2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

SlimCodeML: An Optimized Version of CodeML for the Branch-Site Model

Hannes Schabauer*, Mario Valle†, Christoph Pacher‡, Heinz Stockinger§,
Alexandros Stamatakis¶, Marc Robinson-Rechavi*, Ziheng Yang‖, and Nicolas Salamin*

*University of Lausanne, Department of Ecology and Evolution and
SIB Swiss Institute of Bioinformatics (CH)
†Swiss National Supercomputing Centre, Scientific Computing Group (CH)
‡AIT Austrian Institute of Technology, Safety & Security Department (AT)
§SIB Swiss Institute of Bioinformatics, Vital-IT Group (CH)
¶Heidelberg Institute for Theoretical Studies, Scientific Computing Group (DE)
‖Department of Biology, University College London (UK)
Email: hannes.schabauer@unil.ch

***Abstract*—CodeML (part of the PAML package) implements a maximum likelihood-based approach to detect positive selection on a specific branch of a given phylogenetic tree. While CodeML is widely used, it is very compute-intensive. We present SlimCodeML, an optimized version of CodeML for the branch-site model. Our performance analysis shows that SlimCodeML substantially outperforms CodeML (up to 9.38 times faster), especially for large-scale genomic analyses.**

***Keywords*-scientific computing; molecular evolution; phylogenetics; codon models; likelihood computation**

I. Introduction

The recent advances in high-throughput sequencing techniques are providing researchers with a wealth of genome scale data that allows us to study evolutionary questions that were not feasible a decade ago [1], [2]. Notably, the increase in molecular data available for non-model organisms is creating a surge toward comparative genomics that is strengthened by recent new theoretical developments [3], [4]. However, the full potential of these new data will only be achieved by the developments of further computational and mathematical techniques.

In particular, the analysis of protein-coding gene evolution has benefited from the development of codon models. They allow a drastic increase of the number of genes identified as evolving under positive selection [5]–[7]. Their use show that similar selection pressures can lead to the repeated emergence of the same phenotype in distant lineages via identical genetic changes [8].

The main computational issue hampering wider use of codon models relates to the need to integrate them within a phylogenetic framework. Transition probabilities between all possible pairs of codons must be computed from the substitution matrix describing the codon model, which is done by computing the matrix exponential for each branch of a phylogenetic tree. Unlike simple nucleotide models that are represented by a 4×4 substitution matrix, a codon model is represented by a 61×61 matrix. Several approximations have been proposed to reduce model complexity [9, p.137], but the high dimensionality of the problem is still challenging the use of these approaches to study genomic data.

Our aim is to reduce the computational time when optimizing the likelihood function used to estimate the parameters of the substitution matrix. This is done through a series of novel optimizations which we implemented in existing software. Our new approach makes it feasible to use these codon models on large-scale genomic data.

A. Motivation and problem statement

An important question in evolutionary biology is to understand how protein-coding genes evolve through time and in the various organisms that exist today. One particular evolutionary force that is of great interest is the so-called positive or Darwinian selection, which

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.88

drives the appearance of functional changes in those protein-coding genes. The importance of detecting positive selection is crucial as this evolutionary force is not just the best explanation for adaptation but the only physically possible purely causal explanation [10, p.25]. Yet because of computational limitations the global incidence of positive selection in genomes remains uncharacterized. The increasing number of available genomes [11] is rendering the problem even more complex, because the addition of new species increases the number of branches and thus the number of computations to make.

The most common method to detect positive selection is to test through likelihood ratio test [12] if a codon model allowing positive selection on a particular branch of a phylogenetic tree (hypothesis H_1) explains the data better than a codon model that does not allow for positive selection (hypothesis H_0). If this test is significant and positive selection is confirmed, Bayesian approaches are used to assess the posterior probability of a particular codon along the protein-coding gene sequence to be evolving under positive selection [13]. This is done iteratively for each branch of a phylogenetic tree [14].

CodeML [15] is the main software to carry out tests of positive selection. It has been designed to test one gene and one branch at a time. It is widely used in the community, and few alternatives exist to detect positive selection. CodeML is also the central component for populating the Selectome database [16], which carries out genome-wide analyses of positive selection. As the size of the data analyzed for the Selectome database poses a major computational challenge [17], [18], there is a keen interest in reducing the computational time. Many other researchers will benefit from a faster version of CodeML.

The focus of CodeML is on richness and flexibility of the implemented codon models. In this article, we focus on optimizations for large-scale computations and for next-generation high performance computing systems of one type of codon models called branch-site model (BSM). Therefore, we present SlimCodeML, a newly developed optimized version of CodeML.

B. Related work

There is a rich choice of software available for phylogenetic reconstruction and molecular evolution analyses based on nucleotide data. Most of the current implementations use Maximum Likelihood (ML) estimation of the model parameters, but there are very few ML-based software packages to estimate parameters of codon models: HyPhy [19], CodeML (PAML) [20], IQPNNI [21], and BEAGLE [22]. However, only CodeML implements the branch-site model, which is of strong interest for current genomic analyses.

In contrast to most phylogenetic software available, we are *not* interested in reconstructing the phylogenetic tree itself (tree topology remains unchanged). The latter one has been shown to lead to a memory bound problem [23] due to the necessity to store different tree topologies in memory. The problem presented here is, however, CPU bound because of the need to compute the transition probabilities of a 61×61 substitution matrix over up to $2s - 3$ branches of a phylogenetic tree with s extant species.

Synopsis

The remainder of this paper is organized as follows. Section II discusses computational methodology and the mathematical background, while Section III discusses the corresponding implementations. Section IV summarizes experimental results for SlimCodeML in terms of accuracy and runtime behavior, and Section V concludes and points out future work.

II. Methodology

The current implementation of CodeML requires as input a multiple sequence alignment (MSA) for a set of species and a phylogenetic tree in Newick format, where the specific branch to test for positive selection is identified (Fig. 1). In addition, a dedicated parameter file is read by CodeML to set model parameters and corresponding optimization options [24].

A. Branch-Site Model

CodeML implements different codon substitution models by means of continuous-time Markov models [25]. The codon substitution process is described by the instantaneous rate matrix $Q = \{q_{ij}\}$, where q_{ij} corresponds to the instantaneous transition rate from codon i to codon j ($i \neq j$). Models implemented in CodeML assume that codon transitions can only occur through the change of a single nucleotide in one of the three positions forming a codon (Fig. 1). Transition rates for multiple nucleotide changes are set to 0. The nucleotide changes within each codon are represented

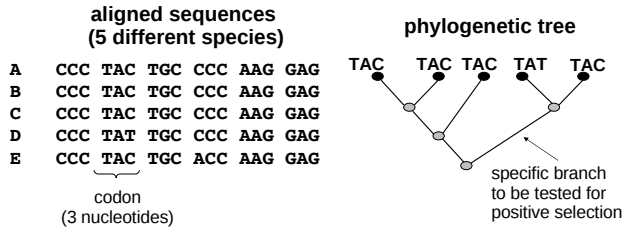

Figure 1. CodeML input. *(left)*: exemplary MSA for 5 different species, each with 6 codons; *(right)*: binary phylogenetic tree with the 5 species indicated by black nodes. Inner nodes in the tree represent extinct ancestors. The last common ancestor of all species is called the root node.

by two parameters. First, the nucleotide change can either be a transition (purine $\rightarrow$ purine or pyrimidine $\rightarrow$ pyrimidine) or a transversion (purine $\rightarrow$ pyrimidine or pyrimidine $\rightarrow$ purine) whose ratio is represented by the parameter κ. Second, the substitution can be synonymous (i.e., no change in the amino acid coded by codon i and j) or non-synonymous (i.e., the coded amino acid changes), which corresponds to the parameter ω (i.e., selective pressure). Finally, the codon frequencies π_i used in the model are determined empirically from the MSA. See Eq. 1 for a summary of the applied codon substitution model [9, p.48].

$$q_{ij} = \begin{cases} 0 & \texttt{two or more differences} \\ \pi_j & \texttt{synonymous transversion} \\ \kappa\pi_j & \texttt{synonymous transition} \\ \omega\pi_j & \texttt{non-synonymous transversion} \\ \omega\kappa\pi_j & \texttt{non-synonymous transition} \end{cases} \quad (1)$$

The BSM [26] further models how changes occur along the branches of a phylogenetic tree. It a priori divides the branches into one *foreground* branch and *background* branches. The ω in Eq. 1 takes different values corresponding to Table I. There are two classes of sites along the background branches: conserved ($0 < \omega_0 < 1$) and neutral ($\omega_1 = 1$). On the foreground branch, there is a proportion $1 - p_0 - p_1$ of sites under positive selection with $\omega_2 > 1$. Table I depicts the applied branch-site model A (H_1) (see [9, Table 8.4]). H_0 is the same as model A but with $\omega_2 = 1$ [9, p.281].

The model parameters that need to be estimated are thus: κ, ω_m ($m = \{0, 1, 2\}$ for H_1 or $m = \{0, 1\}$ for H_0), p_0, p_1, and the branch lengths t_k of the tree.

Site class	Proportion	Background	Foreground
0	p_0	$0 < \omega_0 < 1$	$0 < \omega_0 < 1$
1	p_1	$\omega_1 = 1$	$\omega_1 = 1$
2a	$\frac{(1-p_0-p_1)p_0}{p_0+p_1}$	$0 < \omega_0 < 1$	$\omega_2 > 1$
2b	$\frac{(1-p_0-p_1)p_1}{p_0+p_1}$	$\omega_1 = 1$	$\omega_2 > 1$

Table I
PROPORTIONS AND SELECTIVE PRESSURE OF BRANCH-SITE MODEL A FOR CORRESPONDING BRANCHES IN THE FOUR SITE CLASSES.

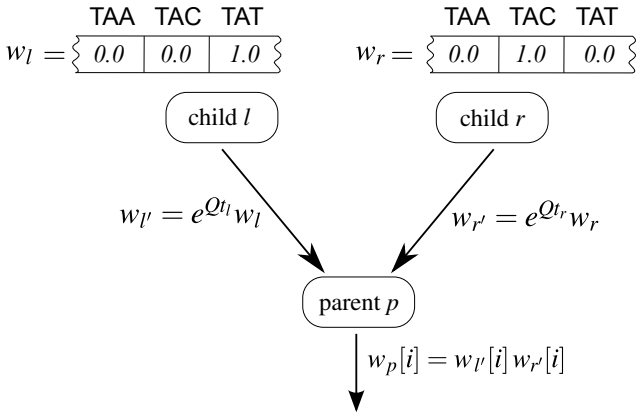

Figure 2. Fundamental operations for computing the phylogenetic likelihood function.

B. Computational approach

In order to compute the likelihood for the BSM, we apply Felsenstein's pruning algorithm [27] that conducts a post-order tree traversal propagating from the leaves toward the root. Along each branch, a partial conditional probability vector for the branch's parent node is computed (Fig. 2) by applying the transition probability matrix to the child's conditional probability vector. The latter one is obtained from the exponential of the instantaneous substitution rate matrix Q multiplied by the corresponding branch length t. At each internal node the conditional probability vectors of each child are multiplied element-wise to obtain the node conditional probability vector. The conditional probability vector of the root node is used, together with the equilibrium codon frequencies, to obtain the likelihood of the BSM.

The maximization of the likelihood of the BSM is achieved through iterative maximization algorithms such as Newton-Raphson methods or an approximation like the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [28, p.162].

C. Likelihood computation

A detailed discussion of phylogenetic likelihood computations can be found in [29]. The computationally most demanding parts to estimate the likelihood of the BSM are the matrix exponential to obtain the transition probability matrix and the computation of the conditional probability vector at each branch. The likelihood of the tree is obtained by computing the dot product of the remaining children's conditional probability vectors at the root of the tree.

1) Transition probability matrix: Given a codon substitution matrix S ($n = 61$ codon states), a diagonal matrix of codon frequencies Π ($\pi_1, \ldots, \pi_{61} > 0$), and a branch length t, the goal is to compute the resulting transition probability matrix P_t. This is done by taking the exponential of the instantaneous rate matrix $Q = S\Pi$ times t to obtain $P_t = e^{Qt}$ (cf. e.g. [30]). The presented approach of computing the matrix exponential P_t is an improved version of the one presented in [31], [9, p.68]. In [32] and [33], the authors present the same decompositional approach but without exploiting the symmetry of S.

First, we transform the problem of computing the matrix exponential of the non-symmetric matrix Qt into the problem of computing the matrix exponential of a *symmetric* matrix At:

$$A := \Pi^{\frac{1}{2}} S \Pi^{\frac{1}{2}}; \tag{2}$$

$$e^{Qt} = \sum_{i=0}^{\infty} \frac{(S\Pi t)^n}{n!}, \tag{3}$$

$$= \Pi^{-\frac{1}{2}} \left(\sum_{i=0}^{\infty} \frac{(At)^n}{n!} \right) \Pi^{\frac{1}{2}}, \tag{4}$$

$$= \Pi^{-\frac{1}{2}} e^{At} \Pi^{\frac{1}{2}}. \tag{5}$$

The new problem of computing the matrix exponential of a symmetric At is always well-conditioned [34] and can be solved efficiently for distinct t using the eigendecomposition of A, i.e., $A = X\Lambda X^\top$, where X contains the left eigenvectors and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_{61})$ contains the real eigenvalues:

$$A = X\Lambda X^\top \Rightarrow \tag{6}$$

$$Z := e^{At} = X e^{\Lambda t} X^\top = \tag{7}$$

$$X \cdot \mathrm{diag}\left(e^{\lambda_1 t}, \ldots, e^{\lambda_{61} t}\right) \cdot X^\top. \tag{8}$$

In [31], the author proceeds with the left to right matrix multiplication, i.e., by computing $\tilde{Y} = X \cdot \mathrm{diag}\left(e^{\lambda_1 t}, \ldots, e^{\lambda_{61} t}\right)$, and

$$Z := \tilde{Y} X^\top. \tag{9}$$

Eq. 9 involves the product of two distinct square matrices, where effective implementations like `dgemm` (BLAS) consume approximately $2n^3$ elementary floating point operations (flops). See, e.g., [35, p.83] for approximate computational costs in flops of basic matrix-matrix operations. As the eigendecomposition of A is symmetric, in *our* approach we compute Z as

$$Z := YY^\top, \tag{10}$$

$$\text{where} \quad Y := X e^{\Lambda t/2}. \tag{11}$$

Eq. 10 involves the product of a square matrix times its transpose, where implementations like `dsyrk` (BLAS) consume approximately n^3 flops. This operation saves about half of the flops and consequently constitutes a major benefit to reduce the runtime of the phylogenetic likelihood function. This benefit *always* applies, but different speedups may be expected for distinct tree topologies.

2) Conditional probability vectors: For a specific branch length t, we need to apply the transition probability matrix $P_t = e^{Qt}$ to a conditional probability vector w, that is, $w' = P_t w$ along all branches of the phylogenetic tree (Fig. 2). This operation is done for each site of the MSA.

After finishing the evaluations on which we report in the next sections, we became aware that a further improvement is possible. The product of the matrix exponential of Qt and the vector w can be computed as follows.

$$e^{Qt} w = \hat{Y}\hat{Y}^\top(\Pi w), \tag{12}$$

$$\text{where} \quad \hat{Y} = \Pi^{-1/2} X e^{\Lambda t/2}. \tag{13}$$

The main advantage compared to the computation above is that he final matrix vector multiplication now involves a symmetric matrix. The latter operation saves about half of the memory accesses and is therefore faster. This is especially beneficial for long MSAs, as the operation is applied to each site.

III. Implementation

CodeML 4 is entirely written in C. SlimCodeML is the optimized version of CodeML v4.4c, where we focus on the BSM. SlimCodeML is written in

compliance with the ISO C++ Standard 1998. We use some routines from the Basic Linear Algebra Subprograms (BLAS) [36]–[38] and the Linear Algebra PACKage (LAPACK) [39], [40] for key parts of the likelihood computations. Thereby we can guarantee high and portable performance of the computational core routines across a large variety of present and future platforms (E.g., [41]).

A. Matrix exponential

Based on the mathematical methods in Section II-C1, the matrix exponential $P_t = e^{Qt}$ has been implemented as a sequence of the following steps.

1. **Matrix $\times$ diagonal matrices.** $A := \Pi^{\frac{1}{2}} S \Pi^{\frac{1}{2}}$. S is multiplied from the left and the right by diagonal matrices, hence this step needs $\mathcal{O}(n^2)$ flops.
2. **Symmetric eigenvalue problem.** This $\mathcal{O}(n^3)$ operation comprises the computation of the eigenvalues Λ and eigenvectors X of A. The eigenvalue problem solver routine `dsyevr` (LAPACK) first reduces the symmetric matrix A to tridiagonal form via Householder transformations. Then, whenever possible, the eigenspectrum is computed using multiple relatively robust representations (MRRR) or a QR/QL method otherwise.
3. **Matrix $\times$ diagonal matrix.** $Y := X e^{\Lambda \frac{t}{2}}$. The dense matrix X is multiplied by a diagonal matrix (containing $e^{\lambda_i \frac{t}{2}}$) from the right. This requires $\mathcal{O}(n)$ flops for building the diagonal matrix and $\mathcal{O}(n^2)$ flops for the matrix multiplication.
4. **Symmetric rank-k operation.** $Z := YY^\top$. We use the symmetric rank-k operation routine `dsyrk` (BLAS) consuming approximately n^3 flops to compute Z. See Eq. 9 and 10 for a discussion of this improvement.
5. **Matrix $\times$ diagonal matrices.** The computation of $P_t := \Pi^{-\frac{1}{2}} Z \Pi^{\frac{1}{2}}$ consumes $\mathcal{O}(n^2)$ flops.

B. Conditional probability vector

$w' := P_t w$. We apply `dgemv` (BLAS) to multiply the previously computed general matrix P_t with the vector w consuming $\mathcal{O}(n^2)$ flops for each site of the MSA. By bundling the individual matrix $\times$ vector operations into a single matrix $\times$ matrix vector operation `dgemm` (BLAS) including all sites of the MSA, we can utilize BLAS level 3 and would further improve runtime performance [42]. However, in order to have a rapid prototype of SlimCodeML, for our first evaluations we abstained from the necessary changes in data structures and hence also from this additional optimization opportunity.

IV. Evaluation

Our evaluations have been done on an Intel Xeon W3540 CPU @ 2.93 GHz and 8 Gbyte RAM, running Ubuntu 11.10, using GCC 4.5.2. We generated the GotoBLAS2 1.13 library for a single-threaded 64-bit architecture (to have a fair comparison with the purely sequential CodeML), and we compiled LAPACK 3.2.2 and linked it against the aforementioned GotoBLAS2. We did not evaluate alternative compilers because the performance of our code is dominated by BLAS and LAPACK performance. As observed in initial tests (data not shown), the influence of the compiler on optimizing the remaining code is negligible. To generate comparable and reproducible results, we fixed the seed for the random number generator, which is used to set the initial tree parameter values in CodeML and in SlimCodeML to the same value. We compared the performance of SlimCodeML to CodeML v4.4c (in the following simply denoted as CodeML).

Table II contains the four datasets we used for evaluation. With respect to the Selectome database, these real-world datasets are representative for the cases i) small number of species / average sequence length, ii) small number of species / very large sequence length, iii) average number of species / small sequence length, and iv) large number of species / short sequence length. Due to the exponential growth of sequenced genomic data, for future analyses we expect both a growing number of species in the MSAs and larger sequences.

Memory consumption of CodeML and SlimCodeML for these datasets is < 200 Mbyte and therefore not performance-critical. Note that due to slightly different intermediate results between SlimCodeML and CodeML, the model parameter optimizations and associated local likelihood optima potentially require a distinct number of iterations. This sensitivity is not a specific property of SlimCodeML, it can also be observed in CodeML when executed with different seed values for the random number generator.

1) Accuracy: We assess the relative differences in obtained likelihood values between CodeML and SlimCodeML (each received with six decimal places) by applying a relative difference $D = \frac{|lnL - \hat{lnL}|}{|lnL|}$, where lnL is the logarithmic likelihood of CodeML

No.	Full name	No. of species	Length (codons)	Ensembl release
i	ENSGT00390000016702.Primates.1.2	7	299	61
ii	ENSGT00580000081590.Primates.1.2	6	5004	55
iii	ENSGT00550000073950.Euteleostomi.7.2	25	67	61
iv	ENSGT00530000063518.Primates.1.1	95	39	61

Table II
ENSEMBL TEST DATASETS (HTTP://ENSEMBL.ORG) AS USED FOR SELECTOME (HTTP://SELECTOME.UNIL.CH) WITH TESTED SUBTREE AND BRANCH.

	CodeML		SlimCodeML	
No.	**Runtime [s]**	**Iterations**	**Runtime [s]**	**Iterations**
i	85	108	43	108
ii	121	80	65	74
iii	1010	241	407	252
iv	52822	1039	8298	509

Table III
RUNTIMES AND NUMBERS OF ITERATIONS FOR CODEML AND SLIMCODEML ON DATASETS I-IV COMBINED FOR H_0+H_1.

and $\hat{lnL}$ is the logarithmic likelihood returned by SlimCodeML. See Section I-A for a description of H_0 and H_1. We obtained D = 0, $9.8 \cdot 10^{-12}$, $5.5 \cdot 10^{-8}$, and $3 \cdot 10^{-9}$ for H_0 datasets i–iv, respectively. For H_1, D = 0, 0, $4.9 \cdot 10^{-8}$, and $1.1 \cdot 10^{-8}$, respectively. The numerical results of CodeML and SlimCodeML are very similar. Therefore, we do not expect to observe differences in the biological interpretation of the results obtained by either implementation.

2) Runtimes and speedups: We measured runtimes of CodeML and SlimCodeML for our test datasets and computed three distinct speedup flavors. The runtimes and corresponding iterations on datasets i–iv can be found in Table III. The deviations in iterations between CodeML and SlimCodeML are caused by slightly different intermediate results (see Section IV); this sensitivity can also be observed by distinct seeds to compute tree parameter start values. A higher number of species usually results in a higher number of branches and consequently in a higher number of iterations and amount of time spent per iteration. This explains the excessive runtimes of dataset iv.

Overall speedups are computed according to $S_o = \frac{S_{t1}}{S_{t2}}$, where S_{t1} is the total runtime of CodeML and S_{t2} is the total runtime of SlimCodeML for either H_0 or H_1, respectively. Per-iteration speedups are computed according to $S_i = \frac{S_{i1}}{S_{i2}}$, where S_{i1} is the total runtime of CodeML and S_{i2} is the total runtime of SlimCodeML normalized by the number of iterations for either H_0 or H_1, respectively. The combined speedup S_c calculates the speedup of H_0 and H_1 combined. Table IV depicts the measured speedups for four different datasets, while Fig. 3 focuses on dataset iv for a varying number of species. In the best case the runtime was reduced from 8 hours and 38 minutes to 55 minutes. The peaks in Fig. 3 can again be explained by deviations in iterations. Per-iteration speedups vary less due to the normalization with respect to the number of iteration steps.

Dataset	**i**	**ii**	**iii**	**iv**
Overall speedup H_0	1.9	2.3	2.6	9.4
Overall speedup H_1	2.0	1.6	2.4	4.4
Combined speedup H_0+H_1	2.0	1.9	2.5	6.4
Per-iteration speedup H_0	2.1	1.8	2.7	3.3
Per-iteration speedup H_1	1.9	1.7	2.5	3.0
Per-iteration speedup H_0+H_1	2.0	1.7	2.6	3.1

Table IV
SPEEDUPS OF SLIMCODEML IN COMPARISON WITH CODEML FOR FOUR DIFFERENT DATASETS.

V. CONCLUSIONS

A. Summary

We presented SlimCodeML, an optimized version of CodeML for the branch-site model (BSM). The improved likelihood computation includes a novel approach to compute the matrix exponential e^{Qt} implemented utilizing the BLAS and LAPACK.

SlimCodeML has very encouraging performance with speedups up to 9.38. On the one hand, the reduced runtime can be attributed to the improved likelihood computation; on the other hand, we partly reach similar likelihoods in a smaller number of iterations. The latter observation suggests further studies concerning the tree parameter optimization process. We achieve high accuracy with respect to likelihood scores.

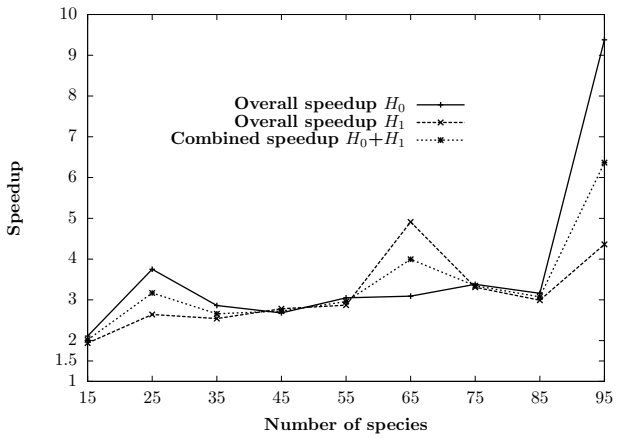

Figure 3. Speedups of SlimCodeML in comparison with CodeML for dataset iv on 15 – 95 species.

B. Future work

SlimCodeML represents the first step toward FastCodeML which will be a new parallel and distributed version of CodeML adapted to the BSM test. While in this paper we focus on the BSM, the optimized likelihood computation can also be applied to further maximum likelihood-based evolutionary models.

C. Rules of thumb

Based on our experience optimizing CodeML, we suggest "rules of thumb" for optimizing the phylogenetic likelihood function which could be applied to a broad range of phylogenetics software, especially for large-scale computations.

- Use BLAS for basic linear algebra operations like scaling of vectors, dot products, matrix $\times$ vector, and matrix $\times$ matrix operations. Benefits: highly tuned for specific architectures, because one can choose from various individual implementations; easy portability to future architectures.
- Use LAPACK for advanced linear algebra operations like eigenvalue problems. Benefits: highly tuned due to utilization of BLAS as basic building block; moreover, benefits from using BLAS are inherited.
- Bundle operations and corresponding data structures (e.g., use a single matrix-matrix multiplication instead of a sequence of matrix-vector operations to utilize BLAS level 3 [42]).
- Exploit matrix properties (e.g., symmetric, triangular, diagonal) by using dedicated routines.
- Consider matrix storage schemes. Row major order (e.g., C) or column major order (e.g., Fortran) have to be respected to increase performance.

ACKNOWLEDGMENTS

This work is supported by the Swiss Platform for High-Performance and High-Productivity Computing (HP2C), the Swiss Federal Government through the Federal Office of Education and Science, and the Swiss National Science Foundation. We thank Sébastien Moretti for providing test datasets.

REFERENCES

[1] M. Anisimova and D. Liberles, "The quest for natural selection in the age of comparative genomics," *Heredity*, vol. 99, no. 6, pp. 567–579, 2007.

[2] R. Jing, A. Vershinin, J. Grzebyta, P. Shaw, P. Smýkal, D. Marshall, M. Ambrose, T. N. Ellis, and A. Flavell, "The genetic diversity and evolution of field pea (Pisum) studied by high throughput retrotransposon based insertion polymorphism (RBIP) marker analysis," *BMC Evol. Biol.*, vol. 10, no. 44, 2010.

[3] S. Blanquart and N. Lartillot, "A site- and time-heterogeneous model of amino acid replacement," *Mol. Biol. Evol.*, vol. 25, no. 5, pp. 842–858, 2008.

[4] N. Rodrigue, H. Philippe, and N. Lartillot, "Mutation-selection models of coding sequence evolution with site-heterogeneous amino acid fitness profiles," *P. Natl. Acad. Sci. USA*, vol. 107, no. 10, pp. 4629–4634, 2010.

[5] M. Han, J. P. Demuth, C. L. McGrath, C. Casola, and M. Hahn, "Adaptive evolution of young gene duplicates in mammals," *Genome Res.*, vol. 19, pp. 859–867, 2008.

[6] R. A. Studer, S. Penel, L. Duret, and M. Robinson-Rechavi, "Pervasive positive selection on duplicated and nonduplicated vertebrate protein coding genes," *Genome Res.*, vol. 18, no. 9, pp. 1393–1402, 2008.

[7] N. Singh, A. Larracuente, T. Sackton, and A. Clark, "Comparative Genomics on the Drosophila Phylogenetic Tree," *Annu. Rev. Ecol. Evol. Syst.*, vol. 40, pp. 459–480, 2009.

[8] P.-A. Christin, D. Weinreich, and G. Besnard, "Causes and evolutionary significance of genetic convergence," *Trends Genet.*, vol. 26, no. 9, pp. 400–405, 2010.

[9] Z. Yang, *Computational Molecular Evolution*. Oxford University Press, 2006.

[10] A. Rosenberg and D. McShea, *Philosophy of Biology – A Contemporary Introduction*. Routledge, 2008.

[11] D. Haussler, S. J. O'Brien, O. A. Ryder, and et al., "Genome 10K: A Proposal to Obtain Whole-Genome Sequence for 10 000 Vertebrate Species," *J. Hered.*, vol. 100, no. 6, pp. 659–674, 2009.

[12] M. Anisimova, J. Bielawski, and Z. Yang, "Accuracy and Power of the Likelihood Ratio Test in Detecting Adaptive Molecular Evolution," *Mol. Biol. Evol.*, vol. 18, no. 8, pp. 1585–1592, 2001.

[13] Z. Yang, W. Wong, and R. Nielsen, "Bayes Empirical Bayes Inference of Amino Acid Sites Under Positive Selection," *Mol. Biol. Evol.*, vol. 22, no. 4, pp. 1107–1118, 2005.

[14] A. Anisimova and Z. Yang, "Multiple Hypothesis Testing to Detect Lineages under Positive Selection that Affects Only a Few Sites," *Mol. Biol. Evol.*, vol. 24, no. 5, pp. 1219–1228, 2007.

[15] Z. Yang, "PAML: a program package for phylogenetic analysis by maximum likelihood," *Comput. Appl. Biosci.*, vol. 13, no. 5, pp. 555–556, 1997.

[16] E. Proux, R. A. Studer, S. Moretti, and M. Robinson-Rechavi, "Selectome: a Database of positive selection," *Nucleic Acids Res.*, vol. 37, pp. 404–407, 2009.

[17] A. Kraut, S. Moretti, M. Robinson-Rechavi, H. Stockinger, and D. Flanders, "Phylogenetic Code in the Cloud – Can it Meet the Expectations?" in *HealthGrid*. IOS Press, 2010, pp. 55–63.

[18] R. A. Studer and M. Robinson-Rechavi, "Large-Scale Analyses of Positive Selection Using Codon Models," in *Evol. Biol.*, P. Pontarotti, Ed. Springer, 2009, pp. 217–235.

[19] S. K. Pond, S. Frost, and S. Muse, "HyPhy: Hypothesis Testing Using Phylogenies," *Bioinformatics*, vol. 21, no. 5, pp. 676–679, 2005.

[20] Z. Yang, "PAML 4: Phylogenetic Analysis by Maximum Likelihood," *Mol. Biol. Evol.*, vol. 24, no. 8, pp. 1586–1591, 2007.

[21] L. Vinh and A. v. Haeseler, "IQPNNI: Moving Fast Through Tree Space and Stopping in Time," *Mol. Biol. Evol.*, vol. 21, no. 8, pp. 1565–1571, 2004.

[22] D. L. Ayres, A. Darling, D. J. Zwickl, P. Beerli, M. T. Holder, P. O. Lewis, J. P. Huelsenbeck, F. Ronquist, D. L. Swofford, M. P. Cummings, A. Rambaut, and M. A. Suchard, "BEAGLE: an Application Programming Interface and High-Performance Computing Library for Statistical Phylogenetics," *Syst. Biol.*, vol. 61, no. 1, pp. 170–173, 2011.

[23] A. Stamatakis, "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[24] Z. Yang, *PAML User Guide Version 4.3*, 2009.

[25] J. Norris, *Markov Chains*. Cambridge University Press, 1997.

[26] J. Zhang, R. Nielsen, and Z. Yang, "Evaluation of an Improved Branch-Site Likelihood Method for Detecting Positive Selection at the Molecular Level," *Mol. Biol. Evol.*, vol. 22, no. 12, pp. 2472–2479, 2005.

[27] J. Felsenstein, "Evolutionary trees from DNA sequences: A maximum likelihood approach," *J. Mol. Evol.*, vol. 17, no. 6, pp. 368–76, 1981.

[28] D. Sorensen and D. Gianola, *Likelihood, Bayesian, and MCMC Methods in Quantitative Genetics*, K. Dietz, M. Gail, K. Krickeberg, J. Samet, and A. Tsiatis, Eds. Springer, 2006.

[29] A. Stamatakis, "Orchestrating The Phylogenetic Likelihood Function on Emerging Parallel Architectures," in *Bioinformatics – High Performance Parallel Computer Architectures*, B. Schmidt, Ed. CRC Press, 2011, pp. 85–115.

[30] T. Müller and M. Vingron, "Modeling Amino Acid Replacement," *J. Comput. Biol.*, vol. 7, no. 6, pp. 761–776, 2000.

[31] Z. Yang, "Notes on Calculation of the Transition Probability Matrix $P(t) = exp(Qt)$," University College London, UK, Tech. Rep., 2003, (bundled with PAML releases).

[32] X. Huang, "Sequence Alignment with an Appropriate Substitution Matrix," *J. Comput. Biol.*, vol. 15, no. 2, pp. 129–138, 2008.

[33] K. Strimmer and A. von Haeseler, "Genetic distances and nucleotide substitution models," in *The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, 2nd ed. Cambridge University Press, 2009, ch. 4.

[34] C. Moler and C. V. Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later," *SIAM Rev.*, vol. 45, no. 1, pp. 3–49, 2003.

[35] R.A. van de Geijn and E.S. Quintana-Ortí, *The Science of Programming Matrix Computations*, 2008.

[36] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM T. Math. Software*, vol. 5, no. 3, pp. 308–323, 1979.

[37] J. Dongarra, "Basic Linear Algebra Subprograms Technical Forum Standard," *J. of High Performance Applications and Supercomputing*, vol. 16, no. 1/2, pp. 1–111/115–199, 2002.

[38] L. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. Whaley, "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," *ACM T. Math. Software*, vol. 28, no. 2, pp. 135–151, 2002.

[39] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. SIAM, 1999.

[40] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. D. Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen, "LAPACK: a portable linear algebra library for high-performance computers," in *Supercomputing*. IEEE, 1990, pp. 2–11.

[41] S. Kestur, J. Davis, and O. Williams, "BLAS Comparison on FPGA, CPU and GPU," in *Annual Symposium on VLSI (ISVLSI)*. IEEE, 2010, pp. 288–293.

[42] K. Goto and R. van de Geijn, "High-Performance Implementation of the Level-3 BLAS," *ACM T. Math. Software*, vol. 35, no. 1, pp. 4:1–4:14, 2008.