Parallelization of large-scale Drug-Protein binding experiments

Antonios Makris*, Dimitrios Michail*, Iraklis Varlamis*, Chronis Dimitropoulos*,
Konstantinos Tserpes*, George Tsatsaronis†‡, Joachim Haupt†, Mark Sawyer§

*Department of Informatics and Telematics, Harokopio University of Athens, Greece
{amakris, michail, varlamis, xronis.dm, tserpes}@hua.gr

†Technische Universität Dresden, Germany.
{joachim.haupt}@biotec.tu-dresden.de

‡Elsevier, Amsterdam, Netherlands
g.tsatsaronis@elsevier.com

§EPCC, University of Edinburgh, Scotland
m.sawyer@epcc.ed.ac.uk

***Abstract*—Drug polypharmacology or "drug promiscuity" refers to the ability of a drug to bind multiple proteins. Such studies have huge impact to the pharmaceutical industry, but in the same time require large investments on wet-lab experiments. The respective in-silico experiments have a significantly smaller cost and minimize the expenses for the subsequent lab experiments. However, the process of finding similar protein targets for an existing drug, passes through protein structural similarity and is a highly demanding in computational resources task. In this work, we propose several algorithms that port the protein similarity task to a parallel high-performance computing environment. The differences in size and complexity of the examined protein structures raise several issues in a naive parallelization process that significantly affect the overall time and required memory. We describe several optimizations for better memory and CPU balancing which achieve faster execution times. Experimental results, on a high-performance computing environment with 512 cores and 2048GB of memory, demonstrate the effectiveness of our approach which scales well to large amounts of protein pairs.**

I. Introduction

Drug-protein –or drug-target– *binding* simulations aim at the advancement of medical and biological knowledge at a fast pace and a low cost. The field of developing computation services, which perform a computational simulation of the structures' bindings is gaining in popularity the last years and the ability to give accurate estimations of potential true positive drug-target bindings, has a strong practical flavor since it aids drug development and enables drug re-positioning [1].

The assumption that a single drug is able to interact with multiple targets is the essence of drug repositioning. The knowledge of drug-protein interactions, from both drug and protein promiscuity view point can be exploited in the rational design of promiscuous drugs with selective polypharmacology. An important step of the drug-protein binding process, according to Haupt et al. [2], as depicted in Figure 1, is the pairwise structural comparison of a large set of proteins and their potential drug binding sites. More specifically, the protein comparison process comprises the following steps: a) protein pairs alignment, b) candidate binding sites extraction, and c) pairwise structural comparison of binding sites.

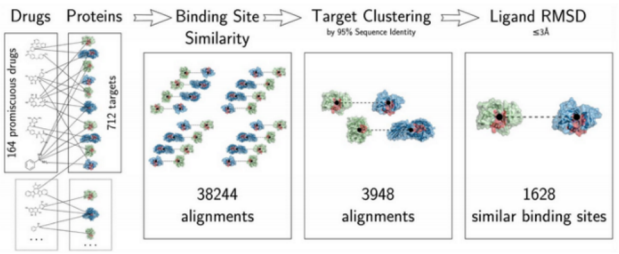


Fig. 1: Protein binding sites comparison pipeline.

In order to take advantage of the resources of a high-performance computing infrastructure and allow the process to scale-up to a large bioinformatics experiment, it is important to analyze the protein comparison process and decompose it to smaller tasks that can be executed in parallel. In this process, possible bottlenecks and the reasons behind them are detected, node synchronization issues are resolved and better memory and CPU balancing is employed.

Several issues that arose during the parallelization of the process have been addressed, such as a) the integration of the different modules in a common parallel processing framework (OpenMPI [3]), b) the efficient management of the available memory, which is shared among the parallel processes and must be carefully allocated in order to prevent the system from

The results presented here were obtained using HPC facilities at EPCC, the supercomputing centre at the University of Edinburgh. The experiments were funded by the Fortissimo project which received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement No 609029.

978-1-5386-3250-5/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCS.2017.39

running out of memory or swapping virtual memory pages to disk, and c) the optimum use of the available processors that minimizes CPU idle time etc.

The major contributions of this work can be summarized as follows:

- The protein similarity measurement process has been ported into a parallel execution environment by integrating different software modules and open source software. and re-engineering the software where necessary.
- An experiment that uses real-world data on a tangible and realistic scenario has been conducted, in order to evaluate the scalability of the solution.
- The parallel process was tested on different parallel configurations achieving a speedup of 210 using 512 cores.

Section II focuses on related works on drug-protein binding and on large scale bioinformatics experiments in parallel architectures. Section III, explains the binding site similarity problem and Section IV presents the software components employed. Section V analyses the shortcomings of a naive parallel solution and Section VI introduces the proposed parallel algorithms. Section VII, provides experimental results on increasing data loads and Section VIII provides the conclusions of this work.

II. Related Work

Research on new purposes of existing drugs [4] falls in different directions such as search for: i) similarity of side effects, ii) similarity of gene expression, and iii) structural similarity of drug binding sites [5]. In drug-protein binding, the drug molecule that produces a signal by binding to a site on the target protein is called "ligand". A drug has several ligands which potentially bind to target proteins, in the so called binding sites which are distinguished into clefts, pockets, or cavities. The rate of this binding is called "ligand affinity".

The current work examines the drug-protein binding problem from the perspective of the structural similarity of binding sites across proteins. Algorithms in this field of bioinformatics, are based on geometrical comparisons between protein structures [2], [6], [7] and use protein structural information from protein databases (e.g. in the Protein Data Bank [8] - PDB). In this work, we port the processing pipeline of [2] to a parallel architecture.

The original pipeline comprises the following steps: a) pairwise protein alignment, b) extraction of candidate binding sites, c) pairwise comparison of all binding sites. Structural based alignment of binding sites can be achieved by iterative search for the best translation/rotation, geometric matching, alignment-free binding site comparison, etc.

With the increasing amount of structural data, an algorithm that can identify the ligand binding sites of proteins on a proteome-wide scale, can be really significant for in-silico drug re-positioning experiments. In [9], the authors propose a new shape descriptor that requires only the Ca atoms to represent the protein structure, which proved fast, scalable to large data set of proteins, and flexible. The approach has been implemented in the protein alignment software SMAP[1] which can be used to align two proteins (query and target one), which is also used for structural comparison of binding sites. SMAP alternatives comprise among others: ProBiS [6], which solves the problem by making use of a maximum clique algorithm, and DaliLite [10] which computes optimal and suboptimal structural alignments, by optimizing a scoring function.

Several large scale bioinformatics experiments exist in the literature and parallel algorithms have been proposed that take advantage of high-performance computing facilities or cloud services (e.g. AWS) and distributed and parallel processing tools such as Apache Hadoop or Spark. Implementations currently target the sequence alignment problem for small [11] or larger sequences [12], phylogenetic analysis [13] and analysis of large scale transcriptomic data [14]. The implemented solutions report parallel efficiencies which reach 0.5 for architectures of 64 cores, whereas the performance for larger architectures is significantly worse. The main reason for this is the increased synchronization and data communication overhead when the number of processing cores increases.

III. Binding-site similarity

When a drug binds to a protein, the drug's ligands form a chemical bond with the binding site of the protein, which can be a surface pocket or an occluded cavity of a certain motif. Medium-sized globular proteins typically have 10-20 binding sites. Ligand-binding sites vary widely in size, most within the range $10^2 - 10^3$Å [15]. The binding sites of protein structures are closely related to protein function, therefore, their identification is essential to the understanding of their interactions with ligands and other proteins. Even in the absence of obvious sequence similarity, structural similarity between two protein structures can imply common ancestry and a similar function. If a protein has a known 3D structure but no known function, inferences concerning function can be made by comparison to other proteins.

The *binding-site similarity* approach, maps the problem of drug-protein binding to a protein-to-protein similarity problem and more specifically to a similarity problem between binding sites in different proteins. Subsequently, the heart of any computation necessary for studying drug-protein bindings and performing drug re-purposing in-silico experiments is a structural similarity computation between protein structures. Almost all computational methods of protein comparison use a simplified representation of proteins, which are partitioned into small patches corresponding to cavities and pockets and then treated as geometric patterns or numerical fingerprints. In this similarity computation process, first the structures of the two proteins are parsed into meaningful 3D coordinates in order to reduce the complexity of the pairwise comparison, then the two resulting patterns are structurally aligned using the transformation that produces the maximum number of equivalent points, and finally a scoring function quantifies the similarity based on aligned features.

[1] http://compsci.hunter.cuny.edu/~leixie/smap/smap.html

From an algorithmic point of view, binding site similarity for drug-protein binding detection, takes as input a set of protein 3D structures (query proteins) and a set of potentially similar protein 3D structures (target proteins) and measures the local similarity between parts of a query and a target protein. The complexity of this computation is high, since all query proteins are compared against all target proteins, each protein having a few dozens of pockets, which are good receptors (binding sites for drugs' ligands). The algorithm first computes the transformations that are needed to align two proteins and then performs all local comparisons among their binding sites.

Each individual local comparison, usually involves modeling the three dimensional structure of atoms as graphs. Subsequent steps of the algorithm try to identify whether the query and target graphs are isomorphic or one of the graphs contains a subgraph that is isomorphic to the other. Since the subgraph isomorphism problem is known to be NP-Complete, some form of backtracking algorithm is used in order to find a graph matching [16], [17]. Another popular approach is to solve the *maximum common subgraph* problem, trying to identify a graph which is isomorphic to both a subgraph of the query and the target graph. The technique of reducing the above problems into solving a maximum-clique or clique-search problem into a compatibility or association graph [18] has been used in the literature by many authors.

The drug-protein binding prediction for a given protein structure currently takes on average 3,000 CPUh for finding the alignments between the protein and all other proteins and 20 CPUh for the subsequent evaluation of binding site similarity. Due to the size of the Protein Data Bank [8], which currently contains 115,000 structures, we are currently facing big challenges in terms of computation time. Similarly, the Binding Database[2] currently contains 1,247,072 binding data (measured binding affinities), for 6,435 protein drug-targets and 550,250 small drug-like molecules.

IV. The Sequential Pipeline

The binding-site similarity pipeline for a pair of proteins comprises several different software components, written in different technologies and programming languages. As a first phase, the protein alignment software SMAP [9], [19] is used to align the two proteins (query and target one). SMAP is a well-known and commonly used software, for 3D ligand binding site comparison and similarity searching of a structural proteome. SMAP downloads the 3D protein structures from the PDB [8].

The output of SMAP is a transformation of the query protein to match the target protein and two object files, one for each protein, that contain information about the residues that the proteins comprise, their pockets, cavities and ligands, which are the potential binding sites for a drug. Since only binding sites of identical promiscuous drugs are aligned against each other, we can use the ligand positions in an aligned pair of proteins to judge the alignment. This is done by measuring

[2]https://www.bindingdb.org/

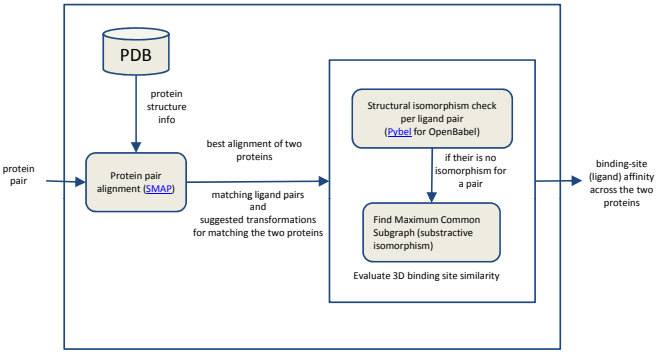


Fig. 2: Initial binding-site similarity pipeline.

the average distance between the atoms of the two ligands, i.e. their root-mean-square deviation (RMSD). Thus, SMAP outputs a list of matching ligand pairs sorted in decreasing RMSD order; two binding sites were considered similar if their alignment yields a RMSD $\leq$ 3Å.

SMAP provides a first estimate of similarity. The next phase of the pipeline is to use additional methods for ligand alignment. The pipeline employs three additional software components, OpenBabel [20], the Small Molecule Subgraph Detector toolkit (SMSD) [21], and RDKit[3].The whole pipeline is integrated using the Python programming language. Regardless the additional toolkit used, a ligand molecule is first read directly from the PDB file into memory and then transformed using the SMAP output (rotation and translation) from the previous phase. Thus, the additional phases of the pipeline are dependent on the SMAP execution phase.

In the final phase, with the use of OpenBabel all possible isomorphisms of the two ligand molecules are computed. If no structural isomorphism is found, the SMSD toolkit is employed in order to perform a substructure search, identifying a maximum common subgraph (MCS)[4]. The longest mapping with the minimum RMSD identified by this procedure, is the final result that is retained. After the isomorphism and/or substructure search alignments are identified, RDKit is employed to align the query and target binding site by performing rotations and translations of maximum atom matches, such that the RMSD is minimized. The final result is the best binding site alignment identified by all the above methods.

V. Parallel Architecture and Challenges

We assume access to a mid-range, industry standard high-performance computing environment, such as INDY[5]. The INDY system consists of several back-end nodes utilizing high performance, low-latency interconnect. Each back-end node has 4 AMD Opteron™, 16 core processors, for a total of

[3]RDKit: Open-Source Cheminformatics Software, http://www.rdkit.org/

[4]The SMSD toolkit combines various algorithms in order to identify the maximum common subgraph, and the decision to use a specific algorithm is based on the complexity of the input molecules. For example molecules, which potentially can be a subgraph based on atoms are handled by the VF+ [16] algorithm first.

[5]https://www.epcc.ed.ac.uk/facilities/demand-computing/indy

64 cores, which share 256GB of memory. For the purposes of this experiment, access to a maximum of 8 such back-end nodes was available, providing a total of 512 cores and 2048GB of memory. INDY is also equipped with a Lustre[6] parallel file system, especially designed to handle highly demanding I/O workloads. INDY utilises IBM's platform HPC cluster management software providing job level dynamic provisioning of compute nodes. Individual back-end nodes are Linux based machines. Regarding available software, the requirements of all components that constitute the pipeline of Figure 2 are the availability of (a) OpenMPI, (b) a Python version 2.7 interpreter, and (c) a Java 5 virtual machine (JVM).

The work has to execute the pipeline of Figure 2 repeatedly for a large number of protein pairs. The main optimization objective is to take full advantage of the allocated resources and minimize the total experiment time. Moreover, the pipeline should be easily executed for different sets of protein pairs without any custom protein-specific optimizations.

A. Baseline

The baseline parallel implementation of the binding-site similarity pipeline, executes the whole pipeline as a black-box taking two proteins as input and finding the best binding-site between them. This includes both major phases of the pipeline described in Section IV. The process is kickstarted by an MPI root process which accepts as input two protein lists (of size n and m respectively) that must be compared, and creates all possible $n \times m$ proteins pairs. The list is subsequently scattered to all p available processes, where each processor executes the pipeline as a black-box for each of the received pairs. Finally, the root process gathers all results into a final output file. The approach is very simple to implement, but contains several drawbacks.

A process that needs to execute the whole pipeline[7] for a single pair of proteins begins with the execution of SMAP, which comprises the following phases: (a) a preprocessing phase that creates and stores an intermediate representation of each protein, and (b) the search for binding-site similarity between the two proteins. Figure 3 contains a more detailed view of the original pipeline which includes critical sub-phases of the various employed components.

B. Parallelization challenges

1) Avoid repetitive protein pre-processing: The parallelization of the preprocessing phase already poses some difficulties. SMAP creates an in-memory representation of the protein called *conformer unit* (CU). Building the conformer unit is a resource intensive procedure, which includes the identification and size characterization of surface pockets and occluded cavities. Locating and measuring the protein pockets and cavities, is based on computational geometry methods,

[6]http://lustre.org

[7]Since both phases of the pipeline require access to the protein structure from the PDB, we assume that all the necessary files (one compressed file per protein) are already downloaded and made available through the Lustre filesystem.

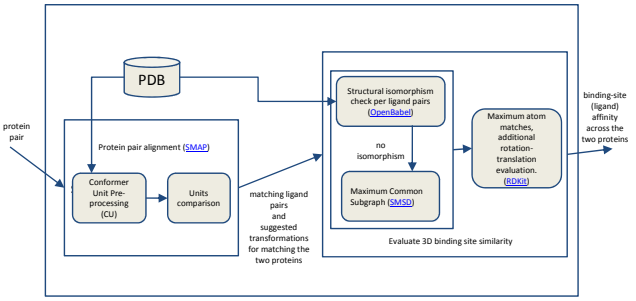


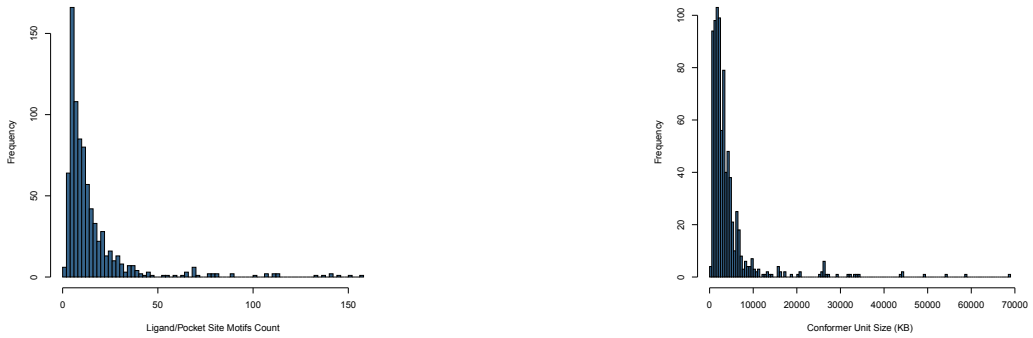
Fig. 3: Detailed view of the processing pipeline.

including alpha shape [22], and discrete flow theory. The identification of the respective SMAP ligand and pocket *motifs* in SMAP is done with the Quickhull algorithm [23]. In order to reduce processing time and avoid rebuilding them for every protein pair comparison, conformer units, once created, are cached to disk. Even in this case, the following scenario may occur, if the allocation of protein pairs to the processes is not explicitly controlled. Two different pairs $\langle p_1, p_2 \rangle$ and $\langle p_1, p_3 \rangle$ can be scheduled in different processors and both get executed simultaneously. Both processes will initiate the preprocessing phase in order to compute the conformer units for protein p_1. Even if we resolve any synchronization issues that may arise (by caching the conformer units in local storage, or using some locking mechanism) when executing the pipeline for a large number of protein pairs, a significant amount of processor cycles and I/O bandwidth is wasted.

2) Balance Non-uniform Proteins Complexity: The structural complexity of different proteins can significantly vary. As depicted in Figure 4, which presents a sample distribution for the set of $\approx$ 800 malaria-related proteins that is used in the experiments, most of the proteins have a handful of motifs, however, there is a significant number of proteins which have 20-50 motifs and there are even a few with more than a 100 motifs (Figure 4a). A similar situation is evident with respect to the size of the conformer units (Figure 4b).

For example, the pair of proteins 4M10 (the structure of Murine Cyclooxygenase-2 Complex with Isoxicam) and 3KQZ (the structure of a protease 2) have 42 and 159 motifs respectively and SMAP produces 3332 ligand pairs as output. Respectively, the pair of proteins 3KWB (CatK covalently bound to a dioxo-triazine inhibitor) and 2AUZ (Cathepsin K complexed with a semicarbazone inhibitor) have 7 and 9 motifs respectively and SMAP returns 8 ligand pairs only.

The ligand alignment phase that follows SMAP execution, is also affected by the non-uniformity of protein complexity. This phase needs to wait for the preprocessing phase to finish for each protein pair and this may differ significantly between pairs. In the previous example, SMAP for the pair 4M10 $\times$ 3KQZ runs for more than 12 hours in order to find the best alignment and generate all the matching ligand pairs and transformations, whereas for the pair 3KWB $\times$ 2AUZ needs only 9 seconds. A similar effect is also observed with respect to memory requirements.



(a) (b)

Fig. 4: Histograms of (a) ligand/pocket size motif count, and (b) conformer unit (CU) size in KBs per protein for a sample of ≈ 800 proteins related to malaria.

The main challenge here is to evenly distribute the protein-pairs on all processors such that the ligand pairs comparisons that follow can start as soon as possible. At the same time it is important to balance the memory requirements across nodes. The worst case scenario is to schedule several memory hungry protein pairs (requiring a few GBs in memory each) to be executed in parallel on the same back-end node. Such a scenario may result in degraded performance due to thrashing.

VI. Improved Parallelization

This section presents various improvements over the baseline implementation that overcome the shortcomings discussed in Section V. In the following, the baseline algorithm which uses scatter-gather to distribute protein pairs and then gather the results, is denoted as algorithm `SG`. During the process of engineering a highly efficient parallel version of the pipeline with a load balancing scheme, we arrive at two other major milestones, which we denote as algorithms `LB-BAR` and `LB-CU-MM`. Since the experiments were performed on a parallel architecture with at most 512 cores, a single process plays the role of the scheduler/load-balancer instead of using a full-blown distributed dynamic load balancing scheme.

Algorithm `LB-BAR` uses a single master process as scheduler/load-balancer to execute the job. The computation is divided into two main parts with a barrier point between them. In the first part the scheduler chooses protein pairs and requests their execution from the worker nodes. The worker executes SMAP for a specified protein pair writing the result on the Lustre filesystem. Each worker is configured to use a local filesystem for caching the generated conformer units to avoid any synchronization issues with other workers which happen to work on pairs which contain the same proteins. Workers notify the scheduler on job completion in order to receive new protein pairs. After SMAP has been executed for all protein pairs, the synchronization barrier is reached by all processes. During the second phase, ligand comparisons are executed for all ligand pairs produced in the first phase. In a similar manner, the scheduler starts scheduling ligand-pairs to workers until all the ligand-pairs have been executed. As ligand-pair comparisons require significantly less time than protein-pair comparison, after the barrier, the scheduler sends batches of work to the workers. The size of these batches is dynamically decreased [24] in a linear way based on the ratio of uncompleted ligand-pairs over available cores.

Algorithm `LB-CU-MM` introduces several additional optimizations. No synchronization barrier is used, which means that the scheduler does some additional bookkeeping, but may schedule a ligand-pair comparison for execution before the execution of SMAP for all protein-pairs is over. This is especially important since a few very slow SMAP protein pairs do not block the execution of the ligand-pairs produced from the other SMAP protein pairs. A second major optimization is the creation of the conformer units (CU) as a separate phase, which is scheduled before the execution of SMAP. The motivation for first computing all conformer units is (a) to avoid performing the same computation multiple times, and (b) to have an estimate of each protein's size and structural complexity. The drawback is that the scheduler needs to keep track of which conformer units have been constructed before it schedules SMAP execution for protein pairs. `LB-CU-MM` starts by scheduling the creation of all conformer units that are part of some protein-pair. Each worker executes the preprocessing phase of SMAP for a protein and stores the resulting conformer unit in the Lustre parallel filesystem. Together with the conformer unit the worker also computes and sends back to the scheduler, the number of protein motifs. After all proteins have been preprocessed, the scheduler starts scheduling protein-pairs to be compared by SMAP. The number of motifs of each protein are used in order to estimate the number of ligand pairs that a pair of proteins is going to produce. As more proteins finish the conformer unit preprocessing, the scheduler starts scheduling protein-pairs to be further compared always giving priority to the protein-pair which has the largest product of motif pairs. Such protein-

pairs are preferred as they are highly likely to require a large amount of time and furthermore produce a large number of ligand pairs which need further processing. After all protein-pairs finish execution, the scheduler starts scheduling ligand-pairs. All three scheduling phases may overlap as no barriers are used.

Even with the above optimizations, for some protein-pairs, SMAP may take a long time, even hours. At this point, the only way to further optimize the execution is to split the SMAP task into smaller tasks, which will be further interleaved by the scheduler. Algorithm `LB-CU-MM` splits the execution of SMAP into motif-pairs. The scheduler splits protein-pairs which contain a large number of motif-pairs into several smaller batches which are then executed in parallel. This is now possible due to the separate preprocessing phase which stores all conformer units.

A final optimization solves the issue of reaching the maximum memory of a backend-node. In the worst case scenario, a bad allocation of tasks to the workers may lead to memory consumption, which is more than the maximum allowed. Each backend-node has 256GB of memory which means that each core has an average of 4GB of memory. The scheduler keeps track of the available free memory of each backend-node, scheduling jobs to be executed with a maximum amount of memory. As an example a protein-pair may be scheduled with 4GB of memory limit while a ligand-pair with only 2GB. The limit is set at the OS level, meaning that the job simply fails if it reaches this amount of memory. Failed jobs are executed in a second round again, but now the maximum memory per job increases to 32GB and the scheduler may execute at most 8 such parallel tasks per backend-node, trading speed for fault-tolerance.

A rough estimate on the cpu and memory requirements of each subtask can be obtained, after the first phase, by the motif count of each protein. Instead of first scheduling large protein-pairs, one could model the problem as an instance of the resource-constrained project scheduling [25] problem and employ more sophisticated heuristics. Moreover, during the last decades a large body of research has been devoted to the concept of dynamic tasking and work-stealing on shared memory and distributed memory architectures. The work on the runtime system of the Cilk [26] parallel programming language, popularized randomized work-stealing as a distributed dynamic load balancing scheme. More recent works, such as [27] scale work-stealing into very large parallel architectures. Another interesting direction is to assert whether work-stealing would be effective in this case.

VII. Experiments

This section contains a detailed experimental evaluation of the aforementioned versions of the binding-site similarity processing chain. In order to perform a real-case experiment with high requirements, we assembled a list of proteins that are related to drugs that treat malaria. DrugBank was used in order to get all the known drug targets of malaria and their UniProt identifiers. From the UniProt identifiers, which correspond to protein chains, only the distinct PDB IDs were kept. Using this short list as query and the whole PDB as target, a set of proteins was obtained which have highly similar drug docking sites. After a careful examination of DrugBank, the listed compounds for malaria and the associated proteins, the result was a short list of approximately 800 proteins. These proteins must be ideally compared with all the proteins in PDB (>115,000), which results in a 800x115,000 executions of the pipeline.

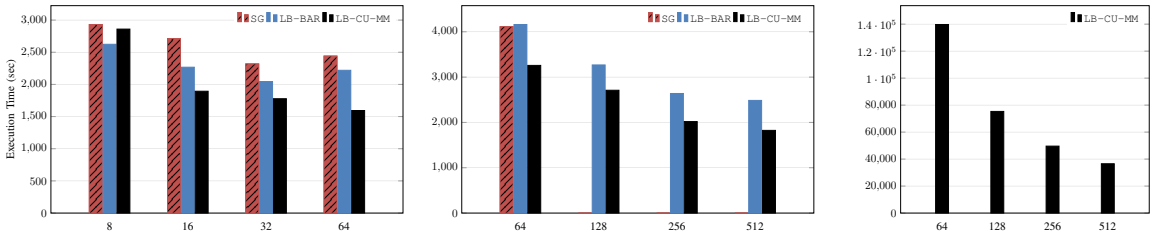
random	motifs	malaria	motifs
4fm5	42	6cox	27
3kwb	7	4ncx	23
		1cjb	25
		1nnu	10
		1d3z	3
		1cmx	7
		1ceq	3
		1c3t	4
		1aar	4
		1a5c	9

TABLE I: The set of proteins used in the small scale experiment.

A. Setup

In order to observe the performance of the processing chains in different scenarios, three different scale experiments were performed. In the *small scale* setup only ten proteins from the malaria list were compared against 2 randomly selected PDB macro-molecules, resulting to 20 unique combinations of protein pairs. Table I lists the 12 PDB identifiers and the number of motifs contained in each, which is an indication of the comparison complexity. The proteins 4fm5 with a large number of motifs and 3kwb with a small number of motifs were chosen. In the *medium scale* setup, the same 10 malaria related proteins were combined with 50 macro-molecules in PDB. Each of the 50 PDB entries is compared with each of the 10 malaria related proteins giving 500 combinations. Two of the 50 proteins have a large number of motifs (4fm5 and 4m10 around 40 each), 5 have a medium number of motifs (from 20 to 30) and the other remaining 43 proteins have a small number of motifs. Our purpose was to compare proteins that either produced a large number either a small number of ligands in order to simulate a real scenario. Finally, in the *large scale* setup the same 10 malaria proteins have been compared with 800 PDB proteins, resulting to 8000 unique combinations of protein pairs.

All experiments were performed on INDY described in Section V. All versions of the parallel pipeline (`SG`, `LB-BAR`, `LB-CU-MM`) are executed for the three different scales. In each case we gather execution times and speedups. All results are averages over five separate runs. For the small scale setup the 20 protein pair comparisons were processed using a single backend-node only, since INDY's backend-nodes have up to 64 cores and 256GB RAM, and evaluated the performance when using 8, 16, 32 and all 64 cores. For the medium and



(a) (b) (c)

Fig. 5: Execution time (in seconds) over number of cores in small (a), medium (b) and large (c) scale experiments.

large scale setup the 500 and 8000 protein pair comparisons respectively were processed using a multi-node setup and the performance was evaluated by using 64, 128, 256 and all 512 cores.

B. Small scale experiment

Figures 5a and 6a present the execution time (in seconds) and speedup of each algorithmic variant per machine cores for the small scale setup. The performance of all three algorithms is similar when using only 8 cores. Increasing them up to 64 produces some variations, however, the experiment is too small in order to fully emphasize each algorithm's advantages. Algorithm `LB-CU-MM` scales better than the remaining ones since it splits large SMAP tasks into many motif-motif tasks which then balance much better into the available cores. `LB-BAR` always behaves better than `SG` which is consistent with our expectations. The reason is that it better balances the second part of the computation which computes ligand-pairs similarities. The running time of `SG` is dominated by the slowest SMAP task which produces a large number of ligand-pairs which are subsequently executed on the same core.

C. Medium scale experiment

In the medium scale experiment, the same 10 malaria related proteins as in the small scale experiment were combined with 50 macro-molecules in PDB giving 500 combinations, which produced 6591 ligand pairs. From the set of these 50 proteins, 2 have a large number of around 40 motifs, 5 have a medium number of motifs (between 20 and 30) and the remaining 43 proteins have a small number of motifs (less than 15). The execution time and speedup can be seen in Figures 5b and 6b respectively. Multiple backend-nodes of INDY were used in this experiment as the number of protein-pairs is larger. Each of the three parallel variants are tested using 64, 128, 256 and 512 cores.

`SG` fails to terminate in a reasonable time even for 128 cores. This is due to the fact that it schedules 128 parallel executions of SMAP. The allocation of these tasks exceeds the 256GB of memory that are available in some backend-node which starts swapping virtual memory pages. This effect, while possible, is not seen in Algorithm `LB-BAR` which successfully finishes the task. The load-balancer/scheduler distributes tasks to workers in a random fashion which in turn distributes the memory requirements between backend nodes. Moreover, the introduction of the barrier point between the two phases, allows the system to settle down between computations.

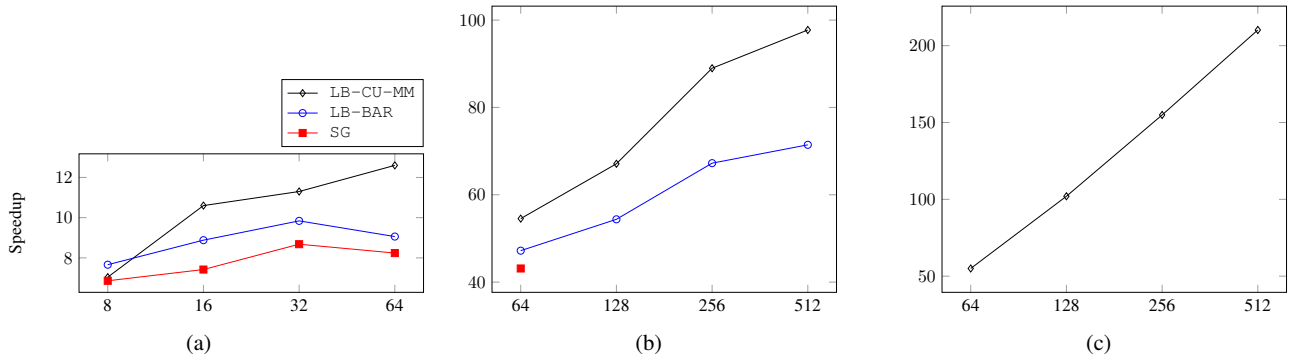
Algorithm `LB-CU-MM` outperforms the other methods and scales better when more cores are available. The decomposition of larger tasks to smaller ones and the prioritization of resource demanding tasks as well as parallel execution of smaller batches of motif-pairs reduces the execution times. In the case of 512 cores, the algorithm achieves a speedup of around 98. However, as can be seen in Figure 6b, the algorithm behaves better when using 64 or 128 cores. The single-process scheduler and its bookkeeping add overhead to the whole process and we assume that larger speedups could be obtained by further optimizing the scheduler operation.

D. Large scale experiment

In the large scale setup the 10 malaria proteins are compared with 800 PDB proteins, resulting in 8000 protein pairs, which in turn generate more than 200,000 ligand pair comparisons. Figures 5c and 6c contain the execution time and speedup. Both algorithms `SG` and `LB-BAR` fail to terminate in a reasonable amount of time. Algorithm `LB-CU-MM` needs 39 hours in order to perform all comparisons when running with 64 cores. When using the maximum of 512 cores it needs 10 hours for the same computations.

For the majority of protein pairs (>80%), the SMAP execution time is less than one minute and more than 96% of the pairs finish in less than 5 minutes. However, there exist pairs that take more than 3 hours. This imbalance further justifies the use of prioritization of SMAP pairs and conformer units extraction method as well as the decomposition of SMAP protein-pairs in smaller batches for parallel execution. The quick execution of ligand matching tasks allows the method to better balance the load among processors. More specifically, 99% of ligand pairs need less than 1 minute to complete and the maximum time for a ligand pair to complete was 7 minutes in our experiments. With the use of `LB-CU-MM` which prioritizes the processing of larger compounds and appropriate balancing of the ligand matching tasks, a 55 speedup was achieved using 64 cores, 102 with 128 cores, 155 with 256 cores, and 210 speedup when using 512 cores.

A critical reason for the effective scaling of `LB-CU-MM` is the additional memory management optimization. In our ex-



(a) (b) (c)

Fig. 6: Speedup over number of cores obtained in small (a), medium (b) and large (c) scale experiments.

periments, several motif-pairs failed to execute when executing with the first memory limit of 4GB, but completed in the second round when running using a 32GB limit.

VIII. Conclusion

This article presented the challenges for the parallelization of a drug-target binding pipeline, which comprises several protein alignment and binding site comparison steps and employs different software modules, written in different frameworks and programming languages. The analysis of challenges and potential bottlenecks allowed us to decompose the sequential execution process and re-assemble everything in a parallel execution process, which achieves better load balancing, memory-handling, and prioritizes time-consuming tasks.

One possible direction for future work is to model the problem as an instance of the resource-constrained project scheduling, using motifs counts as an estimate of subtask requirements in cpu time and memory, and employ more sophisticated heuristics for the initial schedule.

References

[1] A. L. Hopkins, "Network pharmacology: the next paradigm in drug discovery," *Nature chemical biology*, vol. 4, no. 11, pp. 682–690, 2008.

[2] V. J. Haupt, S. Daminelli, and M. Schroeder, "Drug promiscuity in PDB: Protein binding site similarity is key," *PLoS ONE*, vol. 8, no. 6, pp. 1–15, 2013.

[3] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2004, pp. 97–104.

[4] N. Novac, "Challenges and opportunities of drug repositioning," *Trends in pharmacological sciences*, vol. 34, no. 5, pp. 267–272, 2013.

[5] V. J. Haupt and M. Schroeder, "Old friends in new guise: repositioning of known drugs with structural bioinformatics," *Briefings in bioinformatics*, p. bbr011, 2011.

[6] J. Konc and D. Janežič, "Probis algorithm for detection of structurally similar protein binding sites by local structural alignment," *Bioinformatics*, vol. 26, no. 9, pp. 1160–1168, 2010.

[7] G. Liu, M. Liu, D. Chen, L. Chen, J. Zhu, B. Zhou, and J. Gao, "Predicting protein ligand binding sites with structure alignment method on hadoop," *Current Proteomics*, vol. 13, no. 2, pp. 113–121, 2016.

[8] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, 2000.

[9] L. Xie and E. P. Bourne, "A robust and efficient algorithm for the shape description of protein structures and its application in predicting ligand binding sites," *BMC Bioinformatics*, vol. 8, no. Suppl 4, p. S9, 2007.

[10] L. Holm and J. Park, "Dalilite workbench for protein structure comparison," *Bioinformatics*, vol. 16, no. 6, pp. 566–567, 2000.

[11] L. Ping, "Speeding up large-scale next generation sequencing data analysis with pbwa," *Journal of Applied Bioinformatics & Computational Biology*, 2012.

[12] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "BigBWA: approaching the Burrows–Wheeler aligner to big data technologies," *Bioinformatics*, vol. 31, pp. 4003–4005, 2015.

[13] X. Xu, Z. Ji, and Z. Zhang, "Cloudphylo: a fast and scalable tool for phylogeny reconstruction," *Bioinformatics*, p. btw645, 2016.

[14] A. Yang, M. Troup, P. Lin, and J. W. Ho, "Falco: A quick and flexible single-cell rna-seq processing framework on the cloud," *Bioinformatics*, p. btw732, 2016.

[15] J. Liang, C. Woodward, and H. Edelsbrunner, "Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design," *Protein Science*, vol. 7, no. 9, pp. 1884–1897, 1998.

[16] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004.

[17] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976.

[18] J. W. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *Journal of Computer-Aided Molecular Design*, vol. 16, no. 7, pp. 521–533, 2002.

[19] L. Xie, L. Xie, and P. E. Bourne, "A unified statistical model to support local sequence order independent similarity searching for ligand-binding sites and its application to genome-based drug discovery," *Bioinformatics*, vol. 25, no. 12, pp. i305–i312, 2009.

[20] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, "Open babel: An open chemical toolbox," *Journal of Cheminformatics*, vol. 3, no. 1, p. 33, 2011.

[21] S. A. Rahman, M. Bashton, G. L. Holliday, R. Schrader, and J. M. Thornton, "Small Molecule Subgraph Detector (SMSD) toolkit," *Journal of Cheminformatics*, vol. 1, no. 12, 2009.

[22] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Trans. Graph.*, vol. 13, no. 1, pp. 43–72, Jan. 1994.

[23] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[24] T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 1, pp. 87–98, 1993.

[25] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3 – 41, 1999.

[26] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the cilk-5 multithreaded language," *SIGPLAN Not.*, vol. 33, no. 5, pp. 212–223, May 1998.

[27] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, "Scalable work stealing," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 53:1–53:11.