Accuracy of a Maximum Likelihood Phylogeny Reconstruction

David Defour

Accuracy of a Maximum Likelihood Phylogeny Reconstruction

David Defour[1]

ELIAUS, Université de Perpignan, 66860 Perpignan, France
{david.defour}@univ-perp.fr

Abstract. We present an accuracy analysis of the Maximum Likelihood Phylogeny Reconstruction kernel used in PhyML. Historically, PhyML was relying on double precision arithmetic to avoid rounding error as well as to minimize the number of underflow. However double precision arithmetic necessitates more memory and computational resources than single precision. This concerns is particularly important when considering an execution on a multicore architecture such as a GPU. This article provides a numerical analysis of the likelihood kernel executed in PhyML. Based on this analysis, solutions to minimize the impact of rounding error and underflow are proposed.

1 Introduction

GPUs are now powerful and programmable processors that have been used to accelerate general-purpose tasks other than graphics applications. These processors rely on a Single Program Multiple Data (SPMD) programming model. This model is carried out by many vector units working in a Single-Instruction Multiple-Data (SIMD) fashion, and vector register files. To get speedup with such hardware, considered applications have to perform many operations on huge set of independent data. Phylogenetic reconstruction software based on maximum likelihood belongs to this category. However porting such an application on this kind of architecture is not straightforward.

Numerical accuracy has been a critical issue when considering a port of an application for a GPU execution. For many years GPUs were lacking floating-point standard compliance and double precision arithmetic. However things have changed with the newest generations of GPU, such has FERMI[1]. Recent GPU are now exposing high computational capabilities in double precision (dp) compared to single precision (sp). There are IEEE 754-2008 compliant and are among the few processors to provide FMA and hardware support for denormal numbers. All these features have an important impact on the accuracy of numerical computation. However double precision numbers are expensive in terms of memory, register usage and bandwidth requirement. On such architecture dp numbers have to be used with parsimony as register usage is a critical issue. The number of hardware registers and the register usage per instance directly impact the number of instance of the same kernel that can be executed simultaneously.

This paper proposes to conduce a numerical analysis of kernel that evaluates maximum likelihood in PhyML in order to investigate the relevance of a port on GPU. The rest of this article is organized as follow. Section 2 gives an introduction on Phylogenetic reconstruction based on maximum likelihood. In section 3 we are providing a worst-case numerical analysis. The previous analysis is completed in section 4 with measures of PhyML execution on benchmarks. In section 5 we are proposing solutions to overcome numerical underflow and rounding error.

2 Likelihood evaluation of a given phylogeny

Phylogenetic attempts to reconstruct, from data about a collection of modern species, a plausible evolutionary history for the group, a history that is most often represented by a bifurcating (binary) tree, called a phylogeny. Phylogenetic softwares are essential tools in the pharmaceutical industry. There exist several techniques for phylogeny reconstruction, which presents different trade-off between speed and accuracy. Among them, techniques based on Maximum Likelihood (ML) with software such as PhyML or RAxML have shown that there are well suited for phylogeny inference out of nucleotide or amino acid sequences.

In this paper we consider PhyML [2], [3] a software based on a hill-climbing algorithm where the tree topology and branch lengths of a unique tree are simultaneously and iteratively modified such that each modification increases the tree likelihood until an optimum is reached. With PhyML model parameters (transition/transversion ratio, gamma shape parameter) are adjusted along the computation. Among all the steps involve in PhyML, the likelihood evaluation is the most time consuming part as it usually take more than 60 % of the total evaluation time. Our objectif is to focus on this part.

The output of PhyML is a tree that maximize the likelihood of each link describing the relation between the N_t species or taxa. The N_t taxa are located at the leaves and the inner node corresponds to extinct ancestors. Each taxa is describes by N_s sites or alignment column taking a value in the set of possible states noted $\mathcal{A}$. The size $N_{\mathcal{A}}$ of the set $\mathcal{A}$ is equal to four for DNA and 20 for amino acids.

Evaluation of likelihood is based on a matrix describing transition probability among each possible states of $\mathcal{A}$ according to a given time or distance. Major model of transition probability for DNA are K80 [4], F81 [5], HKY85 [6], TN93 [7] or GTR [8].

In PhyML, first an initial tree is computed using a fast distance based algorithm like BIONJ [9]. The probabilities π_x of observing states x of $\mathcal{A}$ are determined as well as the free parameters of the Γ model that represents the differences of evolution between different sites. Then the likelihood of the tree is evaluated.

The likelihood L of a given tree under the independence of site is computed by:

$$L = \prod_{s=1}^{N_s} L_s$$

where L_s corresponds to the likelihood of the phylogeny at site s and N_s corresponds to the number of site. However, in real cases for the reconstruction of large phylogeny, L_s are becoming too small to be represented using single or double precision numbers. To avoid numerical underflow due to the multiplication of small quantity, the technique used in PhyML or RAxML adapted from the solution proposed by Yang [10] consists in evaluating the logarithm of the likelihood lnL in place of the computation of the likelihood:

$$lnL = \sum_{s=1}^{N_p} n_s lnL_s \tag{1}$$

Where, N_p correspond to the number of patterns, n_s the weight of pattern s and lnL_s is the logarithm of the likelihood at site s. lnL_s is evaluated for a virtual root vr placed anywhere in the tree as:

$$lnL_s = log(\frac{1}{\delta_{vr}}) + log(l_s(vr)) \tag{2}$$

where δ_{vr} is the scaling factor for the considered virtual root vr. The computation of the likelihood score $l_s(vr)$ for site s at the internal node vr which has two branches u and v is computed as follows:

$$l_s(vr) = \sum_{x \in \mathcal{A}} \left(\pi_x L_s(u = x) \left(\sum_{y \in \mathcal{A}} P_{xy}(b_{uv}) L_s(v = y) \right) \right) \tag{3}$$

where $P_{xy}(b_{uv})$ corresponds to the probability of substitution from state x to y according to the branch length b_{uv} between nodes u and v.

$L_s(r = x)$ corresponds to the conditional likelihood at site s of the sub-tree which has r as root, under the hypothesis that $x \in \mathcal{A}$ is the observed state at this node. If r is a leaf then $L_s(r = x)$ is equal to 1 if x is the observed state at site s, 0 in the other case. If the observed state is a gap or an unknown character then $L_s(r = x)$ is equal to 1. If r is an internal node then the conditional likelihood is computed as follows:

$$L_s(r = x) = \left(\sum_{y \in \mathcal{A}} P_{xy}(b_{ru}) L_s(u = y) \right) \left(\sum_{y \in \mathcal{A}} P_{xy}(b_{rv}) L_s(v = y) \right) \tag{4}$$

Conditional likelihoods are computed using a recursive scheme where each step involves multiplication by small probability. This may leads to extremely

small numbers, smaller than the smallest representable floating-point number. These numerical underflows are dramatic as all the numerical information is lost. This problem is avoided by testing during the conditional likelihood computation at node r, if all the element of L_s are below a threshold. If this is the case, then all the element of L_s are multiplied by a scaling number δ_n, bringing back conditional likelihood closer to the numerical value 1. In order to keep the meaningfulness of the evaluation of the logarithm of the likelihood (equation 2), we need to keep track of every scaling event in order to annihilate them when needed. For a given node r which has two branches u and v, scaling factors are computed along with the conditional likelihood (equation 4) as follows:

$$\delta_r = \delta_u.\delta_v.\delta_n$$

If r is a leaf then δ_r is set to 1. The choice for δ_n depends on the floating-point representation format (sp or dp) but also of the implementation. It is set to a power of 2 in RAxML whereas it is set to $(max_{x \in \mathcal{A}}(L_s(r = x)))^{-1}$ in PhyML. Setting δ_n to a power of 2 has many numerical advantages. Division and multiplication are exact and do not introduce rounding error. In addition logarithm can be precomputed thus saving times and error.

2.1 Optimization phase

PhyML optimizes the tree by adjusting the free parameters of the model (like branch length). This consists in computing the local minima with algorithms like golden section or Brent methods. Local modifications of the tree topology and branch length are applied simultaneously such that the logarithm of the likelihood is converging to a minimum. When a series of modifications leads to a lower value of likelihood, then the currant phylogeny is mixed with the previous tree. Free parameters are also adjusted periodically (every four series).

The impacts of local modifications on the likelihood score are measured through equation 3. This represents the core of the likelihood evaluation. To accelerate this process, PhyML stores for each internal node C that has one ancestor P and two branches U and V the three conditional likelihood (one for each different combination of subtree $\{U, V\}$, $\{P, U\}$, $\{P, V\}$. The three conditional likelihoods are initially computed using a leaf to root followed by a root to leaf step.

2.2 Data structure

Regarding the previous description, the N_t taxa described by the N_s sites taking value in $\mathcal{A}$ are used to build a tree where each of the $N_t - 3$ internal node stores:

- 3 conditional likelihood corresponding to each sub-tree for each possible value of $\mathcal{A}$ for each site.
- A scaling factor for each site of N
- An adapted logarithm likelihood per site.

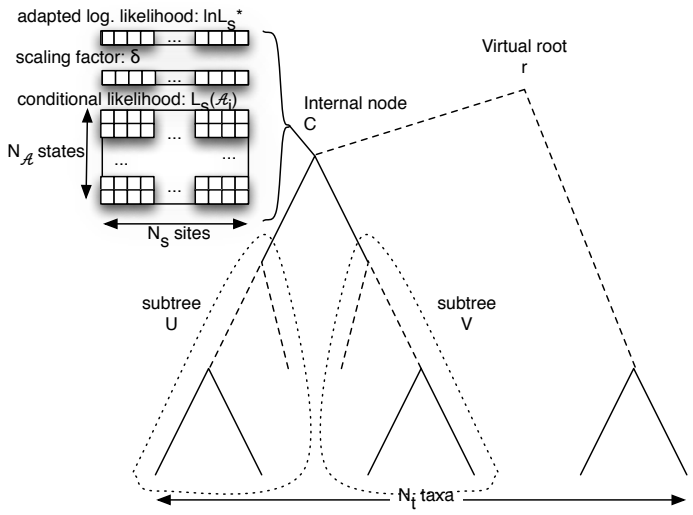


Fig. 1. Description of the internal structure of the phylogeny tree.

This corresponds to $(3.N_{\mathcal{A}} + 2).N_s.(N_t - 3)$ values stored on four or eight bytes for respectively single or double precision floating-point numbers. The memory requirement which is in $\mathcal{O}(N_{\mathcal{A}}.N_s.N_t)$ is dominated by the space for conditional likelihood, and scaling factor when necessary. The scaling vector attached to a given likelihood is not necessary for internal nodes that are close enough to the leaf. Therefore, PhyML is saving some space by allocating memory to the scaling vector only when one or more elements of the scaling vector are different from 1.

3 Numerical analysis

The use of floating-point arithmetic in software leads to several kind of error: error of method, catastrophic cancellation and rounding error. The error of method corresponds to the error due to the approximation of real phenomena by an equation or a formula. Considering likelihood evaluation, this error is encounter in the model description and is not linked with floating-point arithmetic. Catastrophic cancelation is another source of error that happens when subtracting nearby quantities, such that the most significant digits in the operands match and cancel each other. This kind of error is not a problem as likelihood deals with probabilities, which are all positive. However rounding error are of a major concern as likelihood are iteratively computed, which means that rounding error are getting accumulated at each step of the algorithm. For example [11] have notice that numerical instability arises when considering more than 2000 taxa in single precision.

3.1 Floating-point arithmetic and error

The IEEE 754-2008 [12] standard describes floating-point data and representation format such that a floating-point number x is characterized by 3 fields, a mantissa m, an exponent e and a sign s as follows:

$$x = (-1)^s.m.r^e$$

where r is the radix equal to 2 or 10. In the rest of this paper we are dealing with the binary32 and binary64 formats of the IEEE 754-2008 that were formerly known as the single (sp) and double formats (dp) in the IEEE 754-1985. In these format the radix is set to 2, the mantissa is respectively represented on $23 + 1$ or $52 + 1$, with exponent e in the range $[-126; +127]$ or $[-1022; +1023]$.

Throughout the paper, we will note $+$, $-$, and $\times$ the usual mathematical operations, and $\oplus$, $\ominus$ and $\otimes$ the corresponding operations in IEEE 754-2008 floating-point arithmetic, in the *round to nearest* mode. Similarly we represent by x^* the floating-point approximation of the real mathematical value x.

Theorem 1 (Relative error). *The relative error E_R is the difference between the real number x and its floating-point approximation x^* divided by x.*

$$E_R = \frac{|x - x^*|}{|x|}$$

The relative error provides a lower bound on the minimum numbers of reliable digits in the result, compared to the exact mathematical result. To ease the computation of rounding error we introduce the quantity ϵ_n:

Theorem 2 (ϵ_n). *For any integer n, we will represent by ϵ_n a quantity α such that*

$$|\alpha| \leq 2^n$$

For example, based on this representation, the rounding error done during the multiplication or addition of two FP number a and b is such that

$$a \otimes b = (a \times b).(1 + \epsilon_\eta)$$

or

$$a \oplus b = (a + b).(1 + \epsilon_\eta)$$

with $\eta = -24$ with single precision arithmetic or $\eta = -53$ with double precision arithmetic's. We can noticed that $\epsilon_a.\epsilon_b = \epsilon_{a+b}$ and $k.\epsilon_a = \epsilon_{a+\log_2(k)}$. The interested reader can read [13] for further discussion around numerical problems related to floating-point usage.

3.2 Worst-case rounding error analysis

Every floating-point operation introduces a rounding error in the computed results. To determine its impact we compute a guaranteed bound of the rounding error based on the worst-case behavior. This worst-case scenario infers that all the rounding errors are maximum and happen in the same direction.

Let $L_s^*(r = x)$ be the computed conditional likelihood. Then at each node of the internal tree, we compute:

$$\begin{aligned} L_s^*(r = x) &= [\sum_{y \in \mathcal{A}} P_{xy}(b_{ru}) \otimes L_s(u = y)] \otimes [\sum_{y \in \mathcal{A}} P_{xy}(b_{rv}) \otimes L_s(v = y)] \\ &= [\sum_{y \in \mathcal{A}} (P_{xy}(b_{ru}) \times L_s(u = y))(1 + \epsilon_\eta)] \otimes [\sum_{y \in \mathcal{A}} P_{xy}(b_{rv}) \otimes L_s(v = y)] \\ &= [\sum_{y \in \mathcal{A}} P_{xy}(b_{ru}) \times L_s(u = y)](1 + \epsilon_{\eta + \log_2(N_\mathcal{A})}) \otimes \\ &\quad [\sum_{y \in \mathcal{A}} P_{xy}(b_{rv}) \otimes L_s(v = y)](1 + \epsilon_{\eta + \log_2(N_\mathcal{A})}) \\ &= L_s(r = x)(1 + \epsilon_{\eta + \log_2(N_\mathcal{A}) + 1}) \end{aligned} \tag{5}$$

For each conditional likelihood computation, $(\log_2(N_\mathcal{A})+1)$ bits of the results can be lost due to rounding error. This corresponds to 3 bits for DNA input data and 5.32 bits for AA input data.

Conditional likelihood computation is an iterative process conduced at each internal node of the considered tree describing the relation among the N_t taxa. At each step, rounding errors are cumulated. To simplify rounding error evaluation, let's place the virtual root such that the depth of the tree is minimal. This corresponds to an equilibrate tree of depth $\log_2(N_t)$. The maximum rounding error for the computation of the conditional likelihood at the virtual root, is $\log_2(N_t).(\log_2(N_\mathcal{A}) + 1)$ bits. All bits of the computed results of the conditional likelihood can be lost when the number of considered taxa is greater than:

	Single-precision	Double-precision
DNA	$2^6 = 64$	$2^{17.66} = 207104$
AA	$2^{4.51} = 22.8$	$2^{9.96} = 996$

We observe that with AA input data, the conditional likelihood computed in single precision might be meaningless for the reconstruction of a tree with more than 23 species !

The logarithm of the likelihood of a tree is a combination of the conditional likelihood for every site as in equation 1. During this process, the summation of the N_s conditional likelihood can introduce $\log_2 N_s$ extra bits or error. The maximum relative error associated with the computation of lnL is $2^{\log_2(N_s)+\log_2(N_t).(\log_2(N_\mathcal{A})+1)}$ reducing even more the number of taxa that can be considered safely.

Rounding error occurs during the optimization phase as well. The computation of the likelihood score is evaluated using equation 3. Let $l_s^*(vr)$ be the computed likelihood score at node vr for the site s.

$$\begin{aligned} l_s^*(vr) &= \sum_{x\in\mathcal{A}} \left(\pi_x \otimes L_s(u=x) \otimes \left(\sum_{y\in\mathcal{A}} P_{xy}(b_{uv}) \otimes L_s(v=y) \right) \right) \\ &= \sum_{x\in\mathcal{A}} \left((\pi_x \times L_s(u=x)))(1+\epsilon_\eta) \otimes \left(\sum_{y\in\mathcal{A}} P_{xy}(b_{uv}) \times L_s(v=y) \right) (1+\epsilon_{\eta+\log_2(N_\mathcal{A})}) \right) \\ &= \sum_{x\in\mathcal{A}} \left(\pi_x \times L_s(u=x) \times \left(\sum_{y\in\mathcal{A}} P_{xy}(b_{uv}) \times L_s(v=y) \right) (1+\epsilon_{\eta+\log_2(N_\mathcal{A})} + \epsilon_{\eta+1}) \right) \\ &= \left(\sum_{x\in\mathcal{A}} \pi_x \times L_s(u=x) \times \left(\sum_{y\in\mathcal{A}} P_{xy}(b_{uv}) \times L_s(v=y) \right) \right) (1+\epsilon_{\eta+2.\log_2(N_\mathcal{A})} + \epsilon_{\eta+\log_2(N_\mathcal{A}+1)}) \end{aligned} \tag{6}$$

We observe that the rounding error accumulated during the evaluation of the likelihood score can affect the result with a relative error less than $2^{\eta+2.\log_2(N_\mathcal{A})} + 2^{\eta+\log_2(N_\mathcal{A})+1}$. This means that up to 4.3 bits may be lost for DNA input data and 8.7 bits for AA input data.

4 Measured error

4.1 Rounding error analysis on real cases

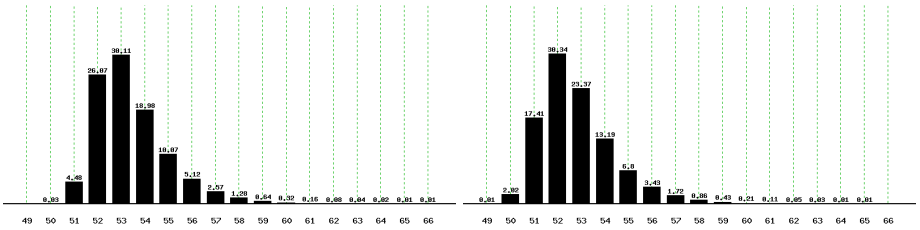
Worst-case analysis is necessary for critical system (plane, missile, satellite, ..) to prevent behavior with dramatic consequences to happen. However phylogenetic reconstruction can tolerates rare rounding error if it leads to a faster reconstruction. The worst-case analysis described in the previous section is pessimistic and do not reflect most of the cases. For example rounding error may compensate each other's or are too often overestimated especially near the leaf where conditional likelihood is exact (0 or 1).

We have executed PhyML on sets of data available on PhyML website. These data set have the following caracteristic:

Data set	Type	Nb of pattern	Nb of species
DNA1	DNA	382	54
AA1	AA		37
DNA2	DNA	896	1566
AA2	AA		

We have designed a version of PhyML based on MPFR to measure the rounding error that occurs during the computation of likelihood score and compared

it to the double precision version of PhyML. We used both DNA and AA input data set distributed with PhyML. The measured rounding error were collected for all the likelihood score to obtained the distribution of the error for DNA (Figure 2,a) and AA (Figure 2,b). Over the 105588238 computations of likelihood score involved with DNA input data, we measured a relative maximum error of $2^{-50.08}$. For AA data, the maximum error was of $2^{-49.14}$ over the 60180549 computation. The measured maximum rounding error is much lower than the maximum error we could expect (section 3.2). This comes from the fact that rounding errors are compensating one another in most of the cases and are rarely getting accumulated. This is confirmed by the graphics of the rounding error distribution. We observe that more than 70 % of the rounding errors are located between 2^{-51} and 2^{-54} and only 0.01% are located close to the worst case.



(a) DNA (b) AA

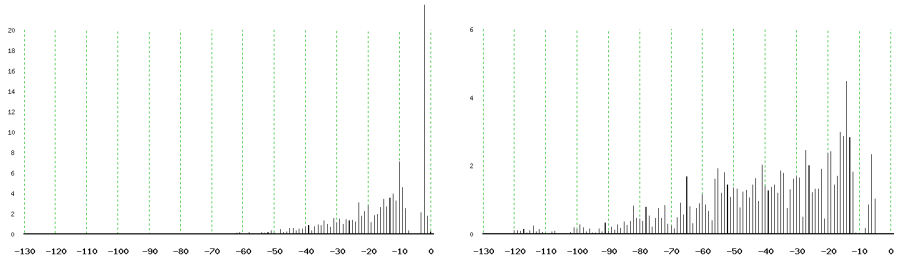
Fig. 2. Cumulated rounding error distribution of the conditional likelihood computation in double precision for DNA and AA data. The abscise x represent the relative error 2^{-x} and the bar represents the proportion of error located between 2^{-x} and 2^{-x-1}

4.2 Likelihood range estimation

Likelihood computation deals with probability numbers. These numbers are ranging from 0 to 1. A worst-case analysis to determine the floating-point range used during the evaluation of the likelihood score is uninformative as it always consider the smallest probability within the transition model, which leads quickly to the smallest exponent possible. We measured the distribution of the exponent over the computation of conditional likelihood for the AA and DNA set of input data provided with PhyML. The results are given in Figure 3.

5 Propositions

Likelihood computation involves two kinds of numerical problems due to the use of floating-point arithmetic as seen in the previous section: accumulation of rounding error and underflow. In this section we are proposing solutions to overcome both these problems.



(a) DNA (b) AA

Fig. 3. Distribution of the floating-point exponent of the conditional likelihood in double precision for DNA and AA data.

5.1 Underflow

Underflows happen with large phylogenetic trees as at each internal node we are multiplying likelihoods, which correspond to extremely small numbers. There are two solutions to deal with underflow.

The first solution consists is increasing the range of representable floating-point numbers. The fastest is to rely on format handled in hardware. For example, the single precision, the double precision, and the extended double precision format of the IEEE 754-2008 standard store the exponent on respectively 8, 11 and 15 bits. However this may not be enough for large phylogeny (figure 3). In that case, the only solution is to use floating-point numbers handled in software. Library like MPFR [14] allows the user to customize the floating-point format used to store and perform computation. However the extra memory and operations is a major problem with likelihood computation.

The second solution is to fully exploit the range offered by floating-point format. As it is noticed in [11], scaling can dramatically decrease performance of the likelihood computation. Likelihood corresponds to probabilities, which range between 0 and 1. Therefore only half of the representation range of both single and double floating-point representation format is exploited. One way to diminish scaling is to exploit full range of floating-point format. This can be done by setting initial conditional likelihood for every leaf to MAX_EXP, with $MAX\_EXP = 2^{+127}$ for sp and $MAX\_EXP = 2^{+1023}$ for dp. This trick will involves division by MAX_EXP every time conditional likelihood are multiplied together. In a similar manner, the scaling steps performed during conditional likelihood computation can be optimized by choosing scaling factor that will make likelihood close to MAX_EXP instead of 1.

5.2 Rounding error

Rounding errors are of a major concern as we noticed in section 3.2. There are several solutions to minimize the impact of rounding error. One can use floating-point arithmetic with more precision likes quad floating-point numbers or use software library of floating-point arithmetic that increases the precision like the

double-double or quad-double floating-point format [15] or MPFR [14]. Another solution is to use compensate arithmetic [16] which keeps track of rounding error in an extra floating-point number. But all these solutions require more memory and more basic operations. This will increase the cost in times and memory of phylogenetic reconstruction by a factor of at least five in time and two in memory.

An interesting approach is to use single precision representation format. This solution is necessary to consider an efficient implementation on accelerator such as GPU. It will divide by two the memory necessary to store conditional values, the bandwidth to transfer data and the time to execute floating-point operations as more instance can run in parallel. However this solution presents the drawback of diminishing the number of floating-point operations that can be considered to evaluation conditional likelihood leading to numerical problems encounter with more than 2000 species in RAxML [11].

An alternative is to develop a mixed precision implementation. Use double precision when accuracy is necessary and single precision to accelerate computation. With the emergence of GPGPU, this alternative is gaining interests [17], [18], [19]. Mixed precision implementation can be address in several ways.

More than 60% of the execution time is used to evaluate likelihood score. A prudent mixed precision implementation can compute and store conditional likelihood in dp to keep good accuracy and perform score likelihood computation in single precision. This solution does not reduce the memory necessary to store conditional likelihood but will reduce its execution time by allowing a SIMD implementation.

Another solution is to benefit of the accuracy of dp during computation and single precision during storage. This solution passes by an iterative computation of conditional likelihood done in dp and once computed stored in sp. Then all the likelihood score evaluations can also be done in sp. The objective is to start with conditional likelihood that do not embed large rounding error and then perform optimization at full speed with single precision on the branches of the tree. With this solution the accumulation of rounding error depends more on the number of optimization done on the tree than on the depth of it. However rounding error are still getting accumulated, which impose recomputing at full precision conditional likelihood periodically.

The algorithm describing the computation of conditional likelihood is described in pseudo C in Listing 1.1. We observe that only three temporary nodes storing conditional likelihood in full precision are necessary.

Listing 1.1. Algorithm for conditional likelihood evaluation in mixed precision

```
mixed_lk_eval(Node){
        double * lk_left, *lk_right, *lk_node;

        *lk_left = mixed_lk_eval(Node->left);
        *lk_right = mixed_lk_eval(Node->right);

        *lk_node = malloc(nb_element * sizeof(double));
        *lk_node = compute_cond_likelihood(lk_left, lk_right);
```

```
        free(lk_left);
        free(lk_left);
        Node->lk = convert_lk_in_sp(lk_node);
}
```

6 Tests

It Hhs to be done. I'm not an expert in PhyML. When I tried the previous proposition, I broke everything, and nothings were working correctly after that !

7 conclusion

We have done a numerical analysis of the core computation of PhyML, a phylogenetic reconstruction software based on maximum likelihood. We have seen that rounding errors as well as numerical underflows are of a major concern. The worst-case analysis is too pessimistic as phylogenetic reconstruction can tolerate rounding error if there are rare enough. We have measured the probability of encountering the worst case on real execution of PhyML. We have given several propositions in order to minimize the impact of underflow as well as rounding error.

This study is necessary before an implementation on GPU were single precision is more efficient than double precision by several orders of magnitude. PhyML will definitely benefit of the presence of the FMA available in the latest generation of GPU such as the Nvidia FERMI as it will divide by two the number of rounding error. However the ratio of floating-point operation done per data is not sufficient yet to expect good performance out of a straightforward implementation on a GPU. The internal algorithm has to be reconsidered.

References

1. NVIDIA: Nvidia's next generation cuda compute architecture: Fermi. whitepaper, nvidia
2. Guindon, S.: Méthodes et algorithmes pour l'approche statistique en phylogénie. PhD thesis, Université Montpellier II (Jully 2003)
3. Guindon, S., Gascuel, O.: A simple, fast and accurate algorithm to estimate large phylogenies by maximum likelihood. Systematic Biology **52** (2003) 696–704
4. Kimura, M.: A Simple Method for Estimating Evolutionary Rates of Base Substitutions Through Comparative Studies of Nucleotide Sequences. Journal of Molecular Evolution **10**(111–120) (1980)
5. Felsenstein, J.: Evolutionary trees from DNA sequences : a maximum likelihood approach. Journal of Molecular Evolution **17** (1981) 368–376
6. Hasegawa, M., Kishino, H., Yano, T.: Dating of the human-ape splitting dating of the human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution **22** (1985) 160–174

7. Tamura, K., Nei, M.: Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. Mol. Biol. Evol. **10** (1993) 512–526
8. Lanave, C., Preparata, G., Sacone, C., Serio3, G.: A new method for calculating evolutionary substitution rates. Journal of Molecular Evolution **20** (1984) 86–93
9. Gascuel, O.: BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. Mol. Biol. Evol. **14** (1997) 685–695
10. Yang, Z.: Maximum likelihood estimation on large phylogenies and analysis of adaptive evolution in human influenza virus a. Journal of Molecular Evolution **51**(5) (2000) 423–432
11. Berger, S., Stamatakis, A.: Accuracy and performance of single versus double precision arithmetics for maximum likelihood phylogeny reconstruction. Proceedings of PBC09, Parallel Biocomputing Workshop, Wroclaw, Poland, (September 2009)
12. : IEEE standard for floating-point arithmetic. (2008) 1–58
13. Goldberg, D.: What every computer scientist should know about floating point arithmetic. ACM Computing Surveys **23**(1) (1991) 5–48
14. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software **33**(2) (2007) 13:1–13:15
15. Hida, Y., Li, X.S., Bailey, D.H.: Algorithms for quad-double precision floating point arithmetic. Computer Arithmetic, IEEE Symposium on **0** (2001) 0155
16. Rump, S.M., Ogita, T., Oishi, S.: Accurate floating-point summation. Technical Report 05.12, Hamburg University of Technology, Hamburg, Germany (2005)
17. Goddeke, D., Strzodka, R., Turek, S.: Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in fem simulations. International Journal of Parallel, Emergent and Distributed Systems **22**(4) (January 2007) 221–256
18. Buttari, A., Dongarra, J., Langou, J., Luszczek, P., Kurzak, J.: Mixed precision iterative refinement techniques for the solution of dense linear systems. International Journal of High Performance Computing Applications **21**(4) (2007) 457–466
19. Grand, S.L., Goetz, A.W., Xu, D., Poole, D., Walker, R.C.: Achieving high performance in amber pme simulations using graphics processing units without compromising accuracy. Technical report (2010)