# Parallel large scale inference of protein domain families

Daniel Kahn [2,3,4] Clément Rezvoy [1,3,5] Frédéric Vivien [2,3,5]
[1]École normale supérieure de Lyon [2] INRIA [3] Université de Lyon
[4]LBBE, UMR 5558, UCBL – CNRS, Villeurbanne, FRANCE
[5]LIP, UMR 5668, ENS Lyon – CNRS – INRIA – UCBL, Lyon, FRANCE
Daniel.Kahn@inria.fr, {Clement.Rezvoy,Frederic.Vivien}@ens-lyon.fr

## Abstract

*The resolution of combinatorial assortments of protein sequences into domains is a prerequisite for protein sequence interpretation. However the recognition and clustering of homologous domains from sequence databases typically scales quadratically with respect to their size which grows exponentially, making it essential to parallelize these complex bioinformatics applications. Here we demonstrate the parallelization of* MKDOM2*, the sequential program that has been instrumental in the construction of the* PRODOM *database of protein domain families. This was challenging because of (1) dependencies between program iterations, (2) their extremely heterogeneous run times and (3) communication bottlenecks that could arise because of the large size of the data. A large scale test of the new program,* MPI_MKDOM2*, demonstrated its robustness against heterogeneous run times, preparing the grounds for future releases of* PRODOM *that would otherwise be out of reach with* MKDOM2 *by several orders of magnitude.*

**Keywords:** Bioinformatics, Grid computing, Sequence clustering, Protein domains, Message passing.

## 1 Introduction

Proteins are often composed of independent and evolutionarily conserved units called modules [8] or domains [19], as illustrated in Figure 1. Indeed much of the functional and structural diversity of proteins arises from the combinatorial nature of these domain arrangements. Understanding domain arrangements of proteins is therefore crucial for proper sequence annotation, providing the grounds for rational protein engineering and genome sequence intepretation. Resolving protein domain arrangements is also a prerequisite for protein classification, either for bioinformatics purposes or for evolutionary studies.

Protein domains sharing a common evolutive history can often be recognized by sequence similarity. Several methods have been proposed to identify and classify domains following this principle. Databases such as Pfam [16] or SMART [15] gather domain families compiled by manually driven computational tools. Consequently, domain family definitions found in these databases rest on human expertise and experimental results such as structural information. These databases are therefore regarded as closely matching the “biological truth”. The construction of these databases, however, requires expertise and experimental data which render them unsuitable for the comprehensive processing of the vast amount of protein sequence data available nowadays. Furthermore, the gap between these domain family databases and the raw databases of protein sequences is constantly widening as the size of the latter is increasing at an exponential pace. For instance, the size of the Uniprot [21] database doubles every two years [1]. In addition, genome and metagenome data have shown that the universe of protein families is very far from being completely represented in the current data: their number still increases linearly as a result of increased coverage of genome space [22]. Some bioinformaticians have thus designed systems to fully automatically build domain family databases, such as EVEREST [13], ADDA [11] and PRODOM [17].

The PRODOM database [5, 17] aims at being a comprehensive repository of families of homologous domains built by an automatic process based solely on sequence similarity. PRODOM is built from all known protein sequences available in the Uniprot database [21]. Since release 36 [6], the PRODOM database has been successfully built using the MKDOM2 algorithm [9]. This algorithm, however, is inherently a sequential and iterative algorithm, whose complexity is quadratic in the size of the protein database processed. Therefore, the exponential increase in processing speed over the years is not sufficient to maintain the construction time of PRODOM at a reasonable level. Indeed, while MKDOM2 took 2 months in 2002 to build a new version of PRODOM, it needed more than 15 months in

[1]Uniprot contained in 1.2 million of sequences in March 2004, 2.8 in February 2006, and 5.7 in February 2008 [18]

1521-9097/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPADS.2008.115

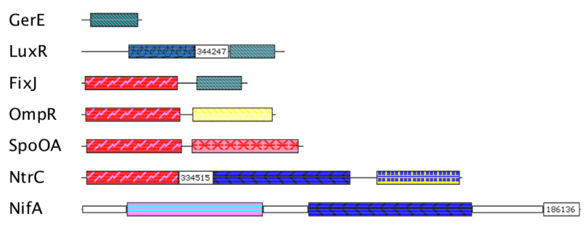


**Figure 1. Example of the decomposition of proteins into domains. Note that different proteins may share homologous domains symbolized here by different cartoons. The functions of proteins (in this case transcriptional activators) depends strongly on their domain arrangements. This example was taken from the ProDom database of protein domain families [5].**

2007. It has thus become impossible to produce new versions of PRODOM using the sequential MKDOM2 program. In principle it could be possible to run incremental updates by clustering homologous domains into pre-existing PRODOM families and running MKDOM2 on the remaining sequences, which would limit the computational cost to $O(n.\Delta n)$ when the relative increment $\Delta n/n$ is small. However such a greedy incremental procedure would forbid the re-examination of previously inferred domain families, even in the light of contradictory new evidence. Moreover, the increase in processor speed nowadays tends to slow down and to be replaced by the multiplication of processor cores. This trend effectively forces the design of parallel versions of historically sequential programs, such as MKDOM2, in order to benefit from the potential gain in computational power.

We have thus designed a new parallel algorithm, MPI_MKDOM2, to build the PRODOM database. Our aim in designing MPI_MKDOM2 was twofold: 1) from the biological point of view we did not want to compromise on the quality of the output, and thus decided to stay as close as possible to the behavior of MKDOM2; 2) from the computer science point of view we wanted an efficient distributed algorithm capable of harnessing the processing power of clusters or even Grids. Satisfying these two objectives appeared to be contradictory because of the assumptions the MKDOM2 algorithm relied upon. The parallelization of MKDOM2 was especially challenging because of 1) the potential inter-dependences between iterations, and 2) the unpredictability and high variability of iteration running times.

In this article, we will start by presenting the MKDOM2 algorithm and the challenges of its parallelization (Section 2). We will present the main characteristics of the proposed solution (Section 3). Finally we will evaluate and discuss the performances of the MPI_MKDOM2 algorithm in terms of speed-up, load balancing and scaling (Section 4).

## 2 The MKDOM2 algorithm

### 2.1 The original algorithm

The MKDOM2 algorithm relies on a simple assumption: the shortest sequence of the dataset —or its repeated unit if it contains an internal repeat— is an atomic domain [9]. In order to avoid biologically meaningless domains this assumption is complemented by the automatic discarding of any sequence shorter than 20 amino-acids. In addition, compositionally biased segments are masked in order to avoid the clustering of similarly biased segments that are usually unrelated evolutionarily [20]. The MKDOM2 algorithm is then defined by the iterative application of its underlying assumption:

- While the sequence database is not empty do
    1. Discard sequences shorter than 20 amino-acids.
    2. Choose as query a shortest sequence (potential ties are randomly broken).
    3. If the query-sequence is longer than 40 amino-acids, check it for internal repeats using an ungapped BLASTP search [3]. If a repeat is found, the repeat becomes the query-sequence.
    4. Using PSI-BLAST [4], search for sequences homologous to the query-sequence.
    5. Define a new family by grouping the query-sequence and the matching segments from the PSI-BLAST search.
    6. Remove all segments belonging to the new family from the database and split sequences accordingly when required.

Figure 2 illustrates this execution under different scenarios depending on the output of Steps 3 and 4. A typical execution of MKDOM2 is made of a huge number of short-time iterations. For instance, in Section 4 we report that MKDOM2 took 17 hours and 20 minutes with 72,413 iterations to process a database containing 73,951 protein sequences. Hence, each iteration took on average 0.86 seconds. Almost all the time in an iteration is spent in Step 4 performing the PSI-BLAST search and in Step 5 updating the database.

### 2.2 Potential parallelizing approaches and challenges

There are two obvious parallelization approaches: we can either parallelize the processing of each iteration or we can process several iterations in parallel. These two approaches have their pros and cons.

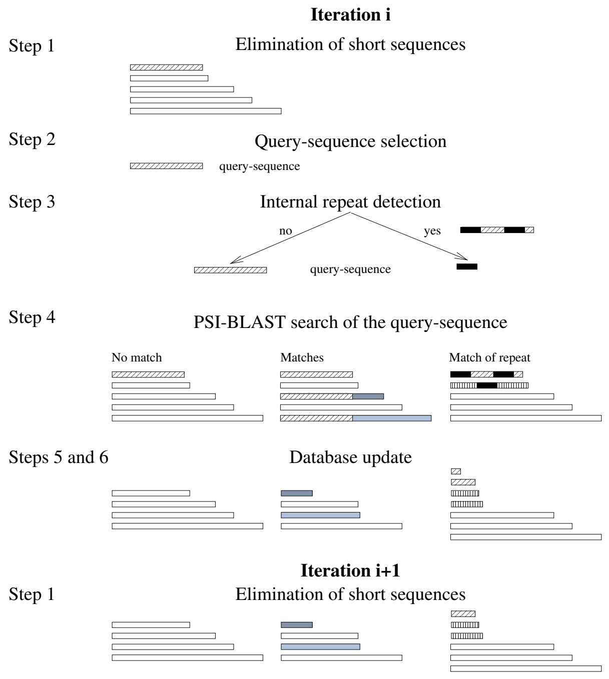
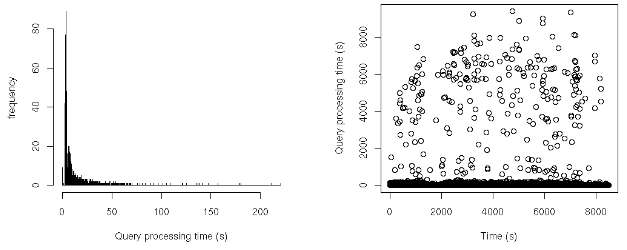


**Figure 2. Outline of sequence processing in the MKDOM2 algorithm [9].**



(a) distribution of processing times (b) extremal values

**Figure 3. Excerpt from query processing times for a database containing all sequences from 263 complete genomes [12].**

### 2.2.1 Parallelizing each iteration

Such an approach would have the great advantage, from the biological point of view, to leave untouched the structure and thus the nature of the heuristics. The parallel and sequential versions of the algorithm would deliver the exact same output. However the PSI-BLAST search of Step 4 is itself an iterative process: it is a succession of up to 10 BLAST searches where, after each search, the results are merged into a Position Specific Scoring Matrix (PSSM) that is used as the query for the next iteration [2]. A distributed implementation of Step 4 would thus require the broadcasting of the new query, and the gathering of the results, for each of the classical BLAST searches of Step 4. This would lead to a strong synchronization of all the processes involved. Such an execution would greatly suffer if the load was not perfectly distributed among the involved processes. Indeed the running time of a BLAST search depends on the number of matches, which cannot be predicted. Therefore we have no means to guarantee a good load balancing: such a tightly coupled fine-grain parallelization is unlikely to deliver impressive speed-ups. It can be noted that the distributed implementation of the BLAST tool suite mpiBLAST [7], does not include blastpgp for similar reasons [2].

[2]The actual number of iterations performed depends on how many results there are and how fast the set of results stabilizes.

### 2.2.2 Executing several iterations in parallel

As this approach would enable us to execute thousands of iterations in parallel, it has the potential of a considerable speed-up. This approach, however, encounters serious problems from the points of view of both computer science and biological results: the potential inter-dependence of iterations, the large variations in the iteration running times, and the modification of the processing order of query-sequences.

**Inter-dependence of iterations.** Let us consider two query-sequences $\mathcal{S}_1$ and $\mathcal{S}_2$ that are processed in parallel. If they belong to the same domain family, they will produce the same result[3] and one of the two executions is thus worthless. Under MKDOM2, $\mathcal{S}_1$ and $\mathcal{S}_2$ would not both have been considered as query-sequences. In such a scenario, processing both $\mathcal{S}_1$ and $\mathcal{S}_2$ is nothing but a waste of resources. Such a scenario is not at all unlikely as slight variations of a sequence are prone to have approximatively the same length and thus are prone to become eligible as query-sequences at around the same time. In fact, this undesirable scenario is thought to be one of the main causes of failure of the previous parallelization attempt [14]. Therefore, we must definitely avoid to run inter-dependent iterations in parallel.

In practice, inter-dependent iterations do not necessarily involve query-sequences belonging to the same domain family. Indeed, two inter-dependent iterations may be defined by query-sequences matching neighbouring domains on a particular sequence $\mathcal{S}$. As domain boundaries are not sharply defined, the regions where the two query-sequences match with $\mathcal{S}$ can slightly overlap. Because of its sequential nature, the MKDOM2 algorithm was implicitly taking care of this peculiarity (yet in an arbitrary way). In a parallel execution we will have to deal with this problem explicitly.

**Variations in query-sequence processing times.** As a test

[3]They will probably not produce the exact same result, but the differences between the two results are assumed to be small and not biologically significant.

of query processing times, we ran MKDOM2 on a large set of protein sequences from the 263 complete genomes compiled in HOGENOM release 03 [12]. For this 340 MB database, processing a query-sequence took on average 33 seconds (the median time being 4 seconds). The distribution of execution times is illustrated on Figure 3(a): it exhibits a steep decay, with a vast majority of query-sequences with short processing times. What this figure does not show, however, is that there are few sequences whose processing times are several orders of magnitude larger than the average processing time. This is exemplified on Figure 3(b): the worst case for this experiment was 9,956 seconds, 299-fold above the average value. We therefore need to design an algorithm that is robust to these large variations in query processing times.

**Impact on the underlying biological hypothesis.** Under the original MKDOM2 algorithm, the processing of a query-sequence can lead to the apparition of a new shortest sequence which is then chosen as the next query-sequence. In a parallel execution, in order to simultaneously process a certain number $n$ of query-sequences, one has first to define these $n$ query-sequences. This a priori forbids that a short sequence produced by the processing of the first of these $n$ query-sequences be processed before these $n$ query-sequences are fully processed [4]. Therefore, a parallel execution will not process dependent query-sequences in the same order than the original MKDOM2 algorithm. This will obviously have an impact on the solution. Ideally, we would like this impact to be small and not biologically significant. We therefore want the parallel execution to stay as close as possible to the sequential one.

**In summary,** the only possible approach for the parallelization of MKDOM2 is to process several query-sequences in parallel. To successfully achieve this goal we have to:

1. be careful not to perform redundant computations;
2. be able to cope with high variations in execution times;
3. stay as close as possible to the original MKDOM2 algorithm to lower the impact of the parallelization on the quality of the output.

## 3 Parallelization approach

### 3.1 Prediction of dependences between query-sequences

In order to avoid running inter-dependent query-sequences in parallel, we rely on an all-against-all BLAST search: initially, we compare all sequences in the database globally to determine whether they share any homology. This may be seen as a very expensive pre-treatment phase, almost as expensive as the whole computation one wants to parallelize. Contrary to the MKDOM2 Algorithm, this all-against-all BLAST search is embarrassingly parallel and thus can be trivially parallelized. Furthermore, this search uses a simple BLAST search which is less time-consuming than a PSI-BLAST search. Therefore, even if this pre-treatment has a large computational cost, it enables us to reach our goal: to very significantly decrease the wall-clock time needed to produce the PRODOM database (including the time needed to perform the pre-treatment). Remember that the unpredicted inter-dependences between query-sequences is considered to be one of the main reasons the previous parallelization attempt failed [14]. Finally, such a search is used for other applications and its cost is therefore shared with other projects such as the construction of protein families in the HOGENOM database [12].

We use the all-against-all result to decide whether any two query-sequences are worth running in parallel. If homology is found between two sequences, they are considered adjacent: neither them nor their subsequences will be simultaneously selected as query-sequences. Since a PSI-BLAST search generally generates broader results than a classical BLAST search, the all-against-all search cannot absolutely guarantee that the results of non-adjacent sequences will not overlap. The all-against-all information, however, ought to be sufficient to ensure that the occurrence of these overlaps is infrequent enough not to have a significant impact on the performance of the parallel execution.

A potential drawback of this approach is that two sequences $\mathcal{S}_1$ and $\mathcal{S}_2$, which are subsequences of homologous sequences, are adjacent even if $\mathcal{S}_1$ and $\mathcal{S}_2$ are not homologous. From the biological point of view, this can impact the order in which query-sequences are processed. From the computer science point of view, this may artificially decrease the number of potential query-sequences towards the end of the iterative process and thus lead to partial starvation (not enough runnable query-sequences to harness the platform resources).

### 3.2 A master-worker approach

The obvious way to parallelize MKDOM2 is to use a master-worker structure. A naive solution would be to follow Algorithm 1. This would lead to very poor performance because:

1. Relative sequentialization of the respective work of the master and of the workers;
2. Variations in query-sequence processing times;
3. Potential processor heterogeneity;
4. Potential communication bottlenecks;
5. High computational burden on the master.

[4]The parallelization does not strictly forbid such a scenario. In fact, it just renders it very undesirable. Indeed, one can always discard the results produced by the last $n-1$ query sequences and run the new shortest one, as under MKDOM2. But a parallelized version that regularly discards the extra computations is doomed to have poor performance.

**Algorithm 1**: Master in a (very) naive master-worker parallelization of MKDOM2.

```
1 while Database is not empty do
2     Select the $n$ shortest and non-homologous sequences
3     Send $\frac{n}{p}$ sequences to each of the $p$ workers
4     Gather the $n$ results
5     Update the database
```

We will now explicit the solutions we propose to overcome these potential problems. But before doing that, we pinpoint a problem from the biological point of view: the order in which results are taken into account.

### 3.2.1 Order of result incorporation

There is no reason for the master to receive the processing results in increasing order of the query-sequences (when we consider the batch of $n$ query-sequences of Step 1). In order not to depart too much from MKDOM2, we decide to enforce that results coming from query-sequences belonging to the same batch are integrated in the database in increasing order of the size of the query-sequences (obviously all results from a batch are integrated before any result from the next batch is taken into account). This may force the master to wait for some results before being able to take into account already received ones. This enforced property is important because results from two query-sequences may overlap even when they are not adjacent in the all-against-all search (see Section 3.1). When two results overlap, the one corresponding to the shortest query-sequence is used to create a domain family and to update the database while the other one is just discarded.

### 3.2.2 Relative sequentialization and variations in processing times

In the scheme of Algorithm 1, first the master prepares the batch of new query-sequences, next these queries are processed, then their results are used to update the database and, finally, a new batch is computed. As we do not want the workers to stay idle waiting for a new batch of query-sequences, we want the master to build and send the new batch of query-sequences before any processor ends its work.

Also, as we have seen, there can be several orders of magnitude between the shortest and longest query-sequence processing times. Under a synchronous execution such as Algorithm 1, each time one of the $n$ query-sequences has a large processing time, all the other processors will be idle most of the time. As we have no way to forecast or prevent this overly lengthy processing, we must be able to supply workers with new query-sequences while waiting for the output of one of the original $n$ queries.

In an attempt to smoothen both the master's workload and the irregularities in query-sequence processing times, the master does not send a single query-sequence by worker but several. We further refine this behaviour to circumvent, with a single mechanism, the two problems we just described. In our actual settings, the master starts computing a new batch of query sequences as soon as it has received the results for at least 50 % of the query-sequences of a given worker. Furthermore, the master sends a new batch of query-sequences to a worker only if it has fewer query-sequences left to process than it received in its last batch.

### 3.2.3 Potential processor heterogeneity

This is simply taken into account by the initial sampling of the platform. We time the processing on all workers of two randomly chosen sequences. The query-sequences are then distributed to workers proportionally to their recorded speeds. This evaluation of processing speed is not perfect. However, discrepancies in the recorded speeds cannot have a cumulative impact because of the mechanism put in place in Section 3.2.2 to cope with the potential lateness of workers.

### 3.2.4 Master load and communication bottlenecks

The load of the master is hard to predict. It depends on the size of the sequence database, the ease in identifying independent query-sequences, the frequency of query-sequences with a large processing time, etc. A worst case may be right after the master finally received the output of a query-sequence with large processing time, resulting in a long backload of results to be incorporated. In order for the master not to be a bottleneck, we have to decide how to propagate the database updates. Indeed the master may send either the updated database or just updating instructions to all workers. The first approach is communication-intensive and requires more work from the master (the databases used in Section 4 already had a size of 360 MB, and we need to process databases with more than 2 GB). The second approach is computation intensive and requires all workers to perform the exact same computation. Therefore it trades computation replication for communications. The specification of a database update is rather small: just two integers per updated sequence (start and end point of cut subsequence). We chose the second approach because it leads to far less communications and puts less burden on the master.

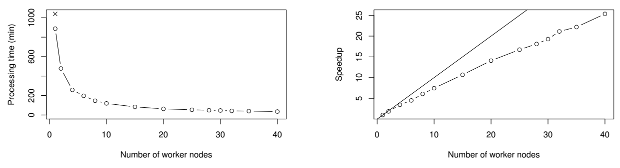
A potential efficiency bottleneck would be the synchronization of master and workers through communications. To avoid such a problem the overall execution was as loosely coupled as possible. In addition to the partial de-synchronization described in Section 3.2.2, we use asynchronous communications and each worker is multi-threaded, one thread dealing with the processing of query-sequences and the other one dealing with all communications (receiving new sequences and sending back results).

### 3.2.5 Other considerations

**Memory requirements.** It is increasingly common to have several processors and/or cores per cluster nodes. Ideally we would start a number of workers per node matching the computing means present. However the available memory per node will be here a limiting factor. Indeed in addition to the high memory cost of PSI-BLAST, we keep the whole database in memory in order to avoid long update phases. The amount of available memory will therefore limit the number of workers per node. Theoretically, it could have been possible to run one thread per core, each processing a different query-sequence while sharing the same memory mapping of the sequence database. However the current implementation of PSI-BLAST does not allow for this. It is also possible to start PSI-BLAST with only one query but several computing threads, which we do, however this option induced only limited increase in efficiency. During the experiments in Section 4, we limited the load to one worker per node even though the nodes had 2 to 4 cores each.

**Input/Output issues.** MKDOM2 in its current implementation relies heavily on disks. Between each iteration the input FASTA file is read and modified to remove all sub-sequences that have been recruited into domain families. Some of these operations are mandatory and required by the fact that BLAST itself is a file-based process. However, some of these operations can be avoided by keeping the dataset in memory in a sorted data structure. Sending several queries at once to the workers also diminishes the number of database updates and helps reduce disk usage. Similarly, grouping queries into batches helps rationalize network usage by sending fewer larger messages across.

**Dispensible queries.** The first iteration of a PSI-BLAST search is a classical BLAST search. If this first iteration does not yield any result, the PSI-BLAST search stops. The adjacency matrix being calculated from a BLAST search, we know that sequences that do not share any similarities with the rest of the database will not return any match but themselves with PSI-BLAST as well. For these queries we know the output of the PSI-BLAST without running the query. However they may still be recruited by other queries. We cannot eliminate them from the start, but when they turn up as query-sequences, we avoid the PSI-BLAST calculation and construct a domain family with a single sequence.



(a) running times (b) speedups

**Figure 4. Processing times and speedups for a medium scale test set encompassing all protein sequences from 32 archaeal genomes (21.5 MB). The running time of MKDOM2 (x) is given for reference.**

## 4 Experimental evaluation

We implemented and MPI_MKDOM2 in C++ using MPICH2 [10] and tested it on clusters of GRID'5000 [1], a nation-wide platform of 5,000 processors dedicated to research in grid computing. We first evaluated performance and speed-ups on a medium scale set comprising all 73,951 protein sequences extracted from 32 archaeal genomes (21.5 MB). We then proceeded with a large scale test consisting of all 950,216 protein sequences in release 03 of the HOGENOM database [12]. This 340 MB dataset covers all proteins from 263 genomes, including a wide range of different organisms.

### 4.1 Medium scale test case

Figure 4 shows running times and speedups achieved by MPI_MKDOM2 for the processing of the medium scale archaeal test set. This experiment was carried out on a cluster comprising homegeneous nodes, each with 2 processors (AMD opteron 2.4 GHz) and 2 GB of main memory. The maximum speedup of 20 was obtained with 30 worker nodes. In this case the speedup is constrained by the fact that the running times of individual queries are short (0.36 seconds in average). The overhead costs inherent to the distributed algorithm, in particular communication costs, become too expensive compared to the cost of the actual computation. Moreover the short run times of atomic processes make load balancing more difficult to achieve between master and worker nodes. The master must validate the results received from the worker nodes and process the database accordingly in order to prepare new independent query batches. The efficiency of the parallelization is bound to decrease severely whenever the cost of this master process exceeds the cost of individual PSI-BLAST runs.

With one worker node, the distributed algorithm performed 14 % faster than the original sequential algorithm.

This is due to the fact that MPI_MKDOM2 is a complete rewrite of MKDOM2 which also incorporates some optimizations over the original code. For instance, the fact that the database is kept sorted in memory at any times allows for faster updates, limiting the number of accesses to the FASTA file on disk.

## 4.2 Large scale processing

Figure 5 shows traces of the processing of the larger 340 MB dataset from 263 genomes in HOGENOM release 03. MPI_MKDOM2 was run on these data for 19 hours on 153 nodes, on a heterogeneous cluster comprising 2 categories of nodes: 118 dual processor, dual core nodes (Intel Xeon 5110, 1.6 GHz) with 2 GB of main memory and 36 dual processor nodes (AMD opteron 2.0 GHz) with 2 GB of RAM as well. During this time, 46 % of the database was processed. With a large database such as this, worst case run times are exacerbated: the maximum individual run time reached 9,965 s (Fig. 5c), an extreme challenge for the parallel process. Although rare, these extreme running times resulted in the build-up of very long queues on the master node: there was an average of 27,000 results awaiting validation at any given time (Fig. 5b). This imposes very strong constraints on the master node and could be expected to severely affect load balancing.

However, contrary to this expectation, MPI_MKDOM2 behaved remarkably well under these conditions. The load balancing between workers correctly played its role since no difference in worker occupancy could be observed between the two kinds of nodes composing the cluster. Moreover, even when more than 40,000 results were in queue, the master continued to swiftly provide independent query sequences to the worker nodes. This resulted in a homogeneous and almost continuous activity of all workers throughout the experiment (Fig. 5f). However some phases could be observed during which the global worker occupancy decreased. These phases seemed to correspond to periods during which a large number of small queries had to be processed, rendering load balancing between master and workers more difficult. On average the parallel process experienced little starvation of worker nodes, with an average raw processor occupancy of 86 %. Increasing the number of concurrent queries also increases the probability that their results will overlap. Indeed the validation step detected 25 % of overlapping results that had to be discarded. Altogether this resulted in 55 % useful computational time during which workers actually performed PSI-BLAST searches that accounted for the final result. This will be improved in the future by cutting the maximal running time of individual PSI-BLAST searches, resulting in shorter queues and fewer result overlaps.

Finally, on this large HOGENOM dataset, MPI_MKDOM2 behaved as expected in terms of sequence processing. Despite the fact that the order in which queries are considered in MPI_MKDOM2 has been loosened comparatively to MKDOM2, the typical query size increased steadily over time, as it would have been expected from a stricter heuristic (Fig. 5d). We conclude that MPI_MKDOM2 constitutes an efficient parallel solution to the automated domain extraction problem, able to tackle large datasets and protein families (Fig. 5e). Therefore this parallelization appears extremely promising for the future maintenance of PRODOM, which would not be feasible otherwise.

**Acknowledgments** The PRODOM project is supported by the FP6 EMBRACE Network of Excellence, the FP7 IMPACT Research Infrastructure Programme and the France-Israel Research Network Program in Bioinformatics. Experiments were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr).

## References

[1] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[2] http://www.mpiblast.org/Docs.FAQ.html#other-blast, 4 June 2007.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–10, 1990.

[4] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–402, 1997.

[5] C. Bru, E. Courcelle, S. Carrere, Y. Beausse, S. Dalmar, and D. Kahn. The ProDom database of protein domain families: more emphasis on 3D. *Nucleic Acids Research*, 33(Database Issue):D212–D215, 2005.

[6] F. Corpet, J. Gouzy, and D. Kahn. Recent improvements of the ProDom database of protein domain families. *Nucleic Acids Res*, 27(1):263–7, 1999.

[7] A.E. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. *Proceedings of Cluster World Conference & Expo*, 2003.

[8] R. F. Doolittle and P. Bork. Evolutionarily mobile modules in proteins. *Sci Am*, 269(4):50–6, 1993.

[9] J. Gouzy, F. Corpet, and D. Kahn. Whole genome protein domain analysis using a new method for domain clustering. *Comput Chem*, 23(3-4):333–40, 1999.

[10] William Gropp. Mpich2: A new start for MPI implementations. In *Euro PVM/MPI*, volume 2474 of *LNCS*, pages 37–42. Springer-Verlag, 2002.

[11] A. Heger and L. Holm. Exhaustive enumeration of protein domain families. *J Mol Biol*, 328(3):749–67, 2003.

[12] Hogenom. `http://pbil.univ-lyon1.fr/databases/hogenom.php`.

[13] E. Portugaly, A. Harel, N. Linial, and M. Linial. EVEREST: automatic identification and classification of protein domains in all protein sequences. *BMC Bioinformatics*, 7:277, 2006.

[14] S. Blanquart. Extraction et Classification Parallèle des Domaines Protéiques. MÃ©moire de m2, Université de Rennes-1, 2004.

[15] J. Schultz, R. R. Copley, T. Doerks, C. P. Ponting, and P. Bork. SMART: a web-based tool for the study of genetically mobile domains. *Nucleic Acids Res*, 28(1):231–4, 2000.

[16] E. L. Sonnhammer, S. R. Eddy, E. Birney, A. Bateman, and R. Durbin. Pfam: multiple sequence alignments and HMM-profiles of protein domains. *Nucleic Acids Res*, 26(1):320–2, 1998.

[17] E. L. Sonnhammer and D. Kahn. Modular arrangement of proteins as inferred from analysis of homology. *Protein Sci*, 3(3):482–92, 1994.

[18] `http://www.expasy.uniprot.org/`.

[19] D. B. Wetlaufer. Nucleation, rapid folding, and globular intrachain regions in proteins. *Proc Natl Acad Sci U S A*, 70(3):697–701, 1973.

[20] J. C. Wootton and S. Federhen. Analysis of compositionally biased regions in sequence databases. *Methods Enzymol*, 266:554–71, 1996.

[21] C. H. Wu, R. Apweiler, A. Bairoch, D. A. Natale, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, R. Mazumder, C. O'Donovan, N. Redaschi, and B. Suzek. The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Res*, 34(Database issue):D187–91, 2006.

[22] S. Yooseph, G. Sutton, D. B. Rusch, A. L. Halpern, S. J. Williamson, K. Remington, J. A. Eisen, K. B. Heidelberg, G. Manning, W. Li, L. Jaroszewski, P. Cieplak, C. S. Miller, H. Li, S. T. Mashiyama, M. P. Joachimiak, C. van Belle, J. M. Chandonia, D. A. Soergel, Y. Zhai, K. Natarajan, S. Lee, B. J. Raphael, V. Bafna, R. Friedman, S. E. Brenner, A. Godzik, D. Eisenberg, J. E. Dixon, S. S. Taylor, R. L. Strausberg, M. Frazier, and J. C. Venter. The Sorcerer II Global Ocean Sampling expedition: expanding the universe of protein families. *PLoS Biol*, 5(3):e16, 2007.

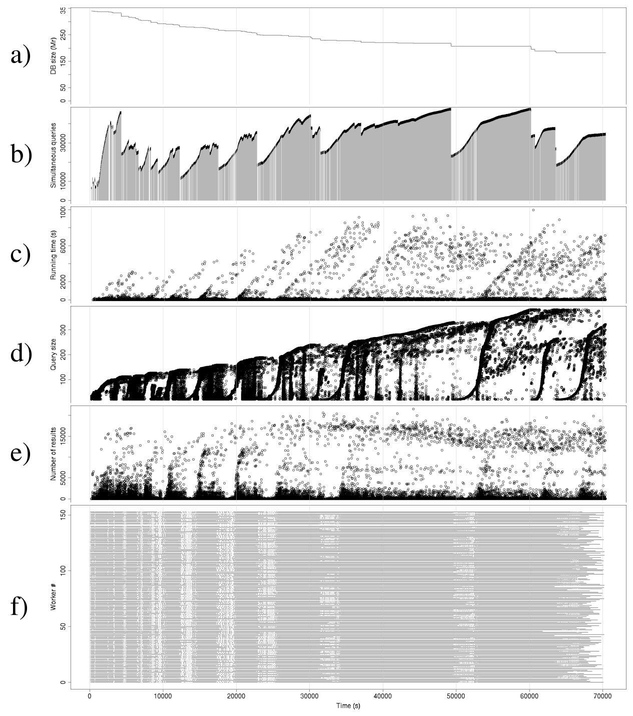


**Figure 5. Large scale MPI_MKDOM2 computation monitored over time. From top to bottom: (a) Size of database in millions of amino-acids. (b) Number of queries considered at a given time; the dark fringe accounts for the queries currently being processed, the light part represents the sequences already processed and awaiting validation. (c) Running times of individual queries. (d) Query sizes in amino-acids. (e) Numbers of matching sequences returned by PSI-BLAST. (f) Worker timeline; each horizontal line represents a worker activity through time. If the line is dark the worker is busy, if the line is light the worker is idle. The queries that the workers were currently computing when they were stopped do not appear in this diagram. The blank line accounts for the master node.**