ProtTest-HPC: Fast Selection of Best-Fit Models of Protein Evolution

Diego Darriba[1,2], Guillermo L. Taboada,[2], Ramón Doallo[2], and David Posada[1]

[1] Bioinformatics and Molecular Evolution Group,
University of Vigo, 36310 Vigo, Spain
[2] Computer Architecture Group,
University of A Coruña, 15071 A Coruña, Spain
{ddarriba,taboada,doallo}@udc.es, dposada@uvigo.es
http://darwin.uvigo.es/software/prottesthpc

Abstract. The use of probabilistic models of amino acid replacement is essential for the study of protein evolution, and programs like ProtTest implement different strategies to identify the best-fit model for the data at hand. For large protein alignments, this task can demand vast computational resources, preventing the justification of the model used in the analysis.

We have implemented a High Performance Computing (HPC) version of ProtTest. ProtTest-HPC can be executed in parallel in HPC environments as: (1) a GUI-based desktop version that uses multi-core processors and (2) a cluster-based version that distributes the computational load among nodes. The use of ProtTest-HPC resulted in significant performance gains, with speedups of up to 50 on a high performance cluster.

1 Introduction

The evolution of protein sequences can be studied using statistical models that describe the probabilities of particular amino acid replacements along specific lineages. Because the number of parameters in these models can be large, in most cases the 20×20 replacement matrices are not estimated *de novo* for each data set. Instead, replacement rates previously estimated from large empirical databases are adopted. Among these, some of the more popular are the Dayhoff [4], JTT [7], mtREV [3], WAG [18], mtArt [1] or LG [10] matrices. Importantly, many phylogenetic calculations like the estimation of tree topologies, branch lengths, nodal support, divergence times or replacement rates benefit from the use of explicit models of evolution. Because, the use of different models can change the outcome of the analysis [15], different model selection tools for protein alignments have been implemented in the past, like ProtTest [2] or ModelGenerator [9]. In addition, some model selection capabilities have been added to more general phylogenetic programs like HYPHY [12], Treefinder [6] or TOPALi [11].

M.R. Guarracino et al. (Eds.): Euro-Par 2010 Workshops, LNCS 6586, pp. 177–184, 2011.

2 ProtTest

The program ProtTest is one of the most popular tools for selecting models of protein evolution, with almost 4,000 registered users. ProtTest is written in Java and uses the program PhyML [5] for the maximum likelihood (ML) estimation of phylogenetic trees and model parameters. The current version of ProtTest (2.4) includes 14 different rate matrices that result in 112 different models when we consider rate variation among sites (+I: invariable sites; +G: gamma-distributed rates) and the observed amino acid frequencies (+F). ProtTest uses the Akaike Information Criterion (AIC) and other information criteria to find which of the candidate models best fits the data at hand. In addition, it can perform multi-model inference and estimate parameter importances [13]. The time required to complete the likelihood calculations, that take most of the runtime of the program, can be variable depending on the size and complexity of the alignments. For large alignments, this task cannot be completed in a reasonable time using a single core. While ModelGenerator/MultiPhyl [8] and TOPALi implement grid computing to speed-up the analyses, they consider fewer models and do not implement model averaging.

3 Java for High Performance Computing

There are several programming options in Java for HPC [16]:

Java Shared Memory Programming. As Java has built-in multithreading support, the use of threads is quite extended due to its portability and high performance, although it is a rather low-level option. Nevertheless, Java now provides concurrency utilities, such as thread pools, tasks, blocking queues, and low-level high-performance primitives (e.g., *CyclicBarrier*), for a higher level programming. However, this option is limited to shared memory machines, which provide less computational power than distributed memory architectures.

Java Distributed Memory Programming. Message-passing is the preferred programming model for distributed memory architectures (e.g., clusters) due to its portability, scalability and usually good performance, although it generally requires significant development efforts. Among currently available Java Message-Passing Java (MPJ) libraries, F-MPJ [17] and MPJ Express [14] deserve to be mentioned for their nested parallelism (MPJ+threads) support for exploiting performance on clusters of multi-core processors.

4 ProtTest-HPC

ProtTest-HPC is a high performance computing application for protein model selection, based on ProtTest, but completely redesigned in order to grant model extensibility, traceability and encapsulation. ProtTest-HPC includes four main hierarchies:

- **Substitution Models** contain the amino-acid model data, although they can be extended to also support nucleotide models.
- **Likelihood Estimators** optimize model parameters as a previous step to model selection. This optimization relies on third-party applications.
- **Execution Strategies** determine how the optimization of the candidate set of models is scheduled (i.e., how the workload is distributed among the available computational resources).
- **Information Criteria** drive the model selection task according to the previous optimization and provide the basis for model-averaging calculations.

4.1 Shared Memory Implementation

ProtTest-HPC uses a thread pool to handle the execution of tasks on shared memory architectures. This implementation is totally portable using thread pools from the Java Concurrence API, which is included in the Java SDK. The task queue contains the whole set of tasks (i.e., candidate models to optimize) which will be processed by the thread pool in a particular order (reverse complexity estimate) (Figure 1).

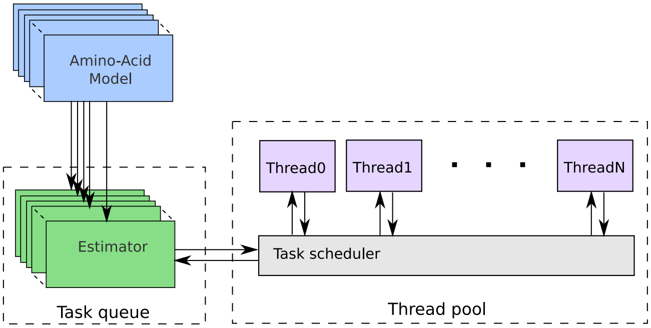

Fig. 1. ProtTest-HPC shared memory strategy

4.2 Distributed Memory Implementation

In order to handle the computation of tasks on distributed memory architectures (e.g., clusters), ProtTest-HPC manages processes, which rely on message-passing communication. ProtTest-HPC uses a distributor process to allocate the workload (Fig. 2) according to three different strategies, one static and two dynamic.

The static approach performs the whole distribution of the tasks before their actual optimization. This distribution is based on the workload estimate for each task, which is key to provide balanced workload assignments. Therefore, message-passing among processes is avoided during computation. As long as the computational load is very hard to estimate, this strategy will usually result in significant runtime differences among processes. The performance of this strategy is highly dependent on the workload estimate and the number of processes

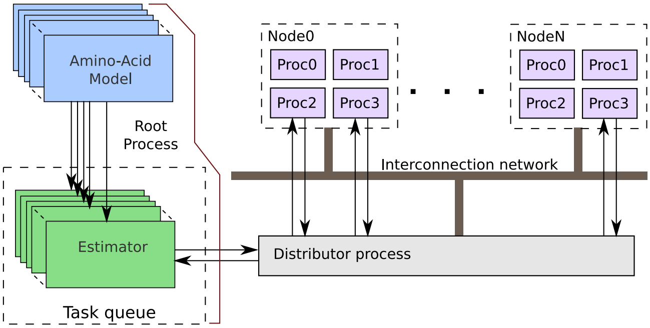

Fig. 2. ProtTest-HPC distributed memory strategy

used (i.e., an inaccurate estimate when scheduling a small number of tasks per process will show poor performance), so it is usually less scalable than dynamic approaches. However, shorter running times will be obtained for small datasets, where the time spent in message-passing becomes more significant.

On the other hand, the behavior of the dynamic approaches is more similar to that of the thread pool, where a task manager distributes the tasks among processes. In a distributed memory implementation, the root process can assume this role, although incurring some overhead. However, the use of an additional dedicated distributor thread can relieve the root process of this work, increasing performance. The computational overhead imposed by this additional thread is almost negligible, as most of the time the thread will be waiting for the processes.

The scalability of ProtTest-HPC using shared or distributed memory was limited by the replacement models with the highest computational load, usually the ”+I+G” models, which could take up to 90% of the overall runtime. In these cases, the runtime was determined by the longest optimization, resulting in poor speedups. Moreover, the higher the number of cores, the higher the workload imbalance due to runtime differences. In fact, it is expected that ProtTest-HPC could take advantage of up to 50 cores, approximately. This important limitation suggests that the combination of the distributed memory version with a parallel maximum-likelihood computation can increase significantly the scalability of ProtTest-HPC. Therefore, this two-level parallelism approach can result in a much more efficient exploitation of the available computational resources.

5 Performance Evaluation

We evaluated the performance of ProtTest-HPC on a representative multi-core cluster under two different scenarios:

- shared memory, using the available cores in a machine.
- distributed memory, running the message-passing version on the whole cluster.

Table 1. Test data sets used for performance evaluation. The base tree used for parameter estimation can be a BIONJ tree fixed across models or the particular ML tree for each model. Execution times are given in minutes.

Data set/ Analysis	Protein	Number Sequences	Length	Base tree	Execution Time
RIB	Ribosomal protein	21	113	Fixed BIONJ	5
RIBML	"	"	"	ML tree	30
COX	Cytochrome C oxidase II	28	113	Fixed BIONJ	10
COXML	"	"	"	ML tree	58
HIV	HIV polymerase	36	1,034	Fixed BIONJ	45
HIVML	"	"	"	ML tree	185
10K	Simulated alignment	50	10,000	Fixed BIONJ	552
20K	"	"	20,000	"	1,470
100K	"	"	100,000	"	4,785

To evaluate the performance of ProtTest-HPC we used 6 real and simulated alignments (Table 1). In all cases the set of candidate models included all 112 models available in ProtTest.

5.1 Shared Memory Benchmarking

Figure 3 and Table 2 show the performance of ProtTest-HPC in an 8-core Harpertown cluster node using shared memory. Here ProtTest-HPC was limited to the use of up to 8 threads (one thread per core). In this scenario, where the number of available threads is significantly lower than the number of models to be optimized, the computational workload was usually well-balanced, and the scalability almost reached the ideal case (i.e., obtaining speedups close to n with n threads). Nevertheless, for the simplest analyses (e.g., COX and RIB) the performance results when using 8 threads were poorer than for more computationally intensive tasks (e.g., COXML and RIBML) as the overhead of threads operation (e.g., synchronizations) and the workload imbalance had a higher impact on the overall performance.

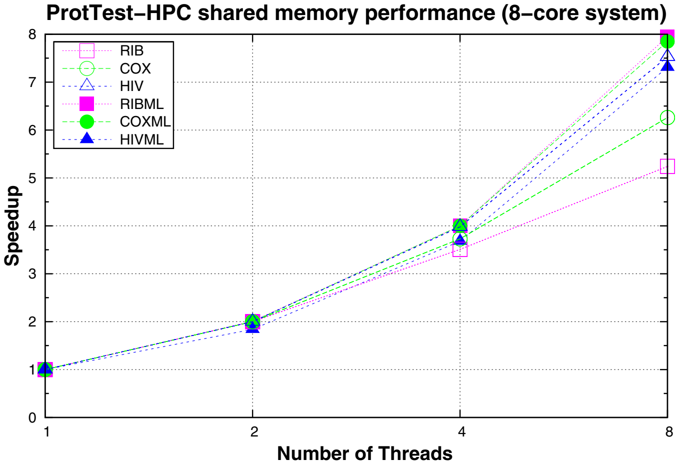

Fig. 3. Scalability of ProtTest-HPC in an 8-core node using shared memory

Table 2. Runtime (seconds) using shared memory on an 8-core Harpertown node

Threads	RIB	COX	HIV	RIBML	COXML	HIVML
1	330	563	2544	1710	3300	9498
2	165	282	1269	851	1647	5149
4	94	151	639	427	825	2581
8	63	90	338	215	415	1300

5.2 Distributed Memory Benchmarking

We explored three different distribution strategies for message-passing. Here we only evaluated the dynamic option because it provided the most balanced workloads without incurring significant penalties (the core devoted to the dedicated distributor thread represents a small percentage of the total number of available cores, 256).

Starting from 16 cores, we ran on the cluster the message-passing parallel implementation of ProtTest-HPC using multiples of 14 cores. The reason for this is that the computational load for a given model depends on the rate heterogeneity and frequency parameters because the replacement matrix is given. Thus, for a specific parameter combination like "+I+G" there are 14 models with similar workload. This suggested the use of a number of cores multiple of 14, so the workload would be more balanced. In this case the number of tasks processed per core is likely to be the same. Additionally, it is expected that models with similar workloads would be optimized by different processes. Finally, ProtTest-HPC currently includes 112 models, and as each model is optimized sequentially by a single core, the maximum number of cores that can be used is 112. Performance in this case was almost linear for the simple analyses up to 28 cores, while in other cases (HIVML) the biggest speedups were obtained with 56 cores (Figure 4 and Table 3). ProtTest-HPC could only take advantage of around 56 cores on a 256-core cluster, as the running times on 56 or 112 cores were similar. This happens because of the coarse-grained paralelism and the differences between the sequential execution times of each substitution model optimization, so this is the main performance bottleneck (the longest model optimization determines the runtime). Moreover, distributing a reduced number of tasks per core severely limits the load balancing benefits, as it is not possible to take advantage of the spare computational power available once a core finishes its task processing.

Table 3. Runtime (seconds) using distributed memory on a Harpertown testbed

Cores	RIBML	COXML	HIVML	10K	20K	100K
1	1710	3300	9498	33160	88129	287134
8	224	429	1880	4421	11750	38284
16	113	251	1172	2963	6590	23417
28	71	152	516	1960	4972	15275
56	64	93	207	1032	3148	7988
112	49	78	206	1028	2593	7178

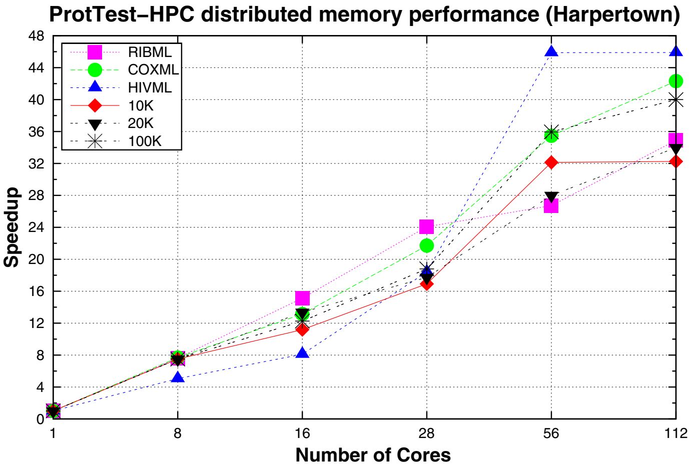

Fig. 4. Scalability of ProtTest-HPC using distributed memory

6 Conclusions

We have developed a high performance computing version of ProtTest for the fast selection of best-fit models of protein evolution. In order to allow for the parallel execution of ProtTest in high performance computing environments, our implementation can work either (i) as a GUI-based desktop version that supports the execution using the available multi-core processors, or (ii) as a cluster-based version that distributes the computational load among the available compute nodes. We show that ProtTest-HPC achieves a significant performance gain over ProtTest, with speedups of up to 50 on an HPC cluster, although the combination of the cluster-based version with a parallel maximum-likelihood computation can increase significantly ProtTest-HPC scalability. For very large alignments, this can be equivalent to a reduction of the running time from more than one day to around half an hour. In this way, statistical model selection for large protein alignments becomes feasible, not only for cluster users, but also for the owners of standard multi-core desktop computers. Moreover, the flexible design of ProtTest-HPC will allow developers to extend future functionalities, whereas third-party projects will be able to easily adapt its capabilities to their requirements.

Acknowledgments. Special thanks to Stephane Guindon for his continuous help with PhyML. This work was financially supported by the European Research Council [ERC-2007-Stg 203161-PHYGENOM to D.P.] the Spanish Ministry of Science and Education [BFU2009-08611 to D.P. and TIN2010-16735 to R.D.] and by the Xunta de Galicia [Bioinformatics (to D.P.) and HPC (to R.D.) Galician Thematic Networks].

References

1. Abascal, F., Posada, D., Zardoya, R.: MtArt: a new model of amino acid replacement for Arthropoda. Mol. Biol. Evol. 24(9), 1–5 (2007)
2. Abascal, F., Zardoya, R., Posada, D.: ProtTest: Selection of best-fit models of protein evolution. Bioinformatics 24(1), 1104–1105 (2007)
3. Adachi, J., Hasegawa, M.: Model of amino acid substitution in proteins encoded by mitochondrial DNA. J. Mol. E 42(4), 459–468 (1996)
4. Dayhoff, M., Schwartz, R., Orcutt, B.: A model for evolutionary change in proteins. Nat'l Biomedical Research Foundation, 345–352 (1978)
5. Guindon, S., Gascuel, O.: A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. Syst. Biol. 52(5), 696–704 (2003)
6. Jobb, G., von Haeseler, A., Strimmer, K.: TREEFINDER: a powerful graphical analysis environment for molecular phylogenetics. BMC Evol. Biol. 4, 18 (2004)
7. Jones, D.T., Taylor, W.R., Thornton, J.M.: The rapid generation of mutation data matrices from protein sequences. Comp. Appl. Biosci. 8(3), 275–282 (1992)
8. Keane, T.M., Naughton, T.J., McInerney, J.O.: MultiPhyl: a high-throughput phylogenomics webserver using distributed computing. Nucleic Acids Res. 35(Web Server issue), W33–W37 (2007)
9. Keane, T., Creevey, C., Pentony, M., Naughton, T., Mclnerney, J.: Assessment of methods for amino acid matrix selection and their use on empirical data shows that ad hoc assumptions for choice of matrix are not justified. BMC Evol. Biol. 6(1), 29 (2006)
10. Le, S.Q., Gascuel, O.: An improved general amino acid replacement matrix. Mol. Biol. Evol. 25(7), 1307–1320 (2008)
11. Milne, I., Lindner, D., Bayer, M., Husmeier, D., McGuire, G., Marshall, D.F., Wright, F.: TOPALi v2: a rich graphical interface for evolutionary analyses of multiple alignments on HPC clusters and multi-core desktops. Bioinformatics 25(1), 126–127 (2009)
12. Pond, S.L.K., Frost, S.D., Muse, S.V.: HyPhy: hypothesis testing using phylogenies. Bioinformatics 21, 676–679 (2005)
13. Posada, D., Buckley, T.R.: Model selection and model averaging in phylogenetics: advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. Syst. Biol. 53(5), 793–808 (2004)
14. Shafi, A., Carpenter, B., Baker, M.: Nested parallelism for multi-core HPC systems using Java. J. Parallel Distr. Com. 69(6), 532–545 (2009)
15. Sullivan, J., Joyce, P.: Model selection in phylogenetics. Annu Rev. Ecol. Evol. S 36, 445–466 (2005)
16. Taboada, G.L., Tourino, J., Doallo, R.: Java for high performance computing: assessment of current research and practice. In: Proc. 7th Intl. Conf. on Principles and Practice of Programming in Java, Calgary, Canada, pp. 30–39 (2009)
17. Taboada, G.L., Tourino, J., Doallo, R.: F-MPJ: scalable Java message-passing communications on parallel systems. J. Supercomput. (2010) (in press)
18. Whelan, S., Goldman, N.: A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. Mol. Biol. Evol. 18(5), 691–699 (2001)