An improvement to DPPDIV

Tomáš Flouri[1] and Alexandros Stamatakis[1,2]

[1] Heidelberg Institute for Theoretical Studies, Germany
[2] Karlsruhe Institute of Technology, Institute for Theoretical Informatics, Postfach 6980, 76128 Karlsruhe

1 Introduction

DPPDIV [3] is a program for estimating species divergence times and lineage-specific substitution rates on a fixed topology. The prior on branch rates is a Dirichlet process prior (DPP) which clusters branches into distinct rate classes. Alternative priors including the global molecular clock and the independent rates model are also available. The priors on node ages include the birth-death (and Yule) model and the uniform distribution. The likelihood is calculated using the sum product algorithm [2, 1]. Markov chain Monte Carlo (MCMC) sampling is used to approximate the posterior distributions of the various parameters and hyperparameters and to obtain estimates of brach rates and divergence times. The proposal mechanism for updating the lineage-specific substitution rates under the DPP is Algorithm 8 described by [5] and implemented in other phylogenetic methods employing this prior [4]. This approach uses Gibbs sampling to update the rate class assignments for each branch by evaluating the relative probabilities of all possible reassignments to existing classes and to placement in new auxiliary classes [5]. The number of auxiliary categories is fixed at four in this implementation to mitigate the computational burden of each Dirichlet process update while still adequately searching parameter space. An additional update is performed to propose changes to the rate values associated with every existing category using a rate multiplier.

In this report, we present a parallel implementation of the likelihood function of DPPDIV that works on multi-core architectures. Moreover, we give two optimised versions of the likelihood functions using the Streaming SIMD Extensions 2 and 3 (SSE), and using the Advanced Vector Extensions (AVX). We first provide a description of the SSE and AVX implementation as well as the parallel extension using OpenMP and test the performances of the implemenentations against the original single-thread program. The SSE-badsed vectorization is a straight-forward

2 Implementation

After profiling DPPDIV, we have discovered that the program spends over 95% of its runtime in a function which computes first the conditional likelihoods of each node of the candidate tree, for each possible site and evolutionary rate. In order to increase the speed of computation we have implemented two optimized versions of the likelihood function which make use of SIMD architecture-specific instrunctions. The first version makes use of the SSE3 instruction set and the second uses the AVX instruction set. Moreover, we have developed a parallel implementation of the likelihood function using OpenMP which works on multi-core architecture and may also make use of the SSE or AVX instruction set.

In the rest of this section we describe the SSE3 and AVX implementations of the computation of the conditional likelihoods for each node, and the computation of the tree likelihood. Finally, we briefly explain the method of parallelization.

2.1 Computation of conditional likelihoods

We first compute the conditional likelihoods for each internal node of the candidate tree, for each site and for each evolutionary rate category. These are computed by a bottom-up traversal of the tree. For a node k that has nodes i and j as its two immediate descendants, the conditional

likelihoods for each possible site assignment is calculated using the sum product method described in [1]

$$L_{S_k}^{(k)} = \Big(\sum_{S_i=A}^{T} P_{S_k S_i}(b_i) L_{S_i}^{(i)} \Big) \Big(\sum_{S_j=A}^{T} P_{S_k S_j}(b_j) L_{S_j}^{(j)} \Big) \tag{1}$$

where $L_{S_k}^{(k)}$ is the likelihood of the data in the subtree rooted at node k, given that the nucleotide state S at k is fixed, and i and j are the two immediate descendants of k. If k is a tip and consists of a nucleotide, say A, then $L_A^{(k)} = 1$ and $L_C^{(k)} = L_G^{(k)} = L_T^{(k)} = 0$. The function $P_{S_k S_i}(b_i)$ gives the probability that a base S_k evolves into S_i after the time given by the branch b_i from i to k.

The conditional likelihoods for each of the four evolutionary rates and each of the four nucleic bases are stored in the corresponding node as vectors. We concentrate on the computation of the conditional likelihoods of only one evolutionary rate as the rest can be iteratively computed in the same way.

The range of the probability function P given the branch lengths of i and j has been precomputed and mapped to the elements of matrices $P^{(i)}$ and $P^{(j)}$, which represent the probability matrices for the left and right descendant of node k, respectively.

$$P^{(i)} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \qquad P^{(j)} = \begin{pmatrix} a'_{1,1} & a'_{1,2} & a'_{1,3} & a'_{1,4} \\ a'_{2,1} & a'_{2,2} & a'_{2,3} & a'_{2,4} \\ a'_{3,1} & a'_{3,2} & a'_{3,3} & a'_{3,4} \\ a'_{4,1} & a'_{4,2} & a'_{4,3} & a'_{4,4} \end{pmatrix}$$

The conditional likelihoods for each possible nucleotide assignment at states i and j are denoted by $L^{(i)}$ and $L^{(j)}$, respectively.

$$L^{(i)} = (b_1, b_2, b_3, b_4) \qquad L^{(j)} = (b'_1, b'_2, b'_3, b'_4)$$

SSE version The actual SIMD instructions are invoked using intrinsic functions and the corresponding vector data types. In this vectorization scheme, we use SSE instructions to load in a vector register r_5 the first two terms

$$a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2,$$
$$a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2$$

of the first two iterations of the left summation of equation 1. The last two terms of the those two iterations are stored in a r_{11}, i.e.

$$r_5[a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2, a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2]$$
$$r_{11}[a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4, a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4]$$

The corresponding elements of the two vectors are then added (vector register r_{12}) to get the left summation part of equation 1 for the first two possible site assignments resulting from the evolution of the site in the left descendant. Schematically, the whole process is described in the following way:

$$
\begin{array}{l}
a_{1,1} \\ a_{1,2}
\end{array}
\xrightarrow{\texttt{\_mm\_load\_pd}} r_0[a_{1,1}, a_{1,2}]
$$

$$
\xrightarrow{\texttt{\_mm\_mul\_pd}} r_2[a_{1,1} \cdot b_1, a_{1,2} \cdot b_2]
$$

$$
\begin{array}{l}
b_1 \\ b_2
\end{array}
\xrightarrow{\texttt{\_mm\_load\_pd}} r_1[b_1, b_2]
$$

$$
\begin{array}{l}
a_{2,1} \\ a_{2,2}
\end{array}
\xrightarrow{\texttt{\_mm\_load\_pd}} r_3[a_{2,1}, a_{2,2}]
$$

$$
\xrightarrow{\texttt{\_mm\_mul\_pd}} r_4[a_{2,1} \cdot b_1, a_{2,2} \cdot b_2]
$$

$$
r_1
$$

$$
\begin{array}{l}
r_2 \\ r_4
\end{array}
\xrightarrow{\texttt{\_mm\_hadd\_pd}} r_5[a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2, a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2]
$$

$$
\begin{array}{l}
a_{1,3} \\ a_{1,4}
\end{array}
\xrightarrow{\texttt{\_mm\_load\_pd}} r_0[a_{1,3}, a_{1,4}]
$$

$$
\xrightarrow{\texttt{\_mm\_mul\_pd}} r_8[a_{1,3} \cdot b_3, a_{1,4} \cdot b_4]
$$

$$
\begin{array}{l}
b_3 \\ b_4
\end{array}
\xrightarrow{\texttt{\_mm\_load\_pd}} r_7[b_3, b_4]
$$

$$
\begin{array}{l}
a_{2,3} \\ a_{2,4}
\end{array}
\xrightarrow{\texttt{\_mm\_load\_pd}} r_9[a_{2,3}, a_{2,4}]
$$

$$
\xrightarrow{\texttt{\_mm\_mul\_pd}} r_{10}[a_{2,3} \cdot b_3, a_{2,4} \cdot b_4]
$$

$$
r_7
$$

$$
\begin{array}{l}
r_8 \\ r_{10}
\end{array}
\xrightarrow{\texttt{\_mm\_hadd\_pd}} r_{11}[a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4, a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4]
$$

$$
\begin{array}{l}
r_5 \\ r_{11}
\end{array}
\xrightarrow{\texttt{\_mm\_add\_pd}} r_{12}[a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2 + a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4, a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2 + a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4]
$$

We then compute the right summation part of equation 1 for the first two possible site assignments, in exactly the same way as in the case of the left summation part,

$$
r_{13}[a'_{1,1} \cdot b'_1 + a'_{1,2} \cdot b'_2 + a'_{1,3} \cdot b'_3 + a'_{1,4} \cdot b'_4, a'_{2,1} \cdot b'_1 + a'_{2,2} \cdot b'_2 + a'_{2,3} \cdot b'_3 + a'_{2,4} \cdot b'_4]
$$

and we multiply the corresponding elements of r_{12} and r_{13} to obtain the conditional likelihoods for state k for the first two possible site assignments. Schematically, using SSE intrinsics

$$
\begin{array}{l}
r_{12} \\ r_{13}
\end{array}
\xrightarrow{\texttt{\_mm\_mul\_pd}} r_{14}[c \cdot d, e \cdot f]
$$

where

$$
\begin{aligned}
c &= a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2 + a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4 \\
d &= a'_{1,1} \cdot b'_1 + a'_{1,2} \cdot b'_2 + a'_{1,3} \cdot b'_3 + a'_{1,4} \cdot b'_4 \\
c &= a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2 + a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4 \\
d &= a'_{2,1} \cdot b'_1 + a'_{2,2} \cdot b'_2 + a'_{2,3} \cdot b'_3 + a'_{2,4} \cdot b'_4
\end{aligned}
$$

Finally, we apply the exact same process described in this section to compute the conditional likelihoods for the other two possible site assignments.

AVX version The main difference between the SSE and AVX implementations is in the fact that AVX instructions work with 256-bit registers and thus we can simultaneously load all four terms of each summation part in equation 1. We then perform the multiplications to get the conditional likelihood for all four site assignements in just one step as opposed to SSE, which does the whole process in two steps since it works with 128-bit vector registers (which can hold two elements).

Compared to the SSE version, we obtain the first two terms of each iteration of the left summation in register r_{11} and the last two terms in register r_{12}. Schematically,

$$\begin{matrix} a_{1,1}\ a_{1,3} \\ a_{1,2}\ a_{1,4} \end{matrix} \xrightarrow{\texttt{\_mm256\_load\_pd}} r_0[a_{1,1}, a_{1,2}, a_{1,3}, a_{1,4}]$$

$$\xrightarrow{\texttt{\_mm256\_mul\_pd}} r_2[a_{1,1} \cdot b_1, a_{1,2} \cdot b_2, a_{1,3} \cdot b_3, a_{1,4} \cdot b_4]$$

$$\begin{matrix} b_1\ b_3 \\ b_2\ b_4 \end{matrix} \xrightarrow{\texttt{\_mm256\_load\_pd}} r_1[b_1, b_2, b_3, b_4]$$

$$\begin{matrix} a_{2,1}\ a_{2,3} \\ a_{2,2}\ a_{2,4} \end{matrix} \xrightarrow{\texttt{\_mm256\_load\_pd}} r_3[a_{2,1}, a_{2,2}, a_{2,3}, a_{2,4}]$$

$$\xrightarrow{\texttt{\_mm256\_mul\_pd}} r_4[a_{2,1} \cdot b_1, a_{2,2} \cdot b_2, a_{2,3} \cdot b_3, a_{2,4} \cdot b_4]$$

$$r_1$$

$$\begin{matrix} a_{3,1}\ a_{3,3} \\ a_{3,2}\ a_{3,4} \end{matrix} \xrightarrow{\texttt{\_mm256\_load\_pd}} r_5[a_{3,1}, a_{3,2}, a_{3,3}, a_{3,4}]$$

$$\xrightarrow{\texttt{\_mm256\_mul\_pd}} r_6[a_{3,1} \cdot b_1, a_{3,2} \cdot b_2, a_{3,3} \cdot b_3, a_{3,4} \cdot b_4]$$

$$r_1$$

$$\begin{matrix} a_{4,1}\ a_{4,3} \\ a_{4,2}\ a_{4,4} \end{matrix} \xrightarrow{\texttt{\_mm256\_load\_pd}} r_7[a_{4,1}, a_{4,2}, a_{4,3}, a_{4,4}]$$

$$\xrightarrow{\texttt{\_mm256\_mul\_pd}} r_8[a_{4,1} \cdot b_1, a_{4,2} \cdot b_2, a_{4,3} \cdot b_3, a_{4,4} \cdot b_4]$$

$$r_1$$

$$\begin{matrix} r_2 \\ r_4 \end{matrix} \xrightarrow{\texttt{\_mm256\_hadd\_pd}} r_9[a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2, a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2, a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4, a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4]$$

$$\begin{matrix} r_6 \\ r_8 \end{matrix} \xrightarrow{\texttt{\_mm256\_hadd\_pd}} r_{10}[a_{3,1} \cdot b_1 + a_{3,2} \cdot b_2, a_{4,1} \cdot b_1 + a_{4,2} \cdot b_2, a_{3,3} \cdot b_3 + a_{3,4} \cdot b_4, a_{4,3} \cdot b_3 + a_{4,4} \cdot b_4]$$

$$\begin{matrix} r_9 \\ r_{10} \end{matrix} \xrightarrow{\texttt{\_mm256\_blend\_pd,0b1100}} r_{11}[a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2, a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2, a_{3,3} \cdot b_3 + a_{3,4} \cdot b_4, a_{4,3} \cdot b_3 + a_{4,4} \cdot b_4]$$

$$\begin{matrix} r_9 \\ r_{10} \end{matrix} \xrightarrow{\texttt{\_mm256\_permute2f128\_pd,0x21}} r_{12}[a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4, a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4, a_{3,1} \cdot b_1 + a_{3,2} \cdot b_2, a_{4,1} \cdot b_1 + a_{4,2} \cdot b_2]$$

Finally, we get all terms of the left summation for each possible site assignment by adding together the corresponding elements of registers r_{11} and r_{12}

$$\begin{matrix} r_{11} \\ r_{12} \end{matrix} \xrightarrow{\texttt{\_mm256\_add\_pd}} r_{13}[c, d, e, f]$$

where

$$\begin{aligned} c &= a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2 + a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4 \\ d &= a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2 + a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4 \\ e &= a_{3,1} \cdot b_1 + a_{3,2} \cdot b_2 + a_{3,3} \cdot b_3 + a_{3,4} \cdot b_4 \\ f &= a_{4,1} \cdot b_1 + a_{4,2} \cdot b_2 + a_{4,3} \cdot b_3 + a_{4,4} \cdot b_4 \end{aligned}$$

In similar fashion we obtain the right summation part in register r_{14}

$$r_{14}[c', d', e', f']$$

where

$$
\begin{aligned}
c' &= a'_{1,1} \cdot b'_1 + a'_{1,2} \cdot b'_2 + a'_{1,3} \cdot b'_3 + a'_{1,4} \cdot b'_4 \\
d' &= a'_{2,1} \cdot b'_1 + a'_{2,2} \cdot b'_2 + a'_{2,3} \cdot b'_3 + a'_{2,4} \cdot b'_4 \\
e' &= a'_{3,1} \cdot b'_1 + a'_{3,2} \cdot b'_2 + a'_{3,3} \cdot b'_3 + a'_{3,4} \cdot b'_4 \\
f' &= a'_{4,1} \cdot b'_1 + a'_{4,2} \cdot b'_2 + a'_{4,3} \cdot b'_3 + a'_{4,4} \cdot b'_4
\end{aligned}
$$

and multiply with r_{14} to get the conditional likelihood for each site assignment for the specific evolutionary rate category, i.e.

$$
\begin{matrix} r_{13} \\ r_{14} \end{matrix} \xrightarrow{\texttt{\_mm256\_mul\_pd}} r_{12}[c \cdot c', d \cdot d', e \cdot e', f \cdot f']
$$

2.2 Computation of tree likelihood

After computing the conditional likelihood for all evolutionary rates of each state, we are in a position to compute the likelihood of the tree by multiplying the conditional likelihood of each site of the root node with the base frequencies. This process is described by equation 2.

$$
L = \sum_{S_0=A}^{T} \pi_{S_0} L^0_{S_0} \tag{2}
$$

The base frequencies for each nucleotide are given by the vector

$$
\pi = (f_1, f_2, f_3, f_4)
$$

and the conditional likelihoods of the root node for each evolutionary rate and each of the four nucleotide bases are represented by the matrix

$$
L^{(0)} = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix}
$$

where each row represents a specific evolutionary rate.

SSE version We compute the likelihood of the tree by multiplying the conditional likelihoods of each site with the base frequency. We first load the terms of equation 2 for each of the four evolutionary rates in registers r_6, r_7, r_8 and r_9, respectively. The first element of each of those registers is the sum of the corresponding first two terms, and the second element is the sum of the last two terms.

$$
\begin{matrix} b_{1,1} \\ b_{1,2} \end{matrix} \xrightarrow{\texttt{\_mm\_load\_pd}} r_0[b_{1,1}, b_{1,2}]
$$

$$
\begin{matrix} r_0 \\ r_1 \end{matrix} \xrightarrow{\texttt{\_mm\_mul\_pd}} r_2[b_{1,1} \cdot f_1, b_{1,2} \cdot f_2]
$$

$$
\begin{matrix} f_1 \\ f_2 \end{matrix} \xrightarrow{\texttt{\_mm\_load\_pd}} r_1[f_1, f_2]
$$

$$
\begin{matrix} b_{1,3} \\ b_{1,4} \end{matrix} \xrightarrow{\texttt{\_mm\_load\_pd}} r_3[b_{1,3}, b_{1,4}]
$$

$$
\begin{matrix} r_3 \\ r_4 \end{matrix} \xrightarrow{\texttt{\_mm\_mul\_pd}} r_5[b_{1,3} \cdot f_3, b_{1,4} \cdot f_4]
$$

$$
\begin{matrix} f_3 \\ f_4 \end{matrix} \xrightarrow{\texttt{\_mm\_load\_pd}} r_4[f_3, f_4]
$$

$$
\begin{matrix} r_2 \\ r_5 \end{matrix} \xrightarrow{\texttt{\_mm\_hadd\_pd}} r_6[b_{1,1} \cdot f_1 + b_{1,2} \cdot f_2, b_{1,3} \cdot f_3 + b_{1,4} \cdot f_4]
$$

Similarly,

$$r_7[b_{2,1} \cdot f_1 + b_{2,2} \cdot f_2, b_{2,3} \cdot f_3 + b_{2,4} \cdot f_4]$$

$$r_8[b_{3,1} \cdot f_1 + b_{3,2} \cdot f_2, b_{3,3} \cdot f_3 + b_{3,4} \cdot f_4]$$

$$r_9[b_{4,1} \cdot f_1 + b_{4,2} \cdot f_2, b_{4,3} \cdot f_3 + b_{4,4} \cdot f_4]$$

We then perform horizontal adds on the vector registers to store the sum of the first two iterations of equation 2 as the first element of r_{12} and the sum of the last two iterations as the second element.

$$\begin{array}{ll} r_6 & \xrightarrow{\texttt{\_mm\_hadd\_pd}} r_{10}[c, d] \\ r_7 & \\ & \xrightarrow{\texttt{\_mm\_hadd\_pd}} r_{12}[c+d, e+f] \\ r_8 & \xrightarrow{\texttt{\_mm\_hadd\_pd}} r_{11}[e, f] \\ r_9 & \end{array}$$

where

$$\begin{aligned} c &= b_{1,1} \cdot f_1 + b_{1,2} \cdot f_2 + b_{1,3} \cdot f_3 + b_{1,4} \cdot f_4 \\ d &= b_{2,1} \cdot f_1 + b_{2,2} \cdot f_2 + b_{2,3} \cdot f_3 + b_{2,4} \cdot f_4 \\ e &= b_{3,1} \cdot f_1 + b_{3,2} \cdot f_2 + b_{3,3} \cdot f_3 + b_{3,4} \cdot f_4 \\ f &= b_{4,1} \cdot f_1 + b_{4,2} \cdot f_2 + b_{4,3} \cdot f_3 + b_{4,4} \cdot f_4 \end{aligned}$$

Finally, we perform a horizontal add on the same vector to obtain the sum of all iterations of equation 2 as the elements of r_{13}.

$$\begin{array}{ll} r_{12} & \xrightarrow{\texttt{\_mm\_hadd\_pd}} r_{13}[c+d+e+f, c+d+e+f] \\ r_{12} & \end{array}$$

AVX version In a similar manner as in the SSE version, we obtain each of the four terms of the summation as the elements of a four-element AVX vector register. For each evolutionary rate we obtain one such register — r_2, r_4, r_6 and r_8.

$$\begin{array}{lll} b_{1,1}\ b_{1,3} & \xrightarrow{\texttt{\_mm256\_load\_pd}} r_0[b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4}] & \\ b_{1,2}\ b_{1,4} & & \\ & & \xrightarrow{\texttt{\_mm256\_mul\_pd}} r_2[b_{1,1} \cdot f_1, b_{1,2} \cdot f_2, b_{1,3} \cdot f_3, b_{1,4} \cdot f_4] \\ f_1\ f_3 & \xrightarrow{\texttt{\_mm256\_load\_pd}} r_1[f_1, f_2, f_3, f_4] & \\ f_2\ f_4 & & \\ b_{2,1}\ b_{2,3} & \xrightarrow{\texttt{\_mm256\_load\_pd}} r_3[b_{2,1}, b_{2,2}, b_{2,3}, b_{2,4}] & \\ b_{2,2}\ b_{2,4} & & \\ & & \xrightarrow{\texttt{\_mm256\_mul\_pd}} r_4[b_{2,1} \cdot f_1, b_{2,2} \cdot f_2, b_{2,3} \cdot f_3, b_{2,4} \cdot f_4] \\ & r_1 & \\ b_{3,1}\ b_{3,3} & \xrightarrow{\texttt{\_mm256\_load\_pd}} r_5[b_{3,1}, b_{3,2}, b_{3,3}, b_{3,4}] & \\ b_{3,2}\ b_{3,4} & & \\ & & \xrightarrow{\texttt{\_mm256\_mul\_pd}} r_6[b_{3,1} \cdot f_1, b_{3,2} \cdot f_2, b_{3,3} \cdot f_3, b_{3,4} \cdot f_4] \\ & r_1 & \\ b_{4,1}\ b_{4,3} & \xrightarrow{\texttt{\_mm256\_load\_pd}} r_7[b_{4,1}, b_{4,2}, b_{4,3}, b_{4,4}] & \\ b_{4,2}\ b_{4,4} & & \\ & & \xrightarrow{\texttt{\_mm256\_mul\_pd}} r_8[b_{4,1} \cdot f_1, b_{4,2} \cdot f_2, b_{4,3} \cdot f_3, b_{4,4} \cdot f_4] \\ & r_1 & \end{array}$$

The goal is to add all terms of all evolutionary rates together. We start by performing a serie of horizontal adds in order to gain a vector register r_{11}, such that each of its elements is the sum of four (of the sixteen) terms of equation 2. The computation is done in the following way,

$$\begin{array}{l} r_2 \\ r_4 \end{array} \xrightarrow{\texttt{\_mm256\_hadd\_pd}} r_9[c,d,e,f] \xrightarrow{\texttt{\_mm256\_hadd\_pd}} r_{11}[c+d, c'+d', e+f, e'+f'] \\ \begin{array}{l} r_6 \\ r_8 \end{array} \xrightarrow{\texttt{\_mm256\_hadd\_pd}} r_{10}[c',d',e',f']$$

where

$$\begin{array}{ll} c = b_{1,1} \cdot f_1 + b_{1,2} \cdot f_2 & c' = b_{3,1} \cdot f_1 + b_{3,2} \cdot f_2 \\ d = b_{2,1} \cdot f_1 + b_{2,2} \cdot f_2 & d' = b_{4,1} \cdot f_1 + b_{4,2} \cdot f_2 \\ e = b_{1,3} \cdot f_3 + b_{1,4} \cdot f_4 & e' = b_{3,3} \cdot f_3 + b_{3,4} \cdot f_4 \\ f = b_{2,3} \cdot f_3 + b_{2,4} \cdot f_4 & f' = b_{4,3} \cdot f_3 + b_{4,4} \cdot f_4 \end{array}$$

Using a permutation instruction we duplicate r_{11} to a new register r_{12} which has the first two and last two elements of r_{11} swapped between themselves. We add r_{11} and r_{12} together to obtain register r_{13} whose first element is the sum of the terms generated by the first two iterations of the summation part and its second element is the sum of the terms generated by the last two iterations. Elements 3 and 4 are just duplicates of 1 and 2, respectively.

$$\begin{array}{l} r_{11} \\ r_{11} \end{array} \xrightarrow{\texttt{\_mm256\_permute2f128\_pd,0x01}} r_{12}[e+f, e'+f', c+d, c'+d'] \xrightarrow{\texttt{\_mm256\_add\_pd}} r_{13}[p,q,p,q] \\ r_{11}$$

where

$$\begin{array}{l} p = b_{1,1} \cdot f_1 + b_{1,2} \cdot f_2 + b_{2,1} \cdot f_1 + b_{2,2} \cdot f_2 + b_{1,3} \cdot f_3 + b_{1,4} \cdot f_4 + b_{2,3} \cdot f_3 + b_{2,4} \cdot f_4 \\ q = b_{3,1} \cdot f_1 + b_{3,2} \cdot f_2 + b_{4,1} \cdot f_1 + b_{4,2} \cdot f_2 + b_{3,3} \cdot f_3 + b_{3,4} \cdot f_4 + b_{4,3} \cdot f_3 + b_{4,4} \cdot f_4 \end{array}$$

Finally, we perform a horizontal add on r_{13} and a copy of it to obtain a register whose every element holds the value of equation 2

$$\begin{array}{l} r_{13} \\ r_{13} \end{array} \xrightarrow{\texttt{\_mm256\_hadd\_pd}} r_{14}[p+q, p+q, p+q, p+q]$$

2.3 Parallelization using OpenMP

We use OpenMP for parallelizing the likelihood computation in DPPDIV. Specifically, we parallelise the for loop that cycles through all sites of a particular node and computes the conditional likelihood for each site. The conditional likelihood computation part iterates through all internal nodes, and computes the conditional likelihood for each site for a specific evolutionary rate. For the computation of the conditional likelihood of an internal node, it is necessary that the conditional likelihoods of its descendants must have already been computed. Thus, for a specific internal node, we paralelise the inner loop which processes each site of for a specific evolutionary rate. For this embarrassingly parallel workload, we use the OpenMP `#pragma omp parallel for` directive.

Similarly, concerning the computation of the tree likelihood, we paralelize the loop that processes each site of the root node using a sum reduction with the directive `#pragma omp parallel for reduction ( + : lnL )`, where `lnL` is the variable where the sum of likelihoods of all sites is ultimately stored in.

3 Experimental results

We demonstrate the performance of the SSE optimized parallel implementation compared to the unoptimised original implementation and its parallel enhancement in Table 1. The experiments were conducted on a 48-core AMD Opteron processor machine with 256GB of memory. Our input data consists of seven species and the alignment size is 448 162 bp long and we run 100 000 markov chain monte carlo (MCMC) cycles. The measured times are plotted in Figure 1.

	Number of threads						
Optimization	1	8	16	24	32	40	48
None	306m35s	39m53s	28m10s	27m32s	25m45s	26m15s	35m55s
SSE	196m55s	29m16s	22m44s	23m2s	20m34s	24m41s	26m14s

Table 1. Experimental results

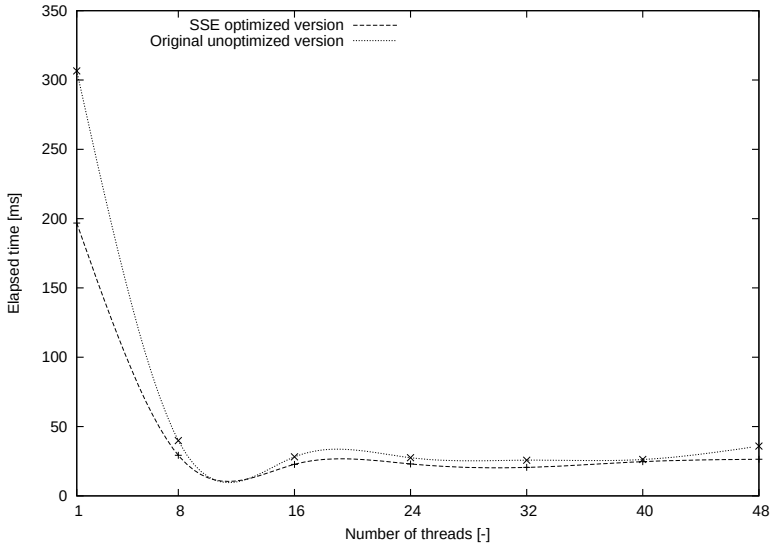

Fig. 1. Performance comparison between SSE and non-optimized version

4 Conclusion

In this report we have made modifications that significantly speed up likelihood computation and hence improve the runtime speed of DPPDIV. We have written two architecture-dependent versions of the likelihood computation, one which uses the SSE3 instruction set and one which uses the AVX instruction set. On top of that, we have encapsulated the loops processing the sites of a specific code in an OpenMP parallel for region which seems to scale fine. With the conducted experiments we have observed that the parallel version scaling is satisfactory and noticed a 30% percent improvement on the single-thread SSE and AVX specific versions. Although we expected the AVX version to be faster than the SSE version by taking in account the official Intel documentation describing the processor specific instruction clock cycles, the experiments show that the SSE and AVX versions perform in the same time, with the surprising fact that SSE is sometimes faster than AVX. This may be due to the fact that the 256-bit data move instructions limit the data throughput.

References

1. Felsenstein, J.: Evolutionary trees from dna sequences: a maximum likelihood approach. Journal of Molecular Evolution 17(6), 368–376 (1981)
2. Gallager, R.G.: Low-density parity-check codes. MIT Press (1963)
3. Heath, T.A., Holder, M.T., Huelsenbeck, J.P.: A dirichlet process prior for estimating lineage-specific substitution rates. Molecular Biology and Evolution 29, 939–955 (2012), http://mbe.oxfordjournals.org/content/early/2011/12/14/molbev.msr255.abstract
4. Huelsenbeck, J.P., Suchard, M.A.: A Nonparametric Method for Accommodating and Testing Across-Site Rate Variation. Systematic Biology 56(6), 975–987 (Dec 2007)
5. Neal, R.M.: Markov chain sampling methods for Dirichlet process mixture models. Journal of Computational and Graphical Statistics 9(2), 249–265 (2000)