Enhancing Metaheuristic-based Virtual Screening Methods on Massively Parallel and Heterogeneous Systems

Baldomero Imbernón
Polytechnic School
Catholic University of San Antonio of Murcia (UCAM)
Murcia, Spain
bimbernon@alu.ucam.edu

José M. Cecilia
Polytechnic School
Catholic University of San Antonio of Murcia (UCAM)
Murcia, Spain
jmcecilia@ucam.edu

Domingo Giménez
Department of Computing and Systems
University of Murcia
Murcia, Spain
domingo@um.es

ABSTRACT

Molecular docking through Virtual Screening is an optimization problem which can be approached with metaheuristic methods. The interaction between two chemical compounds (typically a protein or *receptor* and small molecule or *ligand*) is measured with computationally very demanding scoring functions and can, moreover, be measured at several spots throughout the receptor. For the simulation of large molecules, it is necessary to scale to large clusters to deal with memory and computational requirements. In this paper, we analyze the current landscape of computation, where massive parallelism and heterogeneity are today the main ingredients in large-scale computing systems, to enhance metaheuristic-based virtual screening methods, and thus facilitate the analysis of large molecules. We provide a parallelization strategy aimed at leveraging these features. Our solution finds a good workload balance via dynamic assignment of jobs to heterogeneous resources which perform independent metaheuristic executions under different molecular interactions. 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.

CCS Concepts

•**Applied computing** → **Bioinformatics;** •**Theory of computation** → *Massively parallel algorithms; Shared memory algorithms; Vector / streaming algorithms;*

Keywords

Virtual Screening; metaheuristics; GPU; heterogeneous computing

PMAM'16, March 12-16, 2016, Barcelona, Spain

DOI: http://dx.doi.org/10.1145/2883404.2883413

1. INTRODUCTION

Metaheuristics are frequently used to solve NP-hard problems [31]. Some of these problems are in the field of *Bioinformatics*, e.g., DNA analysis [23] or molecular docking [21]. Of particular interest to us are Virtual Screening (VS) methods [18], which are widely used to enhance the drug discovery process by using computational tools to look for potential drug candidates [29, 30]. Given a receptor protein, large libraries of small molecules (*ligands*) are explored to search for the structures which best bind to the receptor, so generating an optimization problem. Metaheuristics can be used to approach the optimum solution of the problem. A scoring function is used for the fitness of the binding. The score is calculated for several positions of the ligand in the neighborhood of the spots in the protein, and it includes computations between pairs of atoms in the protein and the ligand [29, 30].

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, and thus additional experiments need to be carried out with different metaheuristics and hybridations of basic metaheuristics to discover the best solution to the problem in hand. Additionally, for any particular metaheuristic, a tuning process is traditionally conducted to select appropriate values of some parameters in the metaheuristic. The experimentation with several metaheuristics and their tuning process drastically increases the computational cost.

Of particular interest to us is the parallelism nature that many metaheuristics have by definition, especially those inspired by natural processes, such as genetic algorithms or particle swarm optimization. They are population-based, i.e., they maintain a collection of individual solutions to evolve as the computation proceeds. These algorithms are inherently stochastic, as they use randomization search techniques. Their internal structure demands parallelization, and so, abundant parallel versions have appeared recently [2]. This kind of algorithm is better suited for the current massively parallel landscape of computation.

Indeed, we are witnessing the steady transition to heterogeneous computing systems [1], with heterogeneity representing systems where nodes combine traditional multicore architectures (CPUs) and accelerators (mostly Nvidia GPUs [25] or Intel Xeon Phi cards [27]). Heterogeneity limits sys-

tem growth as it can no longer 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 [6]. The run-time system is still too immature to map processors and computations efficiently. In the meantime, the scientific community is focusing on the latest breakthroughs in high performance computing together with specific fields of interest (metaheuristics, image processing, computational modeling, and so on). This results in a vertical approach enabling remarkable advances in computer-driven scientific simulations, the so-called hardware-software co-design [7].

In this paper, we analyze several kinds of metaheuristics with different algorithmic patterns as applied to Virtual Screening on large-scale massively parallel and heterogeneous systems. Indeed, the volume of data and the processing time are very high whenever virtual screening methods target large, more complex scenarios, and large-scale computational systems should be considered. Therefore, the metaheuristics are designed to leverage multiGPU-based systems that also contain multicore processors. Furthermore, the GPUs in the system may have different computational capabilities, which introduces additional heterogeneity to that inherent to the use of multicore together with GPU. We therefore analyze the heterogeneity inside a node.

The rest of the paper is structured as follows. Section 2 describes the scoring problem we are working with and some relevant knowledge about metaheuristics and Nvidia GPUs. Section 3 shows our metaheuristic-based virtual screening technique and its design for heterogeneous nodes based on multicore processors and multiGPU. Section 4 describes the conditions in which experiments are conducted, and Section 5 shows the results of some experiments in a heterogeneous cluster, with nodes at different speeds and with different number of cores and GPUs, which are also of several types. Finally, Section 6 summarizes the conclusions and gives some directions for future work.

2. BACKGROUND

The principal characteristics of Virtual Screening methods and of the application to them of metaheuristics are briefly discussed here, together with those of the CUDA programming model used for the development of metaheuristics in multicore+multiGPU.

2.1 Virtual Screening

We draw on our description of the Virtual Screening (VS), which was first given in [14, 32]. VS methods are computational techniques used in several scientific areas, such as catalysts and energy materials [11], and mainly drug discovery [19], where experimental techniques can benefit from computational simulation.

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. They actually "dock" 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* [33]. 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 [17]. Indeed, these candidate molecules will continue the drug discovery process roadmap that goes from in-vitro studies, to animal investigations and, eventually, to human trials [9].

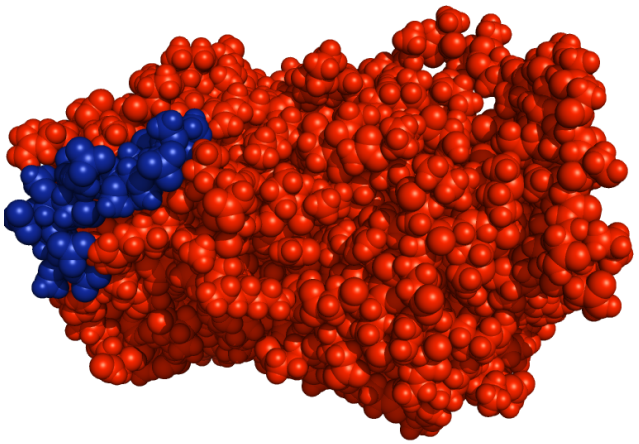

Figure 1: Binding two molecules; receptor (red) and ligand (blue)

Although VS methods have been investigated 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, even thousands of CPU hours for each ligand when simulation strategies are used to compute absolute binding affinities [35].

The relevant non-bonded potentials used in VS calculations are the Coulomb, or electrostatic, and the Lennard-Jones potentials, since they describe very accurately the most important short and long-range interactions between atoms of the protein-ligand system. They are also the most-time consuming calculations in VS methods. For example, in Molecular Dynamics, the calculation of these kernels takes up to 80% of the total execution time [20].

Within those VS methods, of particular interest to us are protein-ligand docking [36, 16] techniques. Docking simulations are typically carried out on the protein surface using known methods like Autodock [24], Glide [12] and DOCK [10]. This 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.

BINDSURF [32] uses GPUs to overcome this problem by dividing the whole protein surface into arbitrary independent regions (or spots). Using the parallelism of GPUs, a large ligand database is screened against the target protein over its whole surface simultaneously, 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.

2.2 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 [4, 8, 13, 15, 22], which are tuned for the problem in hand.

Metaheuristics include an abstraction layer that may provide a sufficiently good solution for an optimization problem, especially with limited computation capacity or inexact information [3]. They reduce the search space, focusing only on the most promising areas, and thus they cannot guarantee the analysis of all possible solutions, which means that they cannot guarantee to find the optimal solution. There are many metaheuristic algorithms with different characteristics [5] that can provide several good solutions to the same problem. Among them, we highlight:

- *Distributed metaheuristics* 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* 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*.

Metaheuristics have been successfully applied to a wide variety of application domains [31]. Of particular interest to us are those that apply metaheuristics to the field of *Bioinformatics*; among them we highlight DNA analysis [23] or molecular docking [21].

2.3 The CUDA programming model

We briefly introduce the CUDA programming model and refer the reader to [25] for insights. Compute Unified Device Architecture (CUDA) is a platform for Graphics Processing Units (GPUs), covering both hardware and software. On the hardware side, the GPU consists of N multiprocessors which are replicated within the silicon area, each endowed with M cores sharing the control unit, and of a shared memory (a small cache explicitly managed by the programmer). Each GPU generation has increased the CUDA Compute Capabilities (CCC), as well as the number of cores and the shared memory size (see Table 1). In conjunction with these developments, power consumption has been reduced by a factor of 2 at each new generation.

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 a SIMD (Single Instruction Multiple Data) fashion (that is, with all threads running the same code but having different IDs). The programmer deploys parallelism by declaring a *grid* composed of blocks equally distributed among all 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.

3. METAHEURISTICS FOR VIRTUAL SCREENING IN 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 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 algorithm which can be also heterogeneous or homogeneous. This section shows our proposal for metaheuristic-based virtual screening applications that leverage massively parallel and heterogeneous systems. We introduce the reader to the design of our virtual screening approach before presenting a homogeneous parallel version of the algorithm. Finally, the heterogeneous version is given.

3.1 Designing metaheuristics for VS methods

Our Virtual Screening technique divides the whole protein surface into arbitrary and independent regions (or spots). Spots are identify by finding out a specific type of atoms in the protein. All these spots are independent from each other and, thus, they offer great opportunities for data-based parallelization. Then, docking simulations for each ligand are 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.* individual or conformation) are different from each other as they have a different position and orientation with respect to each spot. Indeed, they 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. For simplicity our VS technique uses a scoring function based on the Lennard-Jones potential.

Algorithm 1 shows a generic template that we use to generate several metaheuristics for the virtual screening problem. Several authors agree [28, 34] 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, often using population characteristics to guide the search. Some examples of population-based metaheuristics include evolutionary computation, ge-

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

Hardware generation and starting year	Tesla 2007	Fermi 2010	Kepler 2012	Maxwell 2014
Multiprocessors per die (up to)	30	16	15	16
Cores per multiprocessor	8	32	192	128
Total number of cores (up to)	240	512	2880	2048
Shared memory size (maximum in kilobytes)	16	48	48	64
CUDA Compute Capabilities (CCC)	1.x	2.x	3.x	5.x
Peak single-precision performance (GFLOPS)	672	1178	4290	4980
Performance per watt (approx. and normalized)	1	2	6	12

netic algorithms, and particle swarm optimization.

Algorithm 1 Generic template for metaheuristic design

```
Initialize(S)
while no End(S) do
  Select(S,Ssel)
  Combine(Ssel,Scom)
  Improve(Scom)
  Include(Scom,S)
end while
```

Each of the functions in Algorithm 1 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 be also improved by applying a local search; i.e. moving, translating and/or rotating with respect to each spot.

3.2 Homogeneous parallelization strategy

Algorithm 2 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. Then, each GPU calculates the scoring function for a set of candidate solutions. In our homogeneous implementation, those candidate solutions are equally distributed among GPUs in form of CUDA thread blocks. Actually, we identify each candidate solution to a CUDA warp, and warps are grouped into blocks depending on the CUDA thread block granularity.

Algorithm 2 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
```

Moreover, an additional structure called *devices* is created to manage several configuration parameters. This structure stores the number of conformations assigned to each GPU and some GPU runtime parameters such as memory, grid size, maximum threads per block, and so on.

3.3 Exploiting heterogeneity

With this scenario in mind, we introduce a heterogeneity-aware parallelization of our VS methodology. Our departure point is the parallelization strategy previously presented in Algorithm 2, where independent candidate solutions are run on different processors (in our case GPUs that have assorted CUDA Compute Capabilities). Parallel runs do not incur any communication overhead, and the final solution is chosen from all independent executions, given the stochastic nature of metaheuristics. 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. Performance differences shown in the last two rows of Table 1 lead us to believe that there is ample room for improvement.

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, at run-time, the execution time of a small number of iterations of the metaheuristic (five to ten) in order to detect these differences. Importantly, at this stage, the algorithm is not trying to *solve* the docking problem in any meaningful sense (five to ten iterations is not enough to do this), 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

$$Percent = \frac{Ex.time_{actualGPU}}{Ex.time_{slowestGPU}} \quad (1)$$

The slowest GPU will have $Percent = 1$; a GPU two times faster than 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.

4. EXPERIMENTAL SETUP

Experiments have been conducted in two different heterogeneous systems based on multicore+multiGPU configurations. Below we show the main characteristics of these computational systems along with the particular metaheuristics we have used in our docking approach and a description of the target datasets.

4.1 Hardware environment

Tables 2 and 3 show the characteristics of the two computational systems in which experiments were conducted.

- **Jupiter**: is a system with two hexa-cores (12 cores) Intel Xeon E5-2620 at 2 GHz and 32 GB of RAM. The compiler used is gcc version 4.6.3 with the -O3 flag. The node has six GPUs. Two GPUs are NVIDIA Fermi Tesla C2075 with 5375 MBytes of Global Memory and 448 cores (14 Streaming Multiprocessors y 32 Streaming Processors), and four are NVIDIA GPUs GeForce GTX 590 with 1536 MBytes of Global Memory and 512 CUDA cores. Table 2 gives a full description of Jupiter.
- **Hertz**: has four Intel Xeon E3-1220 processors running at 2 GHz and plugged into a dual-channel motherboard endowed with 8 Gigabytes of DDR3 memory. The node includes two GPUs. The faster is an NVIDIA Tesla Kepler K40c with 2880 cores (15 Streaming Multiprocessors and 192 Streaming Processors) running at boost clock of 0.88 GHz, giving a raw processing power of up to 5068 GFLOPS. The memory speed is 3 GHz with a 384-bit memory bus width that provides a bandwidth of 288 GB/sec. The memory size is 12 GB of GDDR5 with ECC capabilities. The slower GPU is a GeForce GTX 580 with 1536 MBytes of Global Memory and 512 CUDA cores. Table 3 gives a full description of Hertz. We use gcc 4.8.2 with the -O3 flag to compile on the CPU, and the CUDA compiler/driver/runtime version 6.5 to compile and run on the GPU.

4.2 Benchmarking

4.2.1 Metaheuristics

The metaheuristic template shown in Algorithm 1 allows experimentation with several basic metaheuristics and combinations/hybridations of them. In order to have a reduced experimentation space and because we are not interested here in the comparison of different metaheuristics but in the exploitation of the parallelism when applying the metaheuristics to Virtual Screening, we consider four metaheuristics of different characteristics for comparison purposes. The performance and efficiency are evaluated for each, so that conclusions on the whole multicore+multiGPU schema can be drawn.

Table 4 summarizes some parameters for the four metaheuristics considered in our experimental section. The first metaheuristic (M1) is a *Genetic Algorithm* with a population of 64 individuals for each spot in the receptor. Elements are selected for combination from the best ones, 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 closer to that of a *Scatter Search* algorithm. It works with a reference set of the same size as M1, and all the 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. The third metaheuristic (M3) is similar to the second, but with a less intensive local search process. This allows us to analyze the influence of the improvement in the overall performance of the execution. The last metaheuristic (M4) is a neighborhood metaheuristic in which local searches are conducted in the candidate solutions for a large initial set. The search is conducted at each spot by changing the position and orientation of the conformations. The population based metaheuristics (M1, M2 and M3) select the best configurations from those in the reference set and those generated by combination and improvement to enter the reference set for the following computation step. M4 applies only one step, and so there is no selection of elements after improving.

4.2.2 Datasets

We test our designs using a set of benchmark instances from the well-known Protein Data Bank [26]. Surface screening was performed over the proteins 2BSM and 2BXG. The two crystal structures of HSA (Human Serum Albumin) were taken from the RCSB Protein Data Bank as the docking template structures. Table 5 shows the size of each compound.

5. EXPERIMENTAL RESULTS

Given that our techniques establish the experimental setup dynamically, results shown below are platform dependent. Therefore, we provide an exhaustive analysis on the two heterogeneous systems previously described. Tables 6 and 7 show the execution time and relative speed-up factor among different implementations and metaheuristic configurations for each target dataset on Jupiter (PDB:2BSM, Table 6, and PDB:2BXG, Table 7). They show the execution times for OpenMP implementations as a starting point for our improvements. Moreover, they also show the execution times for the different configurations on an homogeneous system, which means only the 4 Geforce GTX 590 are used. This homogeneous execution reports a factor of up to 92x speed-up, which situates this problem as being very interesting for execution on multiGPU systems. Heterogeneous systems are targeted here with the inclusion of 2 Tesla C2075. The heterogeneous system is analyzed with a homogeneous algorithm (considering that all the GPUs have the same computational capability) and with a heterogeneous algorithm which balances the execution at runtime. Although GTX590 and Tesla C2075 are different GPU cards, their computational capabilities are pretty much the same. Actually, both are based on the same architecture, code-named Fermi, but with different number of streaming multiprocessors. Therefore, we report here minimal differences between the homogeneous and heterogeneous versions (up to 6% gains). Finally, comparing Tables 6 and 7 we see that the speed-up increases with the problem size, and so the multiGPU versions prove to be scalable. PDB:2BXG dataset is almost 2.7 times larger than PDB:2BSM and the maximum speed-up factors reported for both cases are 63.82x and 92.26x.

Table 2: Hardware resources and experimental setup on Jupiter.

	Vendor and type	Intel CPU	Nvidia GPUs	
	Family	E5 Family	Fermi	Fermi
	Class	Xeon	Tesla	GeForce
	Model	E5-2620	C2075	GTX 590
	Year	2012	2012	2011
Processing elements	Cores per multiprocessor	(does not apply)	32	32
	Number of multiprocessors		14	16
	Total number of cores		448	512
	Clock frequency (MHz)		1147	1215
Maximum number of GPU threads	Per multiprocessor	(does not apply)	1536	1536
	Per block		1024	1024
	Per warp		32	32
Register file	32-bit registers (per multiprocessor)		32768	32768
SRAM memory (per multiproc. on GPUs)	Shared (only GPUs)	(32 KB L1D and 32 KB L1I)	16 or 48 KB	16 or 48 KB
	L1 cache		48 or 16 KB	48 or 16 KB
	(Shared + L1)		64 KB	64 KB
L2 cache	(shared by all cores)	256 KB	768 KB	768 KB
L3 cache		15 MB	(does not apply)	
DRAM memory	Size (Megabytes)	32143	5375	1536
	Speed (MHz)	2x666	2x1566	2x1707
	Width (bits)	256	384	384
	Bandwidth (Gigabytes/sc.)	42.66	144	163.85
	Technology	DDR3	GDDR5	GDDR5
CUDA Compute Capabilities		(d.n.a.)	2.0	2.0

Table 3: Hardware resources and experimental setup on Hertz.

	Vendor and type	Intel CPU	Nvidia GPUs	
	Family	E3 Family	Kepler	Fermi
	Class	Xeon	Tesla	GeForce
	Model	E3-1220	K40c	GTX 580
	Year	2011	2014	2011
Processing elements	Cores per multiprocessor	(does not apply)	192	32
	Number of multiprocessors		15	16
	Total number of cores		2880	512
	Clock frequency (MHz)		745	1544
Maximum number of GPU threads	Per multiprocessor	(does not apply)	2048	1536
	Per block		1024	1024
	Per warp		32	32
Register file	32-bit registers (per multiprocessor)		65536	32768
SRAM memory (per multiproc. on GPUs)	Shared (only GPUs)	(32 KB L1D and 32 KB L1I)	16 or 48 KB	16 or 48 KB
	L1 cache		48 or 16 KB	48 or 16 KB
	(Shared + L1)		64 KB	64 KB
L2 cache	(shared by all cores)	256 KB	1536 KB	768 KB
L3 cache		8 MB	(does not apply)	
DRAM memory	Size (Megabytes)	7964	11520	1536
	Speed (MHz)	2x666	2x3004	2x2004
	Width (bits)	256	384	384
	Bandwidth (Gigabytes/sc.)	21	288.38	192.4
	Technology	DDR3	GDDR5	GDDR5
CUDA Compute Capabilities		(d.n.a.)	2.0	2.0

Table 4: Algorithm parameters for the four metaheuristics.

Metaheuristic	Initial population (S)	% of elements to be selected for $Ssel$	% of elements to be improved
M1	64*spots	100%	0%
M2	64*spots	100%	100%
M3	64*spots	100%	20%
M4	1024*spots	does not apply	100%

Table 5: Number of atoms of the benchmark compounds from Protein Data Bank site.

Compounds	Atoms
2BSM Receptor	3264
2BSM Ligand	45
2BXG Receptor	8609
2BXG Ligand	32

Tables 8 and 9 show the execution time and relative speed-up factor for the different implementations and metaheuristic configurations considered, for each target dataset on Hertz (PDB:2BSM, Table 8, and PDB:2BXG, Table 9). The GPU heterogeneity in this system is higher than in the previous one. There are two GPUs with different architectures (Fermi and Kepler) and with different CUDA compute capabilities (2.0 and 3.5). Thus, our heterogeneous algorithm offers better results in general, reaching up to 1.56x speed-up factor compared to a homogeneous approach. The overall results also improve, and the speed-up factors reported here with two GPUs are equivalent to those reported with 6 GPUs in Jupiter.

The performance advantage of using GPUs for the docking problem is proven in all cases experimented. This advantage is bigger the larger the number of atoms in the receptor protein. Our CUDA implementations take advantage of data-locality through tilling implementation via shared memory, which benefits the receptor scalability.

Finally, we report higher speed-up ratios whenever we increase either the level of intensification in 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 two 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.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we present a parallelization strategy of a Virtual Screening method tailored for heterogeneous and massively parallel systems. Virtual screening methods are computational techniques that aid the drug discovery process but they are very computational demanding applications. Heterogeneity may limit acceleration and waste energy unless programmers develop smarter applications to control those features wisely on the road towards an optimal performance. In a multicore+multiGPU environment, the capacity of the different computational components in the system is exploited 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. Furthermore, the heterogeneity of the system is exploited at two levels: CPU-GPU heterogeneity and heterogeneity due to GPUs with different characteristics, including different architectures, number of cores and compute capability.

A metaheuristic-based solution for virtual screening is used as the case study. In that way, metaheuristics of different types (distributed and neighborhood metaheuristics) are applied to the solution of this novel problem where a scoring function is optimized. The efficient exploitation of the heterogeneous system gives as a result high speed-ups, which are more important for larger problems. Our strategy is particularly useful for non-deterministic algorithms and stochastic behaviors where real-time constraints must be fulfilled.

For future work and in order to deal with larger problems or for better solutions with limited execution times, it could be convenient to adapt our virtual screening method to more complex 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. Moreover, virtual screening is still at a relatively early stage, and we acknowledge that we have tested a relatively simple variant of the algorithm. But, with many other types of scoring functions still to be explored, this field seems to offer a promising and potentially fruitful area of research.

7. ACKNOWLEDGMENTS

This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grants 15290/PI/2010 and 18946/JLI/13, and by the Spanish MEC and European Commission FEDER under grants with references TEC2012-37945-C02-02, TIN2012-31345 and TIN2012-38341-C04-03, and the Nils Coordinated Mobility under grant 012-ABEL-CM-2014A, in part financed by the European Regional Development Fund (ERDF). We also thank NVIDIA for hardware donation under GPU Educational Center 2014-2014 and Research Center 2015-2016.

8. REFERENCES

[1] Top 500 supercomputer site, note = http://www.top500.org/, year = [last accesed 20 October 2015].

[2] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, New York, 2005.

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

Table 6: Execution time (in seconds) obtained with the application to protein PDB:2BSM in Jupiter of the combinations described in section 4.2. Homogeneous System with 4 GeForce GTX 590 and Heterogeneous System with 4 GeForce GTX 590 + 2 Tesla C2075.

PDB:2BSM						
Metaheurisctic	OpenMP	Homogeneus System	Heterogeneus System		SPEED-UP Heterogeneus Computation Vs Homogeneus Computation	SPEED-UP OpenMP Vs Heterogeneus Computation
			Homogeneus Computation	Heterogeneus Computation		
M1	269.45	7.01	5.13	4.98	1.03	52.53
M2	436.36	10.68	7.92	7.68	1.03	55.09
M3	136.71	3.69	2.71	2.54	1.06	50.44
M4	13557.29	298.27	212.42	211.07	1.01	63.82

Table 7: Execution time (in seconds) obtained with the application to protein PDB:2BXG in Jupiter of the combinations described in section 4.2. Homogeneous System with 4 GeForce GTX 590 and Heterogeneous System with 4 GeForce GTX 590 + 2 Tesla C2075.

PDB:2BXG						
Metaheurisctic	OpenMP	Homogeneus System	Heterogeneus System		SPEED-UP Heterogeneus Computation Vs Homogeneus Computation	SPEED-UP OpenMP Vs Heterogeneus Computation
			Homogeneus Computation	Heterogeneus Computation		
M1	1402.63	23.45	16.96	16.77	1.01	82.70
M2	2272.71	35.37	26.57	25.43	1.04	85.53
M3	711.01	11.81	8.72	8.46	1.03	81.53
M4	70505.22	1113.91	764.131	757.32	1.01	92.26

Table 8: Execution time (in seconds) obtained with the application to protein PDB:2BSM in Hertz of the combinations described in section 4.2. Heterogeneous System with 1 Tesla K40c + 1 GeForce GTX 580.

PDB:2BSM					
Metaheurisctic	OpenMP	Heterogeneus System		SPEED-UP Heterogeneus Computation Vs Homogeneus Computation	SPEED-UP OpenMP Vs Heterogeneus Computation
		Homogeneus Computation	Heterogeneus Computation		
M1	580.23	10.57	6.74	1.56	85.97
M2	937.45	16.47	12.37	1.33	75.76
M3	294.21	5.41	4.09	1.31	71.79
M4	29144.06	470.51	334.41	1.40	87.15

Table 9: Execution time (in seconds) obtained with the application to protein PDB:2BXG in Hertz of the combinations described in section 4.2. Heterogeneous System with 1 Tesla K40c + 1 GeForce GTX 580.

PDB:2BXG					
Metaheurisctic	OpenMP	Heterogeneus System		SPEED-UP Heterogeneus Computation Vs Homogeneus Computation	SPEED-UP OpenMP Vs Heterogeneus Computation
		Homogeneus Computation	Heterogeneus Computation		
M1	2327.60	33.92	22.82	1.48	101.96
M2	3908.46	55.56	41.58	1.33	93.98
M3	1336.40	18.13	13.64	1.32	97.96
M4	150958.75	1735.73	1253.64	1.38	120.41

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

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

[6] J. Carretero, J. García-Blas, D. E. Singh, F. Isaila, T. Fahringer, R. Prodan, G. Bosilca, A. Lastovetsky, C. Symeonidou, H. Pérez-Sánchez, et al. Optimizations to enhance sustainability of MPI applications. In *Proceedings of the 21st European MPI Users' Group Meeting*, page 145. ACM, 2014.

[7] G. De Michell and R. K. Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, 1997.

[8] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization*. Springer, 2005.

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

[10] T. J. A. Ewing, S. Makino, A. G. Skillman, and I. D. Kuntz. DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer-Aided Molecular Design*, 15(5):411–428, 2001.

[11] A. A. Franco. Multiscale modelling and numerical simulation of rechargeable lithium ion batteries: concepts, methods and challenges. *RSC Advances*, 2013.

[12] R. A. Friesner, J. L. Banks, R. B. Murphy, T. A. Halgren, J. J. Klicic, D. T. Mainz, M. P. Repasky, E. H. Knoll, M. Shelley, J. K. Perry, and et al. Glide: A New Approach For Rapid, Accurate Docking and Scoring: Method and Assessment of Docking Accuracy. *Journal of Medicinal Chemistry*, 47(7):1739–1749, 2004.

[13] F. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*. Kluwer, 2003.

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

[15] J. Hromkovič. *Algorithmics for Hard Problems*. Springer, second edition, 2003.

[16] S.-Y. Huang and X. Zou. Advances and Challenges in Protein-Ligand Docking. *International journal of molecular sciences*, 11(8):3016–3034, 2010.

[17] A. N. Jain. Scoring functions for protein-ligand docking. *Current Protein and Peptide Science*, 7(5):407–420, 2006.

[18] W. L. Jorgensen. The Many Roles of Computation in Drug Discovery. *Science*, 303:1813–1818, 2004.

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

[20] S. K. Kuntz, R. C. Murphy, M. T. Niemier, J. Izaguirre, and P. M. Kogge. Petaflop computing for protein folding. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, pages 12–14.

[21] E. López-Camacho, M. J. García-Godoy, J. García-Nieto, A. J. Nebro, and J. F. A. Montes. Solving molecular flexible docking problems with metaheuristics: A comparative study. *Appl. Soft Comput.*, 28:379–393, 2015.

[22] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2002.

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

[24] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 19(14):1639–1662, 1998.

[25] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide 6.5*. 2014.

[26] Protein Data Bank. *Nature New Biol*, 233:223, 1971.

[27] R. Rahman. Xeon Phi System Software. In *Intel® Xeon Phi Coprocessor Architecture and Tools*, pages 97–112. Springer, 2013.

[28] G. R. Raidl. A unified view on hybrid metaheuristics. In *Hybrid Metaheuristics*, pages 1–12. Springer, 2006.

[29] U. Rester. 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, 2008.

[30] J. M. Rollinger, H. Stuppner, and T. Langer. Virtual screening for the discovery of bioactive natural products. In *Natural Compounds as Drugs Volume I*, pages 211–249. Springer, 2008.

[31] G. Rozenberg, T. Bäck, and J. N. Kok. *Handbook of Natural Computing*. Springer, 2011.

[32] I. Sánchez-Linares, H. Pérez-Sánchez, J. M. Cecilia, and J. M. García. High-throughput parallel blind virtual screening using BINDSURF. *BMC bioinformatics*, 13(Suppl 14):S13, 2012.

[33] G. Schneider. Virtual screening and fast automated docking methods. *Drug Discovery Today*, 7:64–70, Jan. 2002.

[34] R. J. Vaessens, E. H. Aarts, and J. K. Lenstra. A local search template. *Computers & Operations Research*, 25(11):969–979, 1998.

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

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