$ support that its parallel, metaheuristic-based schema makes it a useful tool in the early stages of drug discovery.

For future work and to tackle larger problems or for better solutions with limited execution times, it could be convenient to adapt our virtual screening method to more complex hardware systems, comprising several computational nodes working together with the message-passing paradigm, and each node with several computational components, e.g., multicore, heterogeneous GPUs and MICs. In this scenario, multicore processors could also be used to compute part of the simulation instead of only monitoring the GPU. Regarding the quality of the results, many other types of scoring functions are still to be explored, including metal and aromatic interactions, and inclusion of implicit solvation effects. This field seems to offer a promising and potentially fruitful area of research.

Acknowledgements

This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 18946/JLI/13, and by the Spanish MEC and European Commission FEDER under grants TIN2015-66972-C5-3-R and TIN2016-78799-P (AEI/FEDER, UE). We also thank NVIDIA for hardware donation under GPU Educational Center 2014-2016 and Research Center 2015-2016.

References

Almeida F, Giménez D, López-Espín JJ and Pérez-Pérez M (2013) Parameterised schemes of metaheuristics: basic ideas and applications with Genetic algorithms, Scatter Search and GRASP. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 43(3): 570–586.

Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW and Yelick KA (2006) The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, University of California at Berkeley, Electrical Engineering and Computer Sciences.

Austin T (2015) Bridging the Moore's Law Performance Gap with Innovation Scaling. In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, p. 1.

Bianchi L, Dorigo M, Gambardella LM and Gutjahr WJ (2009) A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an international journal* 8(2): 239–287.

Blum C, Puchinger J, Raidl GR and Roli A (2011) Hybrid metaheuristics in combinatorial optimization: A survey. *Appl. Soft Comput.* 11(6): 4135–4151.

Blum C and Roli A (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)* 35(3): 268–308.

Cecilia JM, García JM, Nisbet A, Amos M and Ujaldón M (2013) Enhancing data parallelism for ant colony optimization on GPUs. *Journal of Parallel and Distributed Computing* 73(1): 42–51.

Chapman B, Jost G and Van Der Pas R (2008) *Using OpenMP: portable shared memory parallel programming*, volume 10. MIT press.

Cutillas-Lozano JM and Giménez D (2013) Determination of the kinetic constants of a chemical reaction in heterogeneous phase using parameterized metaheuristics. In: *ICCS*.

Cutillas-Lozano LG, Cutillas-Lozano JM and Giménez D (2012) Modeling shared-memory metaheuristic schemes for electricity consumption. In: *Distributed Computing and Artificial Intelligence - 9th International Conference*. pp. 33–40.

De Michell G and Gupta RK (1997) Hardware-software co-design. *Proceedings of the IEEE* 85(3): 349–365.

Dolezal R, Ramalho TC, França TC and Kuca K (2015) Parallel flexible molecular docking in computational chemistry on high performance computing clusters. In: *Computational Collective Intelligence*. Springer, pp. 418–427.

Dréo J, Pétrowski A, Siarry P and Taillard E (2005) *Metaheuristics for Hard Optimization*. Springer.

Drews J (2000) Drug discovery: a historical perspective. *Science* 287(5460): 1960–1964.

DUD (2006) Directory of Useful Decoys. `http://dud.docking.org/`. (accessed, October, 4th, 2016).

Ewing TJA, Makino S, Skillman AG and Kuntz ID (2001) DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer-Aided Molecular Design* 15(5): 411–428.

Franco AA (2013) Multiscale modelling and numerical simulation of rechargeable lithium ion batteries: concepts, methods and challenges. *RSC Advances* .

Friesner RA, Banks JL, Murphy RB, Halgren TA, Klicic JJ, Mainz DT, Repasky MP, Knoll EH, Shelley M, Perry JK and et al (2004) Glide: A New Approach For Rapid, Accurate Docking and Scoring: Method and Assessment of Docking Accuracy. *Journal of Medicinal Chemistry* 47(7): 1739–1749.

Glover F and Kochenberger GA (2003) *Handbook of Metaheuristics*. Kluwer.

Guerrero GD, Cebrián JM, Pérez-Sánchez H, García JM, Ujaldón M and Cecilia JM (2014) Toward energy efficiency in heterogeneous processors: findings on virtual screening methods. *Concurrency and Computation: Practice and Experience* 26(10): 1832–1846.

Hromkovič J (2003) *Algorithmics for Hard Problems*. Second edition. Springer.

Huang SY and Zou X (2010) Advances and Challenges in Protein-Ligand Docking. *International journal of molecular sciences* 11(8): 3016–3034.

Imbernón B, Cecilia JM and Giménez D (2016) Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems. In: *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM, pp. 50–58.

Irwin JJ and Shoichet BK (2005) ZINC–a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling* 45(1): 177–182.

Jain AN (2006) Scoring functions for protein-ligand docking. *Current Protein and Peptide Science* 7(5): 407–420.

Jorgensen WL (2004) The Many Roles of Computation in Drug Discovery. *Science* 303: 1813–1818.

Kirk DB and Wen-mei WH (2013) *Programming massively parallel processors: a hands-on approach*. Elsevier.

Kitchen DB, Decornez H, Furr JR and Bajorath J (2004) Docking and scoring in virtual screening for drug discovery: methods and applications. *Nature Reviews Drug Discovery* 3(11): 935–949.

Kuntz SK, Murphy RC, Niemier MT, Izaguirre JA and Kogge PM (2001) Petaflop Computing for Protein Folding. In: *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*. pp. 12–14.

Li Y, Han L, Liu Z and Wang R (2014a) Comparative assessment of scoring functions on an updated benchmark: 2. evaluation methods and general results. *Journal of chemical information and modeling* 54(6): 1717–1736.

Li Y, Liu Z, Li J, Han L, Liu J, Zhao Z and Wang R (2014b) Comparative assessment of scoring functions on an updated benchmark: 1. compilation of the test set. *Journal of chemical information and modeling* 54(6): 1700–1716.

Lionta E, Spyrou G, K Vassilatis D and Cournia Z (2014) Structure-based virtual screening for drug discovery: principles, applications and recent advances. *Current topics in medicinal chemistry* 14(16): 1923–1938.

López-Camacho E, García-Godoy MJ, García-Nieto J, Nebro AJ and Montes JFA (2015) Solving molecular flexible docking problems with metaheuristics: A comparative study. *Appl. Soft Comput.* 28: 379–393.

McIntosh-Smith S, Price J, Sessions RB and Ibarra AA (2014) High performance in silico virtual drug screening on many-core processors. *International Journal of High Performance Computing Applications* : 1094342014528252.

Michalewicz Z and Fogel DB (2002) *How to Solve It: Modern Heuristics*. Springer.

Minetti G, Alba E and Luque G (2008) Seeding strategies and recombination operators for solving the DNA fragment assembly problem. *Inf. Process. Lett.* 108(3): 94–100.

Morris GM, Goodsell DS, Halliday RS, Huey R, Hart WE, Belew RK and Olson AJ (1998) Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry* 19(14): 1639–1662.

NVIDIA Corporation (2014) *NVIDIA CUDA C Programming Guide 6.5*.

Raidl GR (2006) A unified view on hybrid metaheuristics. In: *Hybrid Metaheuristics*. Springer, pp. 1–12.

Rester U (2008) From virtuality to reality-Virtual screening in lead discovery and lead optimization: a medicinal chemistry perspective. *Current opinion in drug discovery & development* 11(4): 559–568.

Rollinger JM, Stuppner H and Langer T (2008) Virtual screening for the discovery of bioactive natural products. In: *Natural Compounds as Drugs Volume I*. Springer, pp. 211–249.

Rozenberg G, Bäck T and Kok JN (2011) *Handbook of Natural Computing*. Springer.

Sánchez-Linares I, Pérez-Sánchez H, Cecilia JM and García JM (2012) High-throughput parallel blind virtual screening using BINDSURF. *BMC Bioinformatics* 13(Suppl 14): S13.

Schneider G (2002) Virtual screening and fast automated docking methods. *Drug Discovery Today* 7: 64–70.

Sing T, Sander O, Beerenwinkel N and Lengauer T (2005) Rocr: visualizing classifier performance in r. *Bioinformatics* 21(20): 3940–3941.

Sodani A, Gramunt R, Corbal J, Kim HS, Vinod K, Chinthamani S, Hutsell S, Agarwal R and Liu YC (2016) Knights landing: Second-generation intel xeon phi product. *IEEE Micro* 36(2): 34–46.

Song WJ, Mukhopadhyay S and Yalamanchili S (2016) Amdahl's law for lifetime reliability scaling in heterogeneous multicore processors. In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 594–605.

Top500 (2016) Top500 supercomputer site. http://www.top500.org/. (accessed, October, 4th, 2016).

Vaessens RJ, Aarts EH and Lenstra JK (1998) A local search template. *Computers & Operations Research* 25(11): 969–979.

Wang J, Deng Y and Roux B (2006) Absolute Binding Free Energy Calculations Using Molecular Dynamics Simulations with Restraining Potentials. *Biophys J* 91(8): 2798–2814.

Wang R, Lu Y, Fang X and Wang S (2004) An extensive test of 14 scoring functions using the pdbbind refined set of 800 protein-ligand complexes. *Journal of chemical information and computer sciences* 44(6): 2114–2125.

Yuriev E, Agostino M and Ramsland PA (2011) Challenges and Advances in Computational Docking: 2009 in Review. *Journal of Molecular Recognition* 24(2): 149–164.

Yuriev E and Ramsland PA (2013) Latest developments in molecular docking: 2010–2011 in review. *Journal of Molecular Recognition* 26(5): 215–239.

Zhou Z, Felts AK, Friesner RA and Levy RM (2007) Comparative performance of several flexible docking programs and scoring functions: enrichment studies for a diverse set of pharmaceutically relevant targets. *Journal of Chemical Information and Modeling* 47(4): 1599–1608.

Short Biography

1. Baldomero Imbernón received his B.S. degree in Computer Science from Catholic University of Murcia (Spain, 2013) and M.S. degree in New Technologies in Computer Science in University of Murcia (Spain, 2015) specialism High Performance Architectures and Supercomputing. In the last two years, he has authored several journal papers in the areas of bioinformatics and high performance computing. Actually, he is a predoctoral researcher at the Catholic University of Murcia (Spain).

2. José M. Cecilia received his B.S. degree in Computer Science from the University of Murcia (Spain, 2005), his M.S. degree in Computer Science from the University of Cranfield (United Kingdom, 2007), and his Ph.D. degree in Computer Science from the University of Murcia (Spain, 2011). Dr. Cecilia was predoctoral researcher at Manchester Metropolitan University (United Kingdom, 2010), supported by a collaboration grant from the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC) and visiting professor at the Impact group leaded by Professor Wen-Mei Hwu at University of Illinois (Urbana, IL, USA). He has published several papers in international peer-reviewed journals and conferences. His research interest includes heterogeneous architecture as well as bio-inspired algorithms for evaluating the newest frontiers of computing. He is also working in applying these techniques to challenging problems in the fields of Science and Engineering. Now, he is working as Assistant Professor at the Computer Science Department in the Catholic University of Murcia. He is teaching several lectures such as Introduction to Parallel Computing, Object-Oriented Programming, Operative System, Computer Architecture, Computer Graphics; all of them are part of the Computer Science degree.

3. Horacio Pérez Sánchez. I have contributed to computational and physical chemistry with several methodological developments, implementing them in high performance computing (HPC) architectures (Supercomputers, GPUs, Cell Processor) so that they can be accessed by other researchers either as standalone software packages or web tools. Some of them have been commercialized directly with industry. I have applied these developments directly in drug discovery projects (anticoagulants, Parkinson, Alzheimer, cancer, Fabry disease, anti-inflammatories, etc) or for the analysis and interpretation of experimental data (encapsulation processes, ion channels, nutraceuticals, etc). Main results have been published in 56 ISI indexed scientific articles (8 as first author and 20 as corresponding author), 2 international (and licensed) patents, 50 contributions to international conferences with 45 peer reviewed computer engineering and bioinformatics conference proceedings (6 as first author and 21 as corresponding author) and 17 invited talks. I have been awarded with the 2016 HiPEAC technology transfer award and participated in 30 research projects attracting around 725.000 (I have been Principal Investigator in 16 of them, with total funding 327.000), and established 4 contracts with industry and academy.

4. Domingo Giménez is a Professor in the Computer Science Department at the University of Murcia, Spain. He has been a faculty member of the university since 1988, where he teaches algorithms and parallel computing. He received his degree in Mathematics from the University of Murcia in 1982, and his Ph.D in Computer Science from the Polytechnic University of Valencia in 1995. His research interests include scientific applications of parallel computing, matrix computation, scheduling and software auto-tuning techniques.