Title
Massive Parallelization Boosts Big Bayesian Multidimensional Scaling

Permalink
https://escholarship.org/uc/item/0wv8d79z

Journal
Journal of Computational and Graphical Statistics, 30(1)

ISSN
1061-8600

Authors
Holbrook, Andrew J
Lemey, Philippe
Baele, Guy
et al.

Publication Date
2021-01-02

DOI
10.1080/10618600.2020.1754226

Peer reviewed

Published in final edited form as:

J Comput Graph Stat. 2021 ; 30(1): 11–24. doi:10.1080/10618600.2020.1754226.

Massive parallelization boosts big Bayesian multidimensional scaling

Andrew J. Holbrook[1], **Philippe Lemey**[2], **Guy Baele**[2], **Simon Dellicour**[2], **Dirk Brockmann**[3], **Andrew Rambaut**[4,5], **Marc A. Suchard**[1,6,7]

[1]Department of Biostatistics, University of California, Los Angeles

[2]Department of Microbiology, Immunology and Transplantation, Rega Institute, KU Leuven

[3]Institute for Theoretical Biology, Humboldt University Berlin

[4]Institute of Evolutionary Biology, University of Edinburgh

[5]Fogarty International Center, National Institutes of Health

[6]Department of Human Genetics, University of California, Los Angeles

[7]Department of Biomathematics, University of California, Los Angeles

Abstract

Big Bayes is the computationally intensive co-application of big data and large, expressive Bayesian models for the analysis of complex phenomena in scientific inference and statistical learning. Standing as an example, Bayesian multidimensional scaling (MDS) can help scientists learn viral trajectories through space-time, but its computational burden prevents its wider use. Crucial MDS model calculations scale quadratically in the number of observations. We partially mitigate this limitation through massive parallelization using multi-core central processing units, instruction-level vectorization and graphics processing units (GPUs). Fitting the MDS model using Hamiltonian Monte Carlo, GPUs can deliver more than 100-fold speedups over serial calculations and thus extend Bayesian MDS to a big data setting. To illustrate, we employ Bayesian MDS to infer the rate at which different seasonal influenza virus subtypes use worldwide air traffic to spread around the globe. We examine 5392 viral sequences and their associated 14 million pairwise distances arising from the number of commercial airline seats per year between viral sampling locations. To adjust for shared evolutionary history of the viruses, we implement a phylogenetic extension to the MDS model and learn that subtype H3N2 spreads most effectively, consistent with its epidemic success relative to other seasonal influenza subtypes. Finally, we provide MassiveMDS, an open-source, stand-alone C++ library and rudimentary R package, and discuss program design and high-level implementation with an emphasis on important aspects of computing architecture that become relevant at scale.

Keywords

Massive parallelization; GPU; SIMD; Hamiltonian Monte Carlo; Bayesian phylogeography

Corresponding author Andrew J. Holbrook aholbroo@uci.edu.

1 Introduction

The integral that so often arises from the application of Bayes' theorem exacerbates the general difficulties of big data statistical inference. Bayesian statisticians use the language of conditional probability to piece together large, flexible models with ease. Perversely, the integral and its myriad workarounds become less feasible as model complexity grows. Brute force computing rarely suffices for big Bayesian inference, the doubly dangerous collision between big data and massive model. As a field, Bayesian phylogeography illustrates the challenges presented by big Bayes and the practical payoff in overcoming these challenges.

The speed of transport in the global economy is matched by the complexity of travel patterns, which affect the emergence and spread of pathogens (Bloom et al., 2017). Scientists, epidemiologists, and policymakers must quickly visualize, draw actionable conclusions and make predictions from huge swaths of viral sequence data collected from all around the world. Although various approaches exist to study the spread of infectious diseases, recent developments for rapidly evolving pathogens take a probabilistic perspective simultaneously on the spatiotemporal spread and pathogen sequence mutation process. Here, phylogenetic diffusion models represent relatively simple and computationally efficient yet flexible tools to connect spatial dynamics to sequence evolution, and specific implementations of random walk models are available for both discrete and continuous location data for the sampled sequences (Lemey et al., 2009, 2010).

The discrete approach models transitioning between a limited set of discrete states throughout the ancestral history of the pathogen sequences according to a continuous-time Markov chain process and has facilitated the study of global movement patterns of influenza viruses, with most efforts for human flu focusing on influenza A/H3N2 (Bahl et al., 2011; Nelson et al., 2015). A recent study has broadened the focus to all four seasonal influenza viruses, including two influenza A subtypes (H3N2 and H1N1) and two influenza B subtypes (Yamagata and Victoria) (Bedford et al., 2015). This study shows the viruses varying in their degree of persistence and discretized-location switching frequency, with A/H1N1 evincing lower switching rates compared to A/H3N2 and the two B subtypes showing even lower rates. The study associates these differences with how quickly the viruses evolve antigenically. A key element in the connection between antigenic drift and global movement are differences in age distributions of infection and age-specific mobility patterns: viruses capable of evolving faster antigenically will be better at infecting adults, who tend to travel more frequently than children, providing more opportunities for the virus to spread (Bedford et al., 2015).

Discrete phylogeographic reconstructions have important limitations such as their sensitivity to sampling biases and the need to specify arbitrary spatial partitions. The *continuous* diffusion model offers an interesting alternative in this respect, but geographic space is ill-suited for tracking pathogens in humans and other hosts that frequently travel long distances. However, a recent modeling study has demonstrated that complex spatiotemporal patterns of spread evince surprisingly simple and regular wave-like patterns for distances measured along transport networks instead of spatial distances (Brockmann and Helbing, 2013).

We use this concept of ‘effective distance’ to perform phylogeographic inference in latent effective space by adopting a Bayesian MDS approach that enables the quantification and comparison of differences in the rate at which the different seasonal influenza variants travel the air transportation network. Bayesian MDS (Oh and Raftery, 2001) probabilistically projects relationships between high dimensional objects onto low-dimensional space and thus accounts for uncertainty by integrating over constellations of latent locations. To adjust for shared evolutionary history between viral samples, Bedford et al. (2014) combines Bayesian MDS with a latent phylogenetic diffusion model. With pairwise distances between flu samples arising from biochemical assays, those authors used their Bayesian phylogenetic MDS model to draw insights into the changing antigenicity of the four major seasonal influenza subtypes.

Unfortunately, the Markov chain Monte Carlo (MCMC) computations required for a Bayesian MDS (phylogenetic or otherwise) analysis are onerous. The MDS likelihood prevents Gibbs sampling, and complexity of the likelihood evaluations necessary for Metropolis-Hastings grows quadratically with the data. Fosdick et al. (2019) circumvent MCMC computations by fitting the MDS model with an optimization routine. Bedford et al. (2014) partly avoid the issue by completely leaving out the truncation term on the non-negative pairwise distances—the computational bottleneck of the likelihood evaluation—and effectively draw inference based on an incorrect model. Those authors witness high autocorrelation between Markov chain states regardless. Hamiltonian Monte Carlo (HMC) (Neal, 2011), an advanced MCMC algorithm that uses gradient information to craft proposals, could help improve this poor mixing, but the log-likelihood gradient evaluations required by HMC also scale quadratically with the data.

Nonetheless, we assert that correct inference from Bayesian MDS is possible, even scalable, with the help of massively parallel computing that we exploit here to quickly calculate the MDS likelihood and log-likelihood gradient in the context of HMC. We are not the first to investigate parallel implementations in statistical computing: Suchard and Rambaut (2009), Suchard et al. (2010a) and Suchard et al. (2010b) apply graphics processing unit (GPU) computing to optimization and Bayesian inference for phylogenetics and flow cytometry; Lee et al. (2010) perform sequential Monte Carlo with the aid of GPUs; Zhou et al. (2010) leverage GPUs for statistical optimization; and Beam et al. (2016) compute the likelihood and its gradient for a multinomial model with GPUs and thus accelerate HMC for that model.

While GPUs also deliver the greatest speed gains for the inference problem considered here, we find that multi-core central processing units (CPUs) combined with on-chip vectorization follow closely behind in scaling Bayesian MDS for millions of data points. To facilitate adoption of both GPU- and CPU-based parallel computing for Bayesian MDS, we provide the open-source library MassiveMDS http://github.com/suchard-group/MassiveMDS both as an R package and as a stand-alone C++ library. Section 2.6 contains further information on MassiveMDS and related software that we provide readily available online. We now introduce Bayesian MDS and its phylogenetic instantiation.

2 Methods

2.1 From dissimilarity to a latent space

Multidimensional scaling (MDS) encompasses a class of ordination methods that project a collection of objects into a low-dimensional Euclidean space based on dissimilarity measurements between pairs of objects (Kruskal, 1964). Through MDS, objects with smaller dissimilarity generally find themselves nearer in L_2 distance to each other in this latent space than objects with larger dissimilarity. While MDS traditionally has found use as an exploratory data analysis tools, model-based MDS variants exist within the Bayesian framework (DeSarbo et al., 1998; Oh and Raftery, 2001). One posits that each object's latent location is a random variable, translates the MDS projection into a probability model on the observed dissimilarities given distances between latent locations (Ramsay, 1982) and specifies an appropriate prior distribution over these locations. Previously, such distributions have remained arbitrary and relatively uninformative. In this paper, however, the stochastic process that gives rise to the latent location prior distribution is highly-structured and well-informed. Further, the parameters that characterize the process are of chief scientific interest.

We are interested in a finite collection of N items. For any two distinct items i and j we follow Oh and Raftery (2001) and model the observed dissimilarity y_{ij} as conditionally independent, normal random variables, truncated to be positive

$$y_{ij} \sim \mathrm{N}(\delta_{ij}, \sigma^2)\mathrm{I}(y_{ij} > 0) \text{ for } i > j, \tag{1}$$

where the expected dissimilarity $\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ is the L_2 norm between latent locations $\mathbf{x}_i = (x_{i1}, \ldots, x_{iD})^t$ and $\mathbf{x}_j = (x_{j1}, \ldots, x_{jD})^t$ in a low-dimensional, real coordinate space $\mathbb{R}^D$. Given all latent locations $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^t$, the conditional density of the observed data $\mathbf{Y}$ becomes

$$\begin{aligned} p(\mathbf{Y} \mid \mathbf{X}, \sigma^2) &\propto (\sigma^2)^{\frac{N(1-N)}{4}} \exp\left(-\sum_{i>j} r_{ij}\right) \\ r_{ij} &= \frac{(y_{ij} - \delta_{ij})^2}{2\sigma^2} + \log\Phi\left(\frac{\delta_{ij}}{\sigma}\right), \end{aligned} \tag{2}$$

where $\Phi(\cdot)$ is the cumulative distribution function of a standard normal random variable.

Motivation behind this probabilistic transformation from observed dissimilarity into a Euclidean space rests on present limitations in drawing inference about diffusive processes (such as those outlined below) over irregular landscapes (Billera et al., 2001). Inference over irregular landscapes often necessitates extensive data augmentation or numerical integration (Manton, 2013; Nye and White, 2014) to closely approximate the density function of a partially observed sample-path, as the density function is a solution of the stochastic differential equation that governs the diffusion. On the other hand, in a latent Euclidean space, simple Brownian diffusion (Brown, 1828; Wiener, 1958) and its scale mixtures generalization (Lemey et al., 2010) offer closed-form density functions. Further, these functions yield conveniently to the analytic integration necessary to track the multiple end-state locations X of dependent diffusion processes.

2.2 Highly-structured Brownian process prior

Consider a collection of M molecular sequence alignments, where each alignment $\mathbf{S}_m$ for $m = 1, \ldots, M$ contains sequences from N_m evolutionarily related viruses, N_e additional unsequenced viruses from unsampled locations and an $N \times N$ symmetric, dissimilarity matrix $\mathbf{Y}$ for $N = \sum_{m=1}^{M} N_m + N_e$, where in general $N \gg N_e$. Each entry y_{ij} of $\mathbf{Y}$ reports a non-negative dissimilarity measurement between virus i and j for $i, j = 1, \ldots, N$. We follow standard Bayesian phylogenetics hierarchical approaches (Suchard et al., 2003) to model the sequence data $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_M)$ that include, among other parameters ϕ not critical to the development in this paper, phylogenetic trees $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_M)$ which may be known *a prion* or random. Each tree $\mathcal{G}_m$ is a bifurcating, directed graph with N_m terminal degree-1 nodes $(v_1^m, \ldots, v_{N_m}^m)$ that correspond to the tips of the tree, $N_m - 2$ internal degree-3 nodes $(v_{N_m+1}^m + \cdots, v_{2N_m-2}^m)$, a root degree-2 node $v_{2N_m-1}^m$ edge weights $(t_1^m, \ldots, t_{2N_m-2}^m)$ that report the elapsed evolutionary time between nodes. To simplify notation later, let $\mathcal{G}_{(i)} \in \{1, \ldots, M\}$ indicate the alignment to which virus i belongs, with 0 indicating i unsampled. We assume conditional independence between $\mathbf{S}$ and $\mathbf{Y}$ given $\mathcal{G}$. Interested readers may explore, for example, Suchard et al. (2001) or Suchard et al. (2018) for detailed development of $p(\mathbf{S}, \phi, \mathcal{G})$.

Our model posits that a multivariate Brownian diffusion process along the branches of the trees in $\mathcal{G}$ (Lemey et al., 2010) gives rise to $\mathbf{X}$. The Brownian process asserts that the latent location value of a child node v_c^m in tree $\mathcal{G}_m$ is multivariate normally distributed about the latent value of its parent node $V_{\mathrm{pa}(c)}^m$ with variance $t_c^m \times \mathbf{\Sigma}$. The unknown $K \times K$ matrix $\mathbf{\Sigma}$ parameterizes the dispersal rate in the latent space after controlling for correlation in latent values that are shared by descent through $\mathcal{G}_m$. This construction generalizes univariate Comparative Methods (Cavalli-Sforza and Edwards, 1967; Felsenstein, 1985) approaches to model the evolution of continuous-valued random variables first into a multivariate setting and second across multiple trees. Figure 1 illustrates one possible realization of this process for a single tree with $N_m = 4$ tips.

We assume that the latent values at all M root nodes $v_{2N_m-1}^m$ and for the N_e unsequenced viruses are *a priori* multivariate normally distributed with mean $\boldsymbol{\mu}_0$ and variance $\tau_0 \times \mathbf{\Sigma}$ or $\tau_e \times \mathbf{\Sigma}$, respectively. Following Cybis et al. (2015), we can ascribe that jointly $\mathbf{X}$ is matrix normally distributed, with probability density function

$$p(\mathbf{X} | \mathbf{V}_{\mathcal{G}}, \mathbf{\Sigma}, \boldsymbol{\mu}_0, \tau_0, \tau_{\mathrm{e}}) = \frac{\exp\left\{-\frac{1}{2}\mathrm{tr}\left[\mathbf{\Sigma}^{-1}(\mathbf{X} - \boldsymbol{\mu}_0)^t \mathbf{V}_{\mathcal{G}}^{-1}(\mathbf{X} - \boldsymbol{\mu}_0)\right]\right\}}{(2\pi)^{ND/2} |\mathbf{\Sigma}|^{N/2} |\mathbf{V}_{\mathcal{G}}|^{D/2}}, \quad (3)$$

where $\mathbf{V}_{\mathscr{G}} = \{v_{ij}\}$ is a block-diagonal $N \times N$ matrix. Specifically, if virus i is unsequenced, then $v_{ij} = \tau_e$ and $v_{ij} = 0$ for all $j \neq i$. The trees in $\mathscr{G}$ define the remaining M blocks. We define $d_F(u, w)$ to equal the edge-weight sum along the shortest path between node u and node w in a tree $\mathscr{G}_m$. Then, within block m, diagonal elements $v_{i'i'} = \tau_0 + d_F(v^m_{2N_m-1}, v_{i'})$, the elapsed evolutionary time between the root node and tip node i', and off-diagonal elements $v_{i'j'} = \tau_0 + \left[d_F(v^m_{2N_m-1}, v_{i'}) + d_F(v^m_{2N_m-1}, v_{i'}) - d_F(v_{i'}, v_{j'})\right]/2$, the elapsed time between the root and the most recent common ancestor of tip nodes i' and j'.

To complete our model specification, we assume *a priori*

$$\begin{aligned} &\mathbf{\Sigma}^{-1} \sim \text{Wishart}(d_0, \mathbf{T}_0) \text{ and} \\ &\sigma^{-2} \sim \text{Gamma}(s_0, r_0), \end{aligned} \tag{4}$$

with degrees of freedom d_0, rate matrix $\mathbf{T}_0$ shape s_0 and rate r_0. Finally, we specify fixed hyperparameters $(\boldsymbol{\mu}_0, \tau_0, d_0, \mathbf{T}_0, s_0, r_0)$ in our example.

2.3 Inference

We use Markov chain Monte Carlo (MCMC) to learn the posterior distribution

$$\begin{aligned} &p(\mathbf{\Sigma}, \sigma^2, \mathscr{G}, \phi \mid \mathbf{Y}, \mathbf{s}) \propto p(\mathbf{Y} \mid \mathbf{\Sigma}, \sigma^2, \mathscr{G}) \times p(\mathbf{\Sigma}) \times p(\sigma^2) \times p(\mathbf{S}, \phi, \mathscr{G}) \\ &= (\int p(\mathbf{Y} \mid \mathbf{X}, \sigma^2) p(\mathbf{X} \mid \mathbf{\Sigma}, \mathscr{G}) \mathrm{d}\mathbf{X}) \times p(\mathbf{\Sigma}) \times p(\sigma^2) \times p(\mathbf{S}, \phi, \mathscr{G}) \end{aligned} \tag{5}$$

with a random-scan Metropolis-with-Gibbs scheme and the development of a computationally efficient transition kernel to sample the latent values $\mathbf{X}$. We exploit standard Bayesian phylogenetic algorithms (Suchard et al., 2018) based on Metropolis-Hastings sampling for the tree $\mathscr{G}$ and other phylogenetic parameters ϕ. These latter transition kernels are not rate-limiting.

Sampling $\mathscr{G}$ necessitates evaluating $p(\mathbf{X} \mid \mathbf{\Sigma}, \mathscr{G})$. Equation (3) suggests a computational order $\mathscr{O}(\widetilde{N}^3)$ where $\widetilde{N} = \max_m N_m$ to form the matrix inverse $\mathbf{V}_{\mathscr{G}}^{-1}$. However, we follow Pybus et al. (2012) who develop a dynamic programming algorithm to evaluate Equation (3) in $\mathscr{O}(\widetilde{N} D^2)$ via parallelizable post-order traversals of the trees in $\mathscr{G}$. Freckleton (2012) and Ho and Ané (2014) propose similar linear-time algorithms, but the underlying idea of message passing on a directed, acyclic graph extends back at least to Cavalli-Sforza and Edwards (1967) and Pearl (1982). Sampling ϕ is, likewise, linear in $\widetilde{N}$ and also conveniently computable on massively parallel devices (Suchard and Rambaut, 2009).

On the other hand, sampling $\mathbf{X}$ stands as the rate-limiting operation for posterior inference. To appreciate why, the full conditional distribution $p(\mathbf{X} \mid \mathbf{Y}, \mathscr{G}, \mathbf{\Sigma}, \sigma^2)$ is not of standard form, eliminating Gibbs sampling. Any sampler for $\mathbf{X}$ must therefore evaluate or approximate $\log p(\mathbf{Y} \mid \mathbf{X}, \sigma^2)$ or its derivatives. This log-density is a sum over $N(N-1)/2$ terms, where each involves $\mathscr{O}(D)$ operations, such that complete evaluation is $\mathscr{O}(N^2 D)$. Its gradient is N

summations over $N - 1$ terms that also involve $\mathcal{O}(D)$ operations. Again, complete evaluation is $\mathcal{O}(N^2 D)$.

Bedford et al. (2014) describe a previous attempt at fitting a related Bayesian MDS model with a phylogenetically informed prior. Restricted to a modest $N \sim \text{low } 100\text{s}$ similar to the problems that Oh and Raftery (2001) attack, this previous work has relied on a low-dimensional random-walk transition kernel applied at random to a single element $\mathbf{X}_i$ or even x_{ik} of $\mathbf{X}$. The advantage here exploits caching the $\mathcal{O}(N^2)$ terms, since changing the value of a single $\mathbf{x}_i$ invalidates only $N - 1$ terms. The disadvantage lies in the potentially extreme auto-correlation of the MCMC sample that grows with increasing NThe auto-correlation arises out of the posterior dependence between $\mathbf{x}_i$ and for $\mathbf{x}_j$ that our highly-informative phylogenetic prior $p(\mathbf{X} \mid \boldsymbol{\Sigma}, \mathscr{G})$ exacerbates.

Hamiltonian Monte Carlo—We rely on HMC (Duane et al., 1987; Neal, 2011) to efficiently simulate from the posterior distribution with respect to latent data $\mathbf{X}$. HMC is an advanced MCMC methodology that employs deterministic Hamiltonian dynamics to intelligently generate proposal states. We couple each trajectory with a Metropolis accept-reject step that renders the target distribution invariant. The upshot is an algorithm that has aided Bayesian analysis by facilitating posterior inference for models of unprecedented dimension and hierarchical structure.

Specifically, let $\pi(\mathbf{x})$ be the probability density function of the target distribution for generic random variable $\mathbf{x}$ with continuous support. Further, assume that $\pi(\cdot)$ be differentiable, as is the case for the application considered here. HMC works by augmenting the 'position' variable $\mathbf{x}$ by an independent, auxiliary 'momentum' variable $\mathbf{p}$ with Gaussian density $\xi(\mathbf{p})$. An energy function is constructed as the negative logarithm of the density for the joint distribution over $(\mathbf{x}, \mathbf{p})$:

$$H(\mathbf{x}, \mathbf{p}) = -\log(\pi(\mathbf{x})\xi(\mathbf{p})) \propto -\log \pi(\mathbf{x}) + \frac{1}{2}\mathbf{p}^T \mathbf{M}^{-1} \mathbf{p} \,. \tag{6}$$

Here, $\mathbf{M}$ is the covariance of $\mathbf{p}$, but it is also interpretable as the mass matrix for the dynamical system associated to Hamiltonian $H(\mathbf{x}, \mathbf{p})$ and described by the system of equations

$$\begin{aligned} \mathbf{x} &= \frac{\partial}{\partial \mathbf{p}} H(\mathbf{x}, \mathbf{p}) = \frac{1}{2}\mathbf{M}^{-1}\mathbf{p} \\ \mathbf{p} &= -\frac{\partial}{\partial \mathbf{x}} H(\mathbf{x}, \mathbf{p}) = \nabla \log \pi(\mathbf{x}) \,. \end{aligned} \tag{7}$$

By Liouville's theorem, Hamiltonian dynamics conserve the energy $H(\mathbf{x}, \mathbf{p})$. It is a corollary that perfect simulation of the Hamiltonian system is equivalent to perfect sampling from the canonical distribution $\exp(-H(\mathbf{x}, \mathbf{p})) = \pi(\mathbf{x})\xi(\mathbf{p})$, i.e., for any stopping time t, the Metropolis-Hastings acceptance criterion is

$$\min\left(1, \frac{\exp(-H(\mathbf{x}_t, \mathbf{p}_t))}{\exp(-H(\mathbf{x}_0, \mathbf{p}_0))}\right) = \min(1, 1) = 1. \tag{8}$$

Exact simulation is rarely available, but the discretized leapfrog algorithm (Leimkuhler and Reich, 2004) has proven effective for simulating the Hamiltonian dynamics. The discretization *does* lead to errors which cause the Hamiltonian evaluated at the proposed state $(\mathbf{x}^*, \mathbf{p}^*)$ to differ from that of the current state, so an accept-reject step remains necessary. On the other hand, the discretized dynamics preserve volume just like the true Hamiltonian dynamics, so we need no Jacobian corrections to calculate the Metropolis acceptance criterion. We do, however, need to evaluate the log-likelihood gradient at multiple points along the trajectory.

The log-likelihood gradient—In the notation for Bayesian MDS, HMC requires the gradient of the log-likelihood with respect to latent locations $\mathbf{X}$. The gradient of the log-likelihood with respect to a single row $\mathbf{x}_i$ of the matrix $\mathbf{X}$ is

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{x}_i} \log p(\mathbf{Y} \mid \mathbf{X}, \sigma^2) = \frac{\partial}{\partial \delta_{ij}} \log p(\mathbf{Y} \mid \mathbf{X}, \sigma^2) \frac{\partial \delta_{ij}}{\partial \mathbf{x}_i} \\
&= -\sum_{j \neq i} \left(\frac{(\delta_{ij} - y_{ij})}{\sigma^2} + \frac{\phi(\delta_{ij}/\sigma)}{\sigma \Phi(\delta_{ij}/\sigma)} \right) \frac{\partial \delta_{ij}}{\partial \mathbf{x}_i} \\
&= -\sum_{j \neq i} \left(\frac{(\delta_{ij} - y_{ij})}{\sigma^2} + \frac{\phi(\delta_{ij}/\sigma)}{\sigma \Phi(\delta_{ij}/\sigma)} \right) \frac{(\mathbf{x}_i - \mathbf{x}_j)}{\delta_{ij}} \\
&:= -\sum_{j \neq i} \mathbf{r}_{ij} .
\end{aligned} \tag{9}$$

Here, $\phi(\cdot)$ is the probability density function of a standard normal variate, and $\mathbf{r}_{ij}$ is the contribution of the *j*th location to the gradient with respect to the *i*th location.

2.4 Model selection

We use cross-validation when it is difficult *a priori* to motivate a modeling decision. In particular, we are interested in judging the ‘correct’ number of dimensions available for the latent diffusion process. In turn, we interpret this latent dimensionality as a rough quantification of the complexity for the air-traffic space in junction with the pathogen’s evolutionary dynamics.

In cross-validation (Geisser, 1975), the practitioner systematically excludes a fixed proportion of observations, trains the model on those remaining observations and uses the fitted model to predict the held-out data. We repeat this process over a number of ‘folds’ and ascertain the prediction error over the different folds. The model with the smallest total prediction error is deemed best.

Recall that, for MDS, our data is a large distance matrix $\mathbf{Y}$, and observations correspond to off-diagonal elements y_{ij}. A cross-validation fold consists of the held-out observations $\mathbf{Y}^{f}_{IJ}$

and the remaining observations $\mathbf{Y}^{f}_{-(IJ)}$ for some multi-index IJ depending on f. Let s index an MCMC state corresponding to a single draw from the target posterior and denote the complete set of latent locations and model parameters $(\mathbf{X}, \Theta)^{f}_{s}$ for $s = 1, \ldots, S^{f}$. We use the empirical *log pointwise predictive density* (lpd) as measure of predictive accuracy and model fit (Vehtari et al., 2017). The log pointwise predictive density is

$$
\begin{aligned}
lpd &= \sum_{f} \sum_{\substack{i<j \\ i,j \in IJ}} \log p(y^{f}_{ij} \mid \mathbf{Y}^{f}_{-(IJ)}) \\
&= \sum_{f} \sum_{\substack{i<j \\ i,j \in IJ}} \log \int p(y^{f}_{ij} \mid (\mathbf{X}, \Theta)) p((\mathbf{X}, \Theta) \mid \mathbf{Y}^{f}_{-(IJ)}) \mathrm{d}(\mathbf{X}, \Theta) \\
&\approx \sum_{f} \sum_{\substack{i<j \\ i,j \in IJ}} \log \frac{1}{S^{f}} \sum_{s=1}^{S^{f}} p(y^{f}_{ij} \mid (\mathbf{X}, \Theta)^{f}_{s}) p((\mathbf{X}, \Theta)^{f}_{s} \mid \mathbf{Y}^{f}_{-(IJ)}) = \widehat{lpd}\,.
\end{aligned}
$$

Given two models with differing latent dimensions, we choose the model with smaller $\widehat{lpd}$. For more on Bayesian model selection, see Gelman et al. (2013).

2.5 Massive parallelization

HMC is a powerful tool for Bayesian learning, but the likelihood and gradient evaluations it necessitates become overly burdensome for Big Data inference. Efficiently computing $\log p\left(\mathbf{Y} \mid \mathbf{X}, \sigma^{2}\right)$ and its gradient with respect to $\mathbf{X}$ remains a critical and rate-limiting step at $\mathcal{O}(N^{2}D)$ for each evaluation. When N is large, the key insights for effective parallelization of these evaluations are three-fold.

The first is most important: for the likelihood, there are a massive number of stereotyped, and seemingly independent, operations in evaluating r_{ij} for all $i > j$ (a transformation); for the gradient, there are a massive number of stereotyped, and seemingly independent, operations in evaluating $\mathbf{r}_{ij}$ for all $i>j$ (again, a transformation). For both r_{ij} and $\mathbf{r}_{ij}$, the floating point operations required to evaluate $\Phi(\cdot)$ are rate-limiting, so a useful computing strategy limits wall time spent performing these operations by applying the functions in parallel across a range of inputs.

The final two insights are often missed in statistical computing. The above transformations offer a high degree of data-reuse; for example, r_{ij} and $\mathbf{r}_{ij}$ for all j depend on $\mathbf{x}_i$ and y_{ij} and r_{ij} and $\mathbf{r}_{ij}$ for all i depend on $\mathbf{x}_j$ and y_{ij}. A good computing strategy stores these values in a way that facilitates fast reuse. Finally, the realized values of r_{ij} and $\mathbf{r}_{ij}$ are never actually needed, only their sum (a reduction). A good strategy avoids storing intermediate values in costly memory.

Parallelization strategies—The simultaneous execution of multiple mathematical operations through parallelization continues to enable computation to keep pace with Moore's Law that posits processing power doubles approximately every two years.

Parallelization is growing into a dominant theme in large-scale statistical inference (Suchard et al., 2010b) and on hardware including clusters of independent compute nodes, on-chip vector instructions, multithreaded multi-core processors and parallel co-processors such as GPUs. Capitalizing on these hardware features in software implementation could emerge as the most important task facing a computational statistician.

Cluster computing is the most familiar form of parallelization and often scales up to 1000s or more of nodes that contain their own CPUs and random access memory (RAM), linked loosely together through an Ethernet or InfiniBand network. In total, 1000s of CPUs and massive quantities of RAM can comprise a single cluster; however, sharing information between individual nodes has high latency in these distributed-computing environments. Communication latency can severely affect the theoretically achievable speed-up of parallelization.

Assume a computing task has an overall execution cost c_0 and can be evenly distributed across S parallel devices, each executing a thread of work. Then, an ideal implementation will still require actual time $c_0/S + c_1$, where c_1 is the incurred time of communication or additional non-parallelizable computation. The speed-up on this ideal system often falls far short of S-fold. Further, the overhead parameter c_1 often scales with S. The high latency of clusters, therefore, suggests coarse-grain decomposition with larger and relatively independent threads assigned to each compute node to minimize the parallelization overhead. Further, many statistical model fitting algorithms, including high-dimensional optimization and MCMC, are iterative such that ci also scales by the number of iterations. While several tools stand out for cluster-based statistical computing (Schmidberger et al., 2009), we often find that significant financial investment in purchasing or renting a large cluster yields only modest speed-up. For example, Suchard et al. (2010a) benchmark a related MCMC inference problem involving millions of observations, yielding an approximate 20-fold speed-up on a 100-node cluster costing about $250,000 in 2008, or about $3,600 for a month of compute-time today on a cloud-based service.

In this paper, we avoid the high latency cost of iterative algorithms on clusters and instead raise awareness on exploiting parallelization in less expensive shared-computing environments. Specifically, we explore two prominent and one often unrecognized avenues of parallelization to evaluate $\log p(\mathbf{Y} \mid \mathbf{X}, \sigma^2)$ (Equation (2)) and its gradient with respect to $\mathbf{X}$ (Equation (9)), our rate-limiting steps in inference. On the prominent side, these avenues harness the multiple, multithreaded processing cores cast onto CPUs in standard desktop computers and the extreme number of cores working in tandem on GPU cards now ubiquitous as high-performance computing add-ons. Under appreciated stands single-instruction, multiple-data (SIMD) integer or floating-point operation parallelization available within standard CPU cores.

In the parlance of high-performance computing, evaluating Equation (2) is a transformation-reduction, where the transformation reads from RAM the latent locations $\mathbf{X}$ and observed distances $\mathbf{Y}$ and computes r_{ij} for all $i > j$ and the reduction then sums together all r_{ij}. A first glance at this computation suggests two separate loops over all $i > j$; the first presents $\mathcal{O}(N^2)$ embarrassingly parallel tasks with floating point math calls, and the second loop has strong serial dependence. Gradient evaluation requires the N-transformation-reductions given by Equation (9) for each i. Again, each transformation-reduction suggests two separate loops over all $i > j$; here, the inner loop is $\mathcal{O}(N)$ with floating point math operations, and the outer loop again has strong serial dependence.

Rate-limiting are the embarrassingly parallel floating point operations characterized by math function calls. On account of their heavy cost, it is advantageous to broadcast such expensive operations across as broad a swath of input data as possible. GPUs efficiently apply operations to thousands of inputs, and, with the help of SIMD, multi-core CPUs simultaneously apply a single floating operation to independent inputs in the low hundreds. Parallel computing of floating point operations is much faster than serial, but an important caveat of high-performance computing is that a single memory transaction involving a read or write from RAM may take up to two orders-of-magnitude more time than a numerical operation applied to a value sitting in a limited number of storage locations called registers within the CPU or GPU. In practice, multiple consecutive memory transactions occur simultaneously, moving a block of data between RAM and successive layers of high-speed memory in close proximity to the processor called caches. As a result, consecutive memory transactions are considerably faster than random ones, but still pale in speed comparison to on-chip numerical operations.

To avoid unnecessary memory transactions, it should become common practice in statistical computing to “fuse” together performance-dependent transformation-reductions into single loops even if they retain some serial dependence. Through fusion, for example, intermediate r_{ij} values never need to be written to RAM in the transformation, nor read back for the reduction. Instead, one or more partials sums remain in on-chip registers and get incremented as intermediate values become available; these may become available in a parallel fashion and, ultimately, we may reduce either the intermediate values or final partial-sums in parallel over a binary tree. Given B intermediates or partial-sums, the computational order of the binary-tree reduction is $\mathcal{O}(\log B)$, but it carries high inter-thread communication. On hardware with even modest communication latency between threads, such as between multiple cores on CPUs, there is often little speed advantage to a binary-tree reduction. On the other hand, with almost no communication latency between small groups of threads on GPUs, the final parallel reduction often shines. We return to this point below.

Multi-core CPUs—Modern laptop, desktop and server computers hold sockets for 1 to 8 separate CPU chips and each chip consists of 1 to 72 independent processing units called cores that can execute different computing operations simultaneously. Cores on the same chip share only a small amount of low-level, high-speed cache to facilitate communication, while cores on different chips often share only high-level, slow-speed cache if any. A drawback of this architecture is that a single memory bus connects the cache to RAM. The rate at which data moves across this bus, called the memory bandwidth, is several times less than the total rate of numerical operations across all cores. For numerically intensive computation on small amounts of data, this rarely presents a problem. As data sizes grow, memory bandwidth limitations emerge.

Since multi-core hardware is designed to perform independent operations, the operating system provides tools to coordinate their behavior that the computational statistician accesses through software libraries that are built into many programming languages. THREADING BUILDING BLOCKS (TBB) is one popular, open-source and cross-platform library (Reinders, 2007) that provides a convenient and expressive application programming interface (API) for multi-core parallelization of transformations and reductions. For example, the R package RCPP PARALLEL (Allaire et al., 2016) wraps TBB, making it immediately accessible to R and C++ statistical developers.

In a multi-core CPU environment, effective parallelization of the transformation-reductions in Equations (2) and (9) first evenly partitions the task into a modest number of parallel threads S, where $S \leq$ the total number of cores available. Without SIMD, each thread can only perform the rate-limiting floating point operations one-at-a-time but benefits from the large size of the CPU caches that automatically hold multiple copies of all of $\mathbf{X}$ close to the cores. Using TBB, we assign elements r_{ij} (for the likelihood) or $\mathbf{r}_{ij}$ (for the gradient) that hold j constant to the same thread, such that a core loads $\mathbf{x}_j$ and $\mathbf{y}_j$ (the jth column or row of $\mathbf{Y}$) into an on-chip register and reuses it many times. Beyond our specific task partitioning, no specialized programming is necessary for a compiler to generate this code. In parallel, the threads accumulate Spartial-sums that TBB stores to RAM. A final, serial reduction of the Spartial-sums takes negligible time.

Many-core GPUs—GPUs contain 100s to 1000s of cores on a single chip and come, on the smaller side, integrated directly into a CPU or, on the larger side, as add-on cards that interface with laptop, desktop or server computers. Unlike the independent cores in a CPU, small blocks of GPU-cores must execute the same instructions simultaneously, but on potentially different data. While this appears to be a strong disadvantage, it greatly simplifies thread management. Blocks of threads may communicate almost instantly using shared memory in register-space directly in hardware, and efficiently scheduling many more threads to execute than cores hides memory transaction latency with RAM, called global memory on a GPU, because many tasks are in flight simultaneously. Embarrassingly parallel tasks with no communication and low data reuse, such as independent simulation, run modestly faster on a GPU than CPU because the total rate of numerical operations across 1000s of GPU cores is currently larger than 10s of CPU cores, although the gap is narrowing. But GPUs have limited memory cache, so exploiting the shared memory with many short-lived cooperative threads leads to the greatest performance boosts.

To evaluate the log-likelihood transformation-reduction on the GPU, we generate $S = N \times N$ threads and task each thread ij with computing only one entry r_{ij}. We block threads together in $B \times B$ work-groups, such that each work-group IJ contains threads tasked to consecutive i and consecutive j (Figure 2). By setting $B = 16$ to a small power-of-two, threads within a work-group can communicate via shared memory. First, this reduces the number of memory transactions to bring $\mathbf{X}$ and $\mathbf{Y}$ on-chip by a factor of B. For example, the first B threads in each work-group read in corresponding group entries for $\mathbf{x}_i$ and the second B threads read in entries for $\mathbf{x}_j$, then all B^2 threads have quick access. Most importantly, all threads in a work-group independently compute each r_{ij} (and concomitant rate-limiting floating point operations) in parallel, for each of the $B^2 i j$ pairs in the work-group. The threads then use shared memory again to perform a binary-tree reduction. A single thread from each group then writes its partial-sum back to global memory. A final, serial reduction on the CPU of the N^2/B^2 partial-sums takes negligible time.

Recall that the gradient evaluation consists of N-independent transformation-reductions, one for the gradient with respect to each $\mathbf{x}_i$. On the GPU, we generate $S = N \times B$ threads for B a moderate power of 2 (we choose $B = 128$). In parallel across all i, we use B threads to compute the gradient with respect to each individual $\mathbf{x}_i$. Each thread uses a for-loop to compute $[N/B]$ gradient contributions $\mathbf{r}_{ij}$ across individual js, after which the threads work in concert to perform a comprehensive binary reduction of all the terms contributing to the gradient with respect to $\mathbf{x}_i$. Besides the serial evaluations within each thread, the GPU computes the rate-limiting floating point math operations in a massively parallel manner. Furthermore, GPUs have high memory bandwidth, so it is not a problem that each thread requires a copy of $\mathbf{x}_i$ and $\mathbf{y}_i$. Finally, every work-group stores its $\mathbf{x}_i$ and $\mathbf{y}_i$ in place for efficient reuse.

We write our GPU code in the Open Computing Language (OpenCL), an open-source standard maintained by leading hardware vendors, such as AMD, Apple, IBM, Intel and NVIDIA. The OpenCL framework allows for a “program once, execute across many heterogeneous platforms,” including CPUs, GPUs and other emerging hardware accelerators, portable approach using a familiar C-like syntax. In OpenCL, we write a single function, called a kernel, for the log-likelihood and for the gradient transformation-reductions, and the library assigns these kernels to each working group independently for parallel evaluation at run-time.

Within-core vectorization—Multi- and many-core processing mainly benefit from the concurrent execution of multiple threads of instructions. Commonly overlooked in statistical computing stands an alternative form of non-concurrent, data-level parallelism called vector or SIMD processing. In vector processing, a single instruction directs the core to operate simultaneously on a short vector or packet of data stored consecutively in an extended-length register. Beginning in the mid 1990s, SIMD processors began arriving in commodity computers. On Intel x86 hardware, the instruction sets carry the names multiple math extensions (MMX), streaming SIMD extensions (SSE) and, in its most recent form, advanced vector extensions (AVX) that operate on 2 to 8 integer or floating-point values. At the time of writing not widely available, next generation AVX-512 extends AVX from 256 to 512 bit extended registers with availability set to grow over the coming years.

While almost every computer used for statistical computing supports this form of parallelism, few statistical tools explicitly exploit them, relying on compilers to inject occasional SIMD instructions through their automatic optimization procedures. Unfortunately, compiler-based automatic loop vectorization remains in its infancy, forcing developers often to hand-code SIMD instructions at bottlenecks. The learning curve is high and good documentation is scarse, but the performance pay-off makes exploring SIMD worth it. Expressive libraries wrapped into the R toolchain, like RcppXsimd and RcppNT2

(Ushey and Falcou, 2016), are emerging, making SIMD as good as a free-lunch for statistical computing.

One trick to successful SIMD parallelism consists of identifying a rate-limiting transformation in which the input data lie consecutively in memory. For the log-likelihood and its gradient, the evaluation of $\Phi(\cdot)$ is easily identifiable as the most rate-limiting set of operations using an instruction-level program profiler, such as Intel Vtune under Windows and Linux and Instruments on a Mac system. We attack this bottleneck by calling the required floating-point operations on an entire SIMD extended-length register, as opposed to a solitary floating-value, each time. Using SSE, we effectively evaluate $\Phi(\cdot)$ over 2 double-precision floating variables at a time. Using AVX, we effectively evaluate $\Phi(\cdot)$ over 4 doubles at a time and reduce computing time by more than a half.

We illustrate this technique with the calculation of distance between two vectors. If we physically order in RAM the floating-values of **X** as $\{x_{11}, \ldots, x_{1D}, x_{21}, \ldots\}$ - and pad with 0 between $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ if D is not even (SSE) or not divisible by 4 (AVX) - then the transformation $\|\mathbf{x}_i - \mathbf{x}_j\| \to \delta_{ij}$ is ripe for SIMD parallelism. In the case of SSE, exploiting these in computing the dot-product in $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ approximately halves the number of operations. Figure 3 displays the x86 SSE instructions for this transformation when $D = 2$. One instruction loads the set of packed doubles (pd) $\{x_{i1}, x_{i2}\}$ into an extended SIMD register. The next instruction loads and subtracts $\{x_{j1}, x_{j2}\}$, leaving $\{x_{i1} - x_{j1}, x_{i2} - x_{j2}\}$ in register. A third instruction forms the dot-product $(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2$ that is a single double (sd) value and a final non-SIMD instruction returns its square root. SIMD operations can also lead to super-linear speed-up (> x-fold using X-wide SIMD instructions) because they can be more cache-efficient and better identify data-dependence between instructions. This latter feature allows modern CPUs to capitalize on instruction-level parallelism through pipelining and out-of-order execution.

We have placed the algorithmic details corresponding to this discussion in Section A, where Algorithm 1 describes our massively parallel implementation of the log-likelihood computations and Algorithm 2 describes the same for the log-likelihood gradient.

2.6 Software availability

The Bayesian evolutionary analysis by sampling trees (BEAST) software package (Suchard et al., 2018) stands as a popular tool for viral phylogenetic inference. The package already implements MCMC methods to explore $p(\mathbf{S}, \phi, \mathscr{G})$ under a wide variety of evolution modeling assumptions and Cybis et al. (2015) extend BEAST to include $p(\mathbf{X}, \mathbf{\Sigma} \mid \mathscr{G})$. Here, we provide an open-source, stand-alone library MASSIVEMDS http://github.com/suchard-group/MassiveMDS that efficiently computes $\log p(\mathbf{Y} \mid \mathbf{X}, \sigma^2)$ and its gradient and currently integrates directly into BEAST via a simple application programming interface (API). The library contains a combination of C++ code for which standard compilers can generate CPU-specific vectorized instructions at compile-time and OpenCL kernels that the library constructs and compiles at run-time to facilitate GPU-vendor-specific optimization.

Distribution as a stand-alone library source code with a simple API promotes cross-platform compatibility.

To further ease adoption, we have used the RCPP package (Eddelbuettel and François, 2011) to make the library available in the R programming language as a rudimentary package so that R users can exploit massive parallelization without requiring the tool authors to be experts in parallelization themselves; this design model has served well previously (Ayres et al., 2019). Finally, we have facilitated R user access to advanced SIMD tools by making the C++ library Xsimd available with R wrapper package RcppXsimd http://github.com/OHDSI/RcppXsimd.

3 Demonstration

Each year, seasonal influenza infects at least 10% of the world population, causing as many as 500,000 deaths. Prior to 2009, four main influenza subtypes circulated among humans. Of these, influenza A lineages H3N2 and H1N1 are the most prevalent. Influenza B subtypes Yamagata and Victoria contribute to decidedly less infections. Bedford et al. (2015) related this difference in epidemic success to differences in the rate of antigenic evolution. Indeed, H3N2 and H1N1 have higher rates of ‘antigenic drift’ compared to the less prevalent influenza B counterparts (Bedford et al., 2014). This results in different age-of-infection patterns that coupled with age-dependent air travel intensity explain different migration rates (Bedford et al., 2015). Antigenic evolutionary rates were estimated using a Bayesian phylogenetic MDS model with antigenic distances arising from costly chemical assays (Bedford et al., 2014). Here, we use a concept of worldwide air traffic space to derive pairwise distances between individual viral samples. Our goal is to obtain lineage-specific rates of dispersion through this air traffic space using Bayesian phylogenetic MDS.

To this end, massive parallelization facilitates phylogenetic analysis of huge collections of viral data with varied strains. We analyze 1370, 1389, 1393 and 1240 samples of type H1N1, H3N2, VIC and YAM, respectively. The observed sample originates from 189 different countries, making it ideal for testing the proposed air traffic distance framework.

3.1 Viral mobility from air traffic

We use *effective distance* (Brockmann and Helbing, 2013) between countries to incorporate global transport information into our analysis. Effective distances summarize global air travel patterns as a network of 4069 nodes (airports) and 25,453 edges (direct connections). Let α and β index two arbitrary nodes on this network. Brockmann and Helbing (2013) construct $p_{\alpha\beta}$, the probability of traveling from α to β based on flight frequency numbers, and use this probability to render

$$d_{\alpha\beta}^{e} = 1 - \log p_{\alpha\beta}$$

the effective distance between the nodes. This measure is inversely proportional to the probability of traveling between nodes, and the log transform guarantees additivity of edge lengths, a direct corollary of the fact that transition probabilities multiply. On the other hand,

$d^e_{\alpha\beta}$ does not generally equal $d^e_{\beta\alpha}$ (consider the probability of traveling from New York to the Solomon Islands), so we further symmetrize the measure to make amenable to a continuous latent space representation. Finally we aggregate the distances by country to form our data.

3.2 Dispersal inference

To elicit the prior on the tree-based covariance through time $\mathbf{V}_{\mathscr{G}}$ for the latent diffusion process, we incorporate a set of posterior trees from the analysis of Bedford et al. (2015) into the model as a finite mixture. That analysis assumes the Hasegawa, Kishino and Yano (Hasegawa et al., 1985) process with discretized rate variation (Yang, 1996) to model sequence substitution as a continuous-time Markov chain along an unknown tree $\mathscr{G}$. For a prior distribution over $\mathscr{G}$, Bedford et al. (2015) elect for a flexible, non-parametric coalescent-based process (Gill et al., 2013) and give each strain its own tree. Bedford et al. (2015) use Haemagglutinin coding sequences, numbering 9139 H3N2, 3789 H1N1, 2577 VIC and 1821 YAM in total and coming from the 9 geographic regions of USA/Canada, South America, Europe, India, North China, South China, Japan/Korea, Southeast Asia and Oceania.

First, we visualize the worldwide air traffic space using a two-dimensional latent space model. In Figure 4, the posterior medians of 189 countries arrange themselves according to our worldwide air traffic distances. Continental and otherwise geographic blocks of countries (given by similar colors) hew together. Within blocks, economic powerhouses tend toward the center of the space because they are more closely connected by air travel to other economic giants from other blocks and hence other blocks in general. For example, the United States (US) ‘represents’ the Americas in red while Japan (JP) and China (CN) represent East Asia in pink.)

As indicated, the rate of dispersal for each individual viral strain is an important quantity of scientific interest, and a key question is whether one may accurately infer these rates with a phylogenetic MDS model trained on latent airspace data. For the multivariate Brownian diffusion, $\mathrm{tr}^{(\mathbf{\Sigma})}$ is the univariate measure of instantaneous dispersal satisfying

$$\langle \mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{x} \rangle = \mathrm{tr}(\mathbf{\Sigma})\mathrm{d}t,$$

where $\mathbf{\Sigma}$ is the same as in Equation (3) and $\mathrm{d}\mathbf{x}$ is the instantaneous change in $\mathbf{x}$ as described by the stochastic differential equations governing multivariate Brownian motion. To infer these quantities we must choose the latent dimensionality of our MDS model. As described above, we use 5-fold cross-validation, which dictates a 6-dimensional latent space: the average log-likelihoods for latent dimensions 2 through 7 are -7.1×10^6, -4.2×10^6, -3.4×10^6, 3.5×10^6, -2.8×10^6, and -7.0×10^6.

Conditioning on the finite mixture of trees within a Gibbs sampler, we use a GPU to generate 2 million HMC states in roughly 48 hours. GPU based HMC accelerates sampling over latent locations, but we generate a large number of Markov chain states because changes in tree topology and branch lengths result in an array of posterior geometries, all of which require exploration. With a fixed tree, one needs to generate an order of magnitude fewer samples for similar problems at similar scale. Gibbs steps for both the MDS scale parameter σ^2 and the low-dimensional covariance $\mathbf{\Sigma}$ are straightforward and do not slow sampling of other model parameters. Using this MCMC sample, we obtain empirical posterior densities for strain-specific dispersal rates and present them in Figure 5. In order, the posterior modes of the evolutionary diffusion rates for the four subtypes are 900, 700, 640 and 550 squared effective distance units per year for H3N2, H1N1, VIC, and YAM, respectively. The relative distributions are in line with discrete migration rate estimates between worldwide regions obtained by Bedford et al. (2015), and they largely follow differences in posterior means for antigenic drifts for the same lineages (1.01, 0.62, 0.42, and 0.32 for H3N2, H1N1, VIC, and YAM, respectively, Bedford et al. (2014)). That scalings between the studies are different should not be surprising: one set of distances arises from biological measures; another comes from transportation metrics. Nonetheless, our result corroborates the results of the former study in terms of relative evolutionary rates.

Distinct relative rates of dispersal cohere to qualitatively different phylogenies. We present posterior modes of inferred strain-specific trees obtained from the same MCMC sample in Figure 6. The trees belonging to the subtype B lineages and H1N1 are much bushier than that of H3N2. The latter lineage maintains a steady rate of evolution, and the former lineages display a periodic switch between years characterized by long and short branch lengths. Short branches indicate small effective population sizes – a result of rapid population turnover – while long branches indicate large effective population size. Indeed, Bedford et al. (2014) infer similar phylogenies.

Of the four subtypes in our study, YAM has the oldest most recent common ancestor (MRCA), which takes place around 1994. VIC, the other influenza B subtype, has the next oldest MRCA (circa 1997), followed by the influenza A subtypes H1N1 (circa 1998) and H3N2 (circa 2000). Rooted at their MRCAs, the trees extend through worldwide air traffic space: Figure 6 colors each by position with respect to the first latent dimension. As a relic of its rapid dispersion, H3N2 has branches that quickly oscillate between brown, green and red as the lineage travels through the latent space. On the other hand type B viruses (and to a lesser extent H1N1) have entire clusters characterized by a single locality in air traffic space, as indicated by slowly changing hews.

3.3 Parallelization

To produce the CPU results in this section (as well as Section B), we use an iMac Pro with a 10-core Intel Xeon processor clocked at 3.0 GHz, 32 GB DDR4 memory (2666 MHz), and 23.75MB cache. With hyperthreading $(\times 2$ instructions per cycle) and AVX $(\times 4)$, it achieves 240 Gflops peak double-precision floating point performance. For the GPU results, we use an NVIDIA Quadro GP100 with 3485 single-precision floating point CUDA cores and 16 GB HBM2 memory, achieving roughly 5 Tflops double-precision floating point performance.

Table 1 compares GPU, multi-core, and single core implementations of log-likelihood and log-likelihood gradient evaluations for 5,338 samples or approximately 14 million pairwise distance data points. We also compare SSE vectorization and no SIMD against AVX vectorization for a single core. For each processor setting, we perform 100 evaluations and report the average speedup. Reported speedups are relative to AVX-based, single-core processing for each evaluation type (likelihood or gradient).

Single core, SSE computations are slightly slower (likelihood: 590 ms; gradient: 950 ms), and non-vectorized computations are significantly slower (likelihood: 1,016 ms; gradient: 1,715 ms), than single core, AVX CPU computations. For multi-core CPU processing, we find relative speedups that scale roughly linearly with the number of cores. With 12 cores, AVX averages 44 ms per likelihood evaluation and 77 ms per gradient evaluation, roughly 10 times faster than the respective 420 and 716 ms per evaluation for single core AVX. Again, these results arise from an application with 5338 locations, so they are particularly encouraging given that we use a CPU with roughly 24MB cache, maxing out at $N \approx 1730$ 6-dimensional locations ($\mathbf{X}_{N \times 6}$) and their pairwise observations ($\mathbf{Y}_{N^2}$) stored in double-precision. Nonetheless, we posit that a top-of-the-line, modern CPU with 70MB cache capable of holding roughly 2955 locations and concomitant observation matrix could deliver even greater speed.

Averaging 4.5 ms for the likelihood and 4 ms for the gradient, GPU implementations are reliably around 100 times faster than single core, AVX implementations. For inference for the illustration with H1N1, H3N2, VIC, and YAM, the GPU requires 48 hours to generate 2 million HMC states. Back of the envelope calculation shows the same posterior inference requiring almost a full solar revolution for the single core AVX implementation. We place additional scaling studies in Section B of the Appendix.

Finally, we allow that there are many criteria by which to judge software and respective hardware implementations. The NVIDIA Quadro GP100 we use is top-of-the-line and typically represents a purchase additional to whichever computer one might be working with, whereas the majority of CPUs do not. We also recognize that such technology advances at great speeds, gradually becoming less expensive and proliferating in use. For these competing reasons we have developed software to exploit the strengths of both CPUs and GPUs, whether through vectorized, multi-core or many-core processing.

4 Discussion

We developed Bayesian phylogenetic MDS to visualize pathogen diffusions and learn related scientific quantities. We used ‘airspace distance’ between viral samples to model the dispersion of four different strains of flu: H1N1, H3N2, Victoria and Yamagata. Doing so, we obtained established strain-specific diffusion rates.

But inference for large collections of viral samples is not easy. We showed that Bayesian MDS is ripe for parallel computation, and that massive parallelization provides massive speedups for likelihood evaluations, likelihood-gradient evaluations and, hence, HMC iterations. In particular, GPU-based calculations were over 100 times faster than respective single-core based calculations and over 20 times faster than respective multi-core calculations. In practical terms, massive parallelization can finish in a day what a single core can do in a year! Moreover, these massive accelerations are available to Bayesian MDS in general and not limited to phylogenetic MDS.

We note that there are other models that are worth exploring: Hoff et al. (2002) outlines latent space approaches that are alternatives to MDS; Ramsay (1982) provides alternatives to the truncated normal such as, e.g. the inverse-Gaussian; Oh and Raftery (2007) employ a mixture of Gaussians as prior over latent positions. All three of these directions would be amenable to phylogenetic extensions similar to that of MDS developed here. Indeed, a phylogenetic extension of Oh and Raftery (2007) would be useful for clustering pathogens, and, hence, predicting evolutionary dynamics. For viral samples accompanied by metadata labels, one might use latent locations as predictors of, e.g., patient outcomes. In this case, Holbrook et al. (2017) provides a road map for joint inference over the hierarchical model's MDS and predictive components.

A different kind of question is whether one might make GPU and multi-core SIMD speedups available for a broader class of Bayesian models. Li et al. (2019) use neural networks to approximate an arbitrary model's log-posterior gradient and thus avoid expensive HMC gradient computations in a Big Data setting. On the other hand, GPUs greatly accelerate fitting and evaluation of deep neural networks (Bergstra et al., 2011). It seems natural to combine these insights to power HMC based Bayesian inference on a massive scale.

Less straightforward are geometric extensions to phylogenetic MDS. For example, Zhou et al. (2018) rely on the similarities between hyperbolic space and tree space as defined in Billera et al. (2001) (i.e. negative curvature) to visualize tree structure using the Poincaré ball. Inference for a respective Bayesian model could be done using an intrinsic version of geodesic Monte Carlo (Holbrook et al., 2018). Another interesting, geometrically inspired model is Lorentzian MDS (Clough and Evans, 2017). Here, time between samples would contribute negative distance while space between sequences would contribute positive distance, leading to visualization with non-symmetric axes. Geometric and otherwise, all the above directions are potentially fruitful for Bayesian phylogenetic inference.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 725422-ReservoirDOCS) and from the National Institutes of Health (R01 AI107034, R01 HG006139 and LM011827) and the National Science Foundation (IIS 1251151 and DMS 1264153). We gratefully acknowledge support from NVIDIA Corporation with the donation of parallel computing resources used for this research. The Artic Network receives funding from the Wellcome Trust through project 206298/Z/17/Z. PL acknowledges support by the Research Foundation - Flanders ('Fonds voor Wetenschappelijk Onderzoek – Vlaanderen', G066215N, G0D5117N and G0B9317N). GB acknowledges support from the Interne Fondsen KU Leuven / Internal Funds KU Leuven under grant agreement C14/18/094. AJH is funded by NIH-NIAID grant K25AI153816.

Appendix A.: Algorithms

We present Algorithms 1 and 2 for parallel computing of the likelihood and log-likelihood gradient, respectively. Algorithmic details remain the same for multi-core CPU and GPU approaches, but implementations do not. Also, for the CPU implementation, B is the size of the SIMD extended register, but it is the size of the work group for the GPU implementation.

Algorithm 1

Parallel computation of likelihood

```
parfor $IJ \in \{1, \ldots, \lfloor N/B \rfloor\} \times \{1, \ldots, \lfloor N/B \rfloor\}$ do
    parfor $ij \in \{1, \ldots, B\} \times \{1, \ldots, B\}$ do
        if $I \times B + i < N$ and $J \times B + j < N$ then
            copy $\mathbf{x}_i, \mathbf{x}_j$ to local ▷ first 2 $B$ threads
            calculate $\delta_{ij}$ ▷ all $B^2$ threads, using SIMD Figure 3
            copy $y_{ij}$ to local
            locally compute $r_{ij}$
        end if
    end parfor
    compute partial sum $r_{IJ}$ ▷ binary tree reduction on chip
    write $r_{IJ}$ to global memory ▷ using single thread
end parfor
$p(\mathbf{Y} \mid \mathbf{X}, \sigma^2) \leftarrow \sigma^{N(1-N)/2}\exp(-\sum_{IJ} r_{IJ})$ ▷ on CPU
```

Algorithm 2

Parallel computation of gradient

```
parfor $i \in \{1, \ldots, N\}$ do
    copy $\mathbf{x}_i$ to local ▷ $B$ threads
    parfor $J \in \{1, \ldots, \lfloor N/B \rfloor\}$ do
        $j \leftarrow J$
        while $j < N$ do
            copy $\mathbf{x}_j$ to local ▷ $B$ threads
            $\Delta_{ij} \leftarrow \mathbf{x}_i - \mathbf{x}_j$ ▷ first two steps of SIMD Figure 3
            calculate $\delta_{ij}$ ▷ final two steps of SIMD Figure 3
            copy $y_{ij}$ to local
            $\nabla_{iJ} \leftarrow \nabla_{iJ} - \left(\frac{(\delta_{ij} - y_{ij})}{\sigma^2} + \frac{\phi(\delta_{ij}/\sigma)}{\sigma\Phi(\delta_{ij}/\sigma)}\right)\frac{\Delta_{ij}}{\delta_{ij}}$
            $j \leftarrow j + B$
        end while
    end parfor
    $\frac{\partial}{\partial \mathbf{x}_i}\log p(\mathbf{Y} \mid \mathbf{X}, \sigma^2) \leftarrow \sum_J \nabla_{iJ}$ ▷ binary tree reduction on chip
end parfor
```

Appendix B.: Additional scaling studies

First, we provide insight into one of the main computational challenges of the MDS likelihood: the truncation term

$$\sum_{i < j} \log\Phi(\delta_{ij}/\sigma)\,.$$

This term is computationally intensive because of its $\mathcal{O}(N^2)$ floating point operations. Since the term is the sum of a single simple function applied independently to all δ_{ij}, parallelization should deliver significant speedups. Figure 7 shows relative speedups (over no SIMD, single core) of likelihood computation using SIMD vector processing, multi-core CPU and GPU processing. In 100 independent iterations, we generate 5,338 samples (approximately 14 million data points) and time the likelihood and gradient evaluations. When the truncation term is not calculated, the 12-core implementation is only 4 times faster than the single core without SIMD, and GPU calculations are only 16 times faster. But when truncation is included (i.e., the correct model), 12-core implementation is more than 16 times faster, and GPU 200 times faster, than the single core implementation without SIMD.

Figure 8 shows seconds per likelihood and log-likelihood gradient evaluations for GPU and multi-core implementations. Results are based on the two-dimensional latent space model and distances arising from randomly sampled Gaussian points. Speeds are averaged over 100 independent tests. Lower values correspond to less computing time. Both for the likelihood and the gradient, GPU evaluation speed (bottom) stays orders of magnitude faster than multi-core and single core evaluation speeds.

References

Allaire J, Francois R, Ushey K, Vandenbrouck G, Geelnard M, and Intel. 2016. RcppParallel: Parallel Programming Tools for ‘Rcpp’. R package version 4.3.19.

Ayres DL, Cummings MP, Baele G, Darling AE, Lewis PO, Swofford DL, Huelsenbeck JP, Lemey P, Rambaut A, and Suchard MA 2019. BEAGLE 3: Improved Performance, Scaling, and Usability for a High-Performance Computing Library for Statistical Phylogenetics. Systematic Biology 68:1052–1061. [PubMed: 31034053]

Bahl J, Nelson MI, Chan KH, Chen R, Vijaykrishna D, Halpin RA, Stockwell TB, Lin X, Wentworth DE, Ghedin E, et al. 2011. Temporally structured metapopulation dynamics and persistence of influenza A H3N2 virus in humans. Proceedings of the National Academy of Sciences 108:19359–19364.

Beam AL, Ghosh SK, and Doyle J. 2016. Fast Hamiltonian Monte Carlo using GPU computing. Journal of Computational and Graphical Statistics 25:536–548.

Bedford T, Riley S, Barr IG, Broor S, Chadha M, Cox NJ, Daniels RS, Gunasekaran CP, Hurt AC, Kelso A, et al. 2015. Global circulation patterns of seasonal influenza viruses vary with antigenic drift. Nature 523:217–220. [PubMed: 26053121]

Bedford T, Suchard MA, Lemey P, Dudas G, Gregory V, Hay AJ, McCauley JW, Russell CA, Smith DJ, and Rambaut A. 2014. Integrating influenza antigenic dynamics with molecular evolution. Elife 3:e01914. [PubMed: 24497547]

Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A, et al. 2011. Theano: Deep learning on GPUs with Python. Pages 1–48 in NIPS 2011, BigLearning Workshop, Granada, Spain vol. 3 Citeseer.

Billera L, Holmes S, and Vogtmann K. 2001. Geometry of the space of phylogenetic trees. Advances in Applied Mathematics 27:733–767.

Bloom DE, Black S, and Rappuoli R. 2017. Emerging infectious diseases: a proactive approach. Proceedings of the National Academy of Sciences 114:4055–4059.

Brockmann D. and Helbing D. 2013. The hidden geometry of complex, network-driven contagion phenomena. science 342:1337–1342. [PubMed: 24337289]

Brown R. 1828. A brief account of microscopical observations made in the months of June, July and August, 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. Philosophical Magazine 4:161–173.

Cavalli-Sforza LL and Edwards AW 1967. Phylogenetic analysis. models and estimation procedures. American Journal of Human Genetics 19:233–257. [PubMed: 6026583]

Clough JR and Evans TS 2017. Embedding graphs in Lorentzian spacetime. PloS one 12:e0187301. [PubMed: 29107967]

Cybis G, Sinsheimer J, Bedford T, Mather A, Lemey P, and Suchard M. 2015. Assessing phenotypic correlation through the multivariate phylogenetic latent liability model. Annals of Applied Statistics 9:969–991.

DeSarbo WS, Kim Y, and Fong D. 1998. A Bayesian multidimensional scaling procedure for the spatial analysis of revealed choice data. Journal of econometrics 89:79–108.

Duane S, Kennedy AD, Pendleton BJ, and Roweth D. 1987. Hybrid Monte Carlo. Physics letters B 195:216–222.

Eddelbuettel D. and François R. 2011. Rcpp: Seamless R and C++ integration. Journal of Statistical Software 40:1–18.

Felsenstein J. 1985. Phylogenies and the comparative method. American Naturalist 125:1–15.

Fosdick BK, McCormick TH, Murphy TB, Ng TLJ, and Westling T. 2019. Multiresolution network models. Journal of Computational and Graphical Statistics 28:185–196. [PubMed: 31447541]

Freckleton RP 2012. Fast likelihood calculations for comparative analyses. Methods in Ecology and Evolution 3:940–947.

Geisser S. 1975. The predictive sample reuse method with applications. Journal of the American statistical Association 70:320–328.

Gelman A, Stern HS, Carlin JB, Dunson DB, Vehtari A, and Rubin DB 2013. Bayesian data analysis. Chapman and Hall/CRC.

Gill MS, Lemey P, Faria NR, Rambaut A, Shapiro B, and Suchard MA 2013. Improving Bayesian population dynamics inference: a coalescent-based model for multiple loci. Molecular biology and evolution 30:713–724. [PubMed: 23180580]

Hasegawa M, Kishino H, and Yano T. 1985. Dating the human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution 22:160–174. [PubMed: 3934395]

Ho LST and Ané C. 2014. A linear-time algorithm for Gaussian and non-Gaussian trait evolution models. Systematic Biology 3:397–402.

Hoff PD, Raftery AE, and Handcock MS 2002. Latent space approaches to social network analysis. Journal of the american Statistical association 97:1090–1098.

Holbrook A, Lan S, Vandenberg-Rodes A, and Shahbaba B. 2018. Geodesic Lagrangian Monte Carlo over the space of positive definite matrices: with application to Bayesian spectral density estimation. Journal of Statistical Computation and Simulation 88:982–1002. [PubMed: 31105358]

Holbrook A, Vandenberg-Rodes A, Fortin N, and Shahbaba B. 2017. A Bayesian supervised dual-dimensionality reduction model for simultaneous decoding of LFP and spike train signals. Stat 6:53–67. [PubMed: 28529731]

Kruskal JB 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29:1–27.

Lee A, Yau C, Giles MB, Doucet A, and Holmes CC 2010. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. Journal of computational and graphical statistics 19:769–789. [PubMed: 22003276]

Leimkuhler B. and Reich S. 2004. Simulating Hamiltonian dynamics vol. 14. Cambridge university press.

Lemey P, Rambaut A, Drummond A, and Suchard M. 2009. Bayesian phylogeography finds its roots. PLoS Computational Biology 5:e1000520. [PubMed: 19779555]

Lemey P, Rambaut A, Welch J, and Suchard M. 2010. Phylogeography takes a relaxed random walk in continuous space and time. Molecular Biology and Evolution 27:1877–1885. [PubMed: 20203288]

Li L, Holbrook A, Shahbaba B, and Baldi P. 2019. Neural network gradient Hamiltonian Monte Carlo. Computational Statistics 34:281–299. [PubMed: 31695242]

Manton JH 2013. A primer on stochastic differential geometry for signal processing. IEEE Journal of Selected Topics in Signal Processing 7:681–699.

Neal RM 2011. MCMC using Hamiltonian dynamics. Handbook of Markov Chain Monte Carlo 2.

Nelson MI, Viboud C, Vincent AL, Culhane MR, Detmer SE, Wentworth DE, Rambaut A, Suchard MA, Holmes EC, and Lemey P. 2015. Global migration of influenza A viruses in swine. Nature communications 6.

Nye TM and White M. 2014. Diffusion on some simple stratified spaces. Journal of mathematical imaging and vision 50:115–125.

Oh M. and Raftery A. 2001. Bayesian multidimensional scaling and choice of dimension. Journal of the American Statistical Association 96:1031–1044.

Oh M-S and Raftery AE 2007. Model-based clustering with dissimilarities: A Bayesian approach. Journal of Computational and Graphical Statistics 16:559–585.

Pearl J. 1982. Reverend Bayes on inference engines: A distributed hierarchical approach. Pages 133–136 in AAAI-82: Proceedings of the Second National Conference on Artificial Intelligence.

Pybus OG, Suchard MA, Lemey P, Bernardin FJ, Rambaut A, Crawford FW, Gray RR, Arinaminpathy N, Stramer SL, Busch MP, et al. 2012. Unifying the spatial epidemiology and molecular evolution of emerging epidemics. Proceedings of the National Academy of Sciences 109:15066–15071.

Ramsay JO 1982. Some statistical approaches to multidimensional scaling data. Journal of the Royal Statistical Society. Series A (General) 145:285–312.

Reinders J. 2007. Intel Threading Building Blocks. First ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Schmidberger M, Morgan M, Eddelbuettel D, Yu H, Tierney L, and Mansmann U. 2009. State-of-the-art in parallel computing with R. Journal of Statistical Software 47.

Suchard M, Kitchen C, Sinsheimer J, and Weiss R. 2003. Hierarchical phylogenetic models for analyzing multipartite sequence data. Systematic Biology 52:649–664. [PubMed: 14530132]

Suchard M. and Rambaut A. 2009. Many-core algorithms for statistical phylogenetics. Bioinformatics 25:1370–1376. [PubMed: 19369496]

Suchard M, Wang Q, Chan C, Frelinger J, Cron A, and West M. 2010a. Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures. Journal of Computational and Graphical Statistics 19:419–438. [PubMed: 20877443]

Suchard M, Weiss R, and Sinsheimer J. 2001. Bayesian selection of continuous-time Markov chain evolutionary models. Molecular Biology and Evolution 18:1001–1013. [PubMed: 11371589]

Suchard MA, Holmes C, and West M. 2010b. Some of the what?, why?, how?, who? and where? of graphics processing unit computing for Bayesian analysis. Bulletin of the International Society for Bayesian Analysis 17:12–16.

Suchard MA, Lemey P, Baele G, Ayres DL, Drummond AJ, and Rambaut A. 2018. Bayesian phylogenetic and phylodynamic data integration using BEAST 1.10. Virus Evolution 4:vey016.

Ushey K. and Falcou J. 2016. RcppNT2: 'Rcpp' Integration for the 'NT2' Scientific Computing Library. R package version 0.1.0.

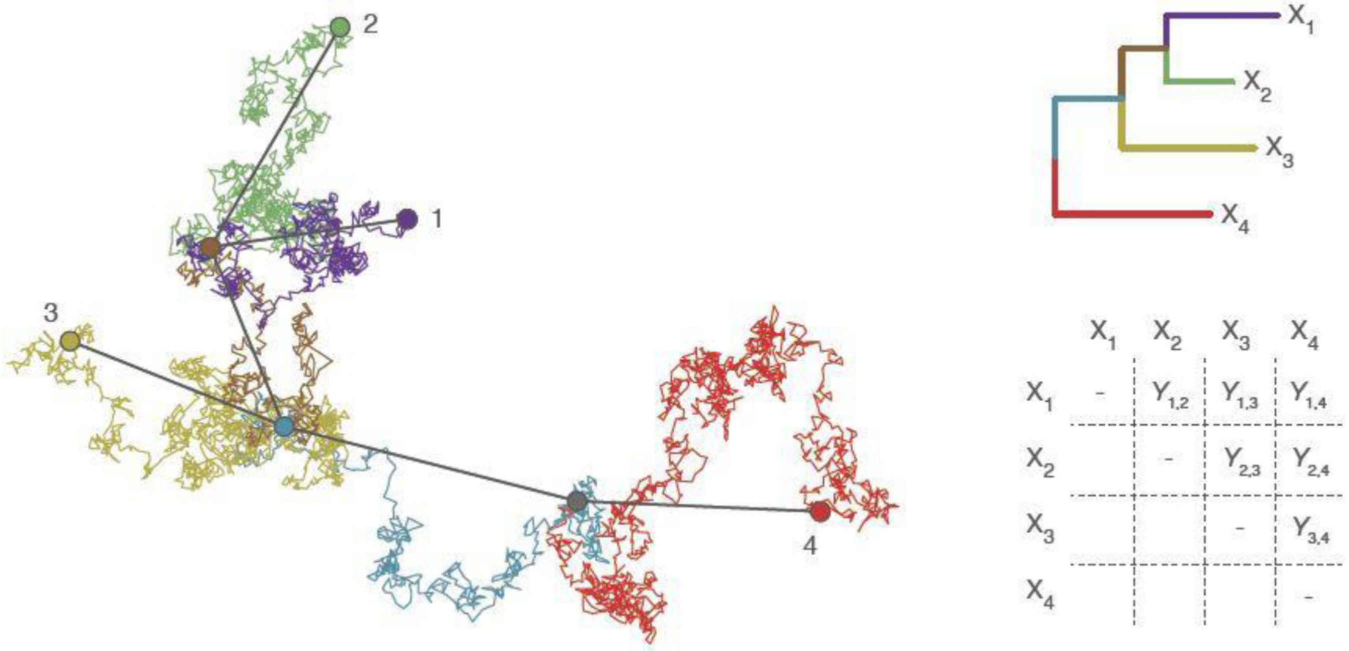
Vehtari A, Gelman A, and Gabry J. 2017. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing 27:1413–1432.

Wiener N. 1958. Nonlinear Problems in Random Theory. MIT Press and Wiley.

Yang Z. 1996. Among-site rate variation and its impact on phylogenetic analyses. Trends Ecol. Evol 11:367–372. [PubMed: 21237881]

Zhou H, Lange K, and Suchard M. 2010. Graphics processing units and high-dimensional optimization. Statistical Science 25:311–324. [PubMed: 21847315]

Zhou Y, Smith BH, and Sharpee TO 2018. Hyperbolic geometry of the olfactory space. Science advances 4:eaaq1458.



	X_1	X_2	X_3	X_4
X_1	-	$Y_{1,2}$	$Y_{1,3}$	$Y_{1,4}$
X_2		-	$Y_{2,3}$	$Y_{2,4}$
X_3			-	$Y_{3,4}$
X_4				-

Fig. 1.

Multivariate Brownian diffusion along a phylogeny as a latent Gaussian model prior. This example phylogeny has 4 tips, labeled 1, 2, 3 and 4. We depict the conditionally independent diffusion realizations in two dimensions along each branch in different colors. The root and two internal node realizations are colored as gray, blue and brown circles, while the four tip node realizations $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ are highlighted in purple, green, yellow and red, respectively. Dynamic programming enables us to integrate over all possible root and internal node realizations, returning the joint distribution of $\mathbf{X}$ as an informed prior. Pairwise distances between tip realizations relate to the observed dissimilarity distances y_{ij} of the model.

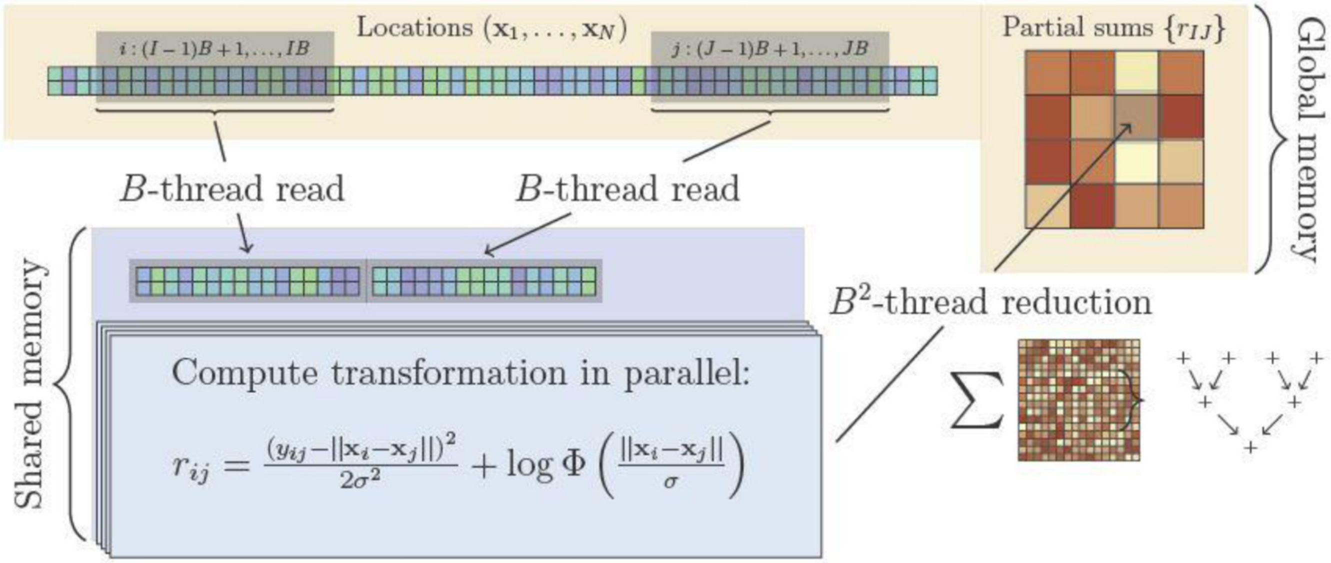
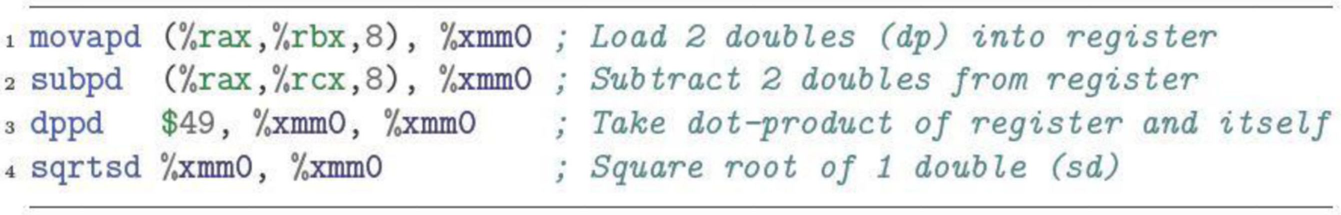

Locations $(\mathbf{x}_1,\dots,\mathbf{x}_N)$
$i:(I-1)B+1,\dots,IB$
$j:(J-1)B+1,\dots,JB$
Partial sums $\{r_{IJ}\}$
Global memory
B-thread read
B-thread read
Shared memory
B^2-thread reduction
Compute transformation in parallel:
$r_{ij}=\frac{(y_{ij}-\|\mathbf{x}_i-\mathbf{x}_j\|)^2}{2\sigma^2}+\log\Phi\left(\frac{\|\mathbf{x}_i-\mathbf{x}_j\|}{\sigma}\right)$
$\sum$

Fig. 2.

Massive parallelization strategy for computing the log-likelihood: each working group independently reads two separate batches of latent locations data from global memory, computes location pair specific likelihood contributions in parallel, efficiently adds these contributions in a binary reduction and writes the resulting partial sum to global memory.



```
movapd (%rax,%rbx,8), %xmm0 ; Load 2 doubles (dp) into register
subpd  (%rax,%rcx,8), %xmm0 ; Subtract 2 doubles from register
dppd   $49, %xmm0, %xmm0    ; Take dot-product of register and itself
sqrtsd %xmm0, %xmm0         ; Square root of 1 double (sd)
```

Fig. 3.

Single instruction, multiple data (SIMD) Intel x86 CPU processor instructions to compute $\|\mathbf{x}_i - \mathbf{x}_j\|$ for $D = 2$. These SIMD instructions simultaneously act on 2 double-precision floating-point values. In 4 lines of code, we approximately halve the total number of instructions executed to compute the distance between two vectors, resulting in almost a 2-fold speed-up.

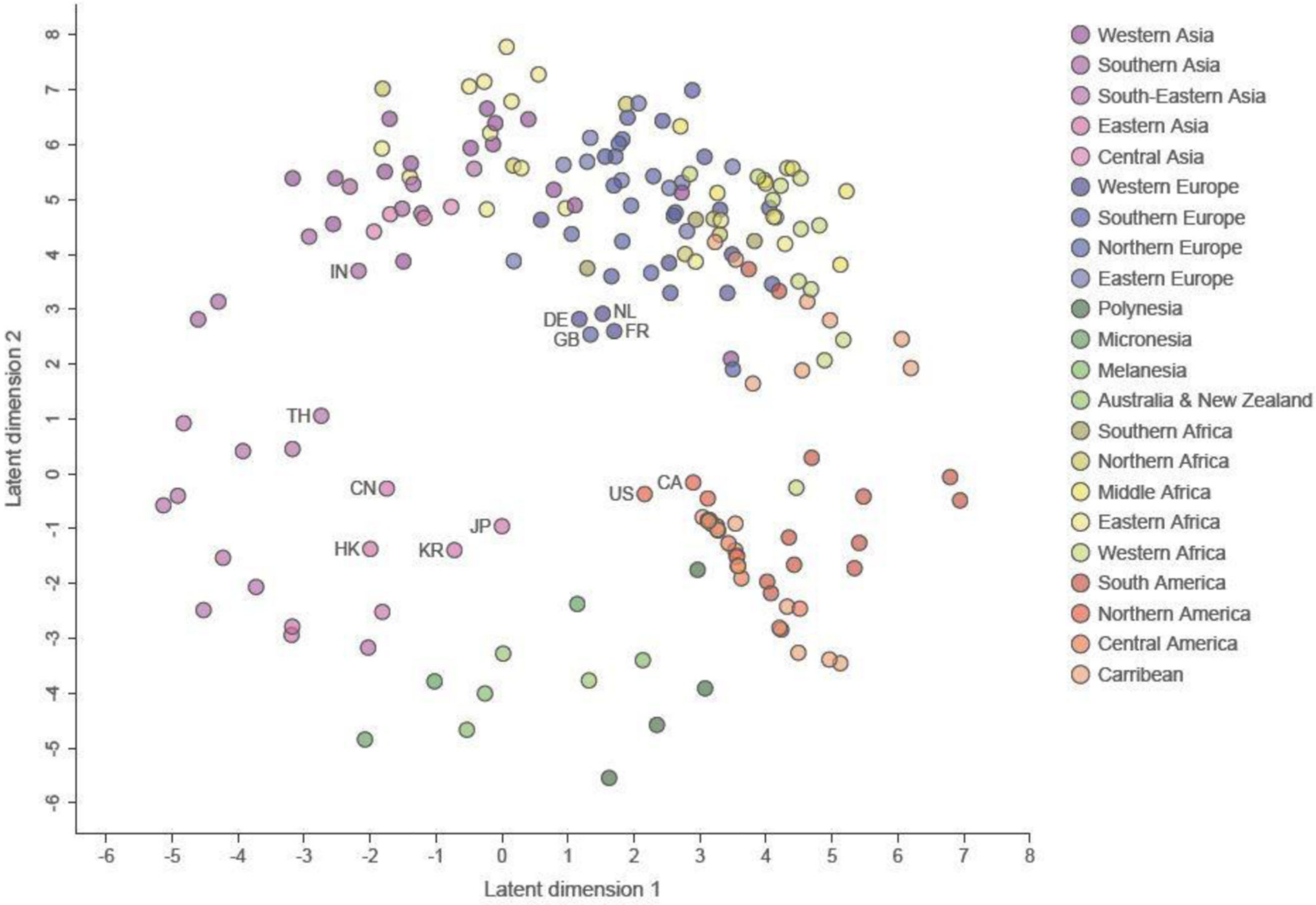


Fig. 4.

Geography of a worldwide, latent air traffic space. A 2-dimensional Bayesian multidimensional scaling model with effective worldwide air traffic space distances for data results in 189 country specific posterior medians.

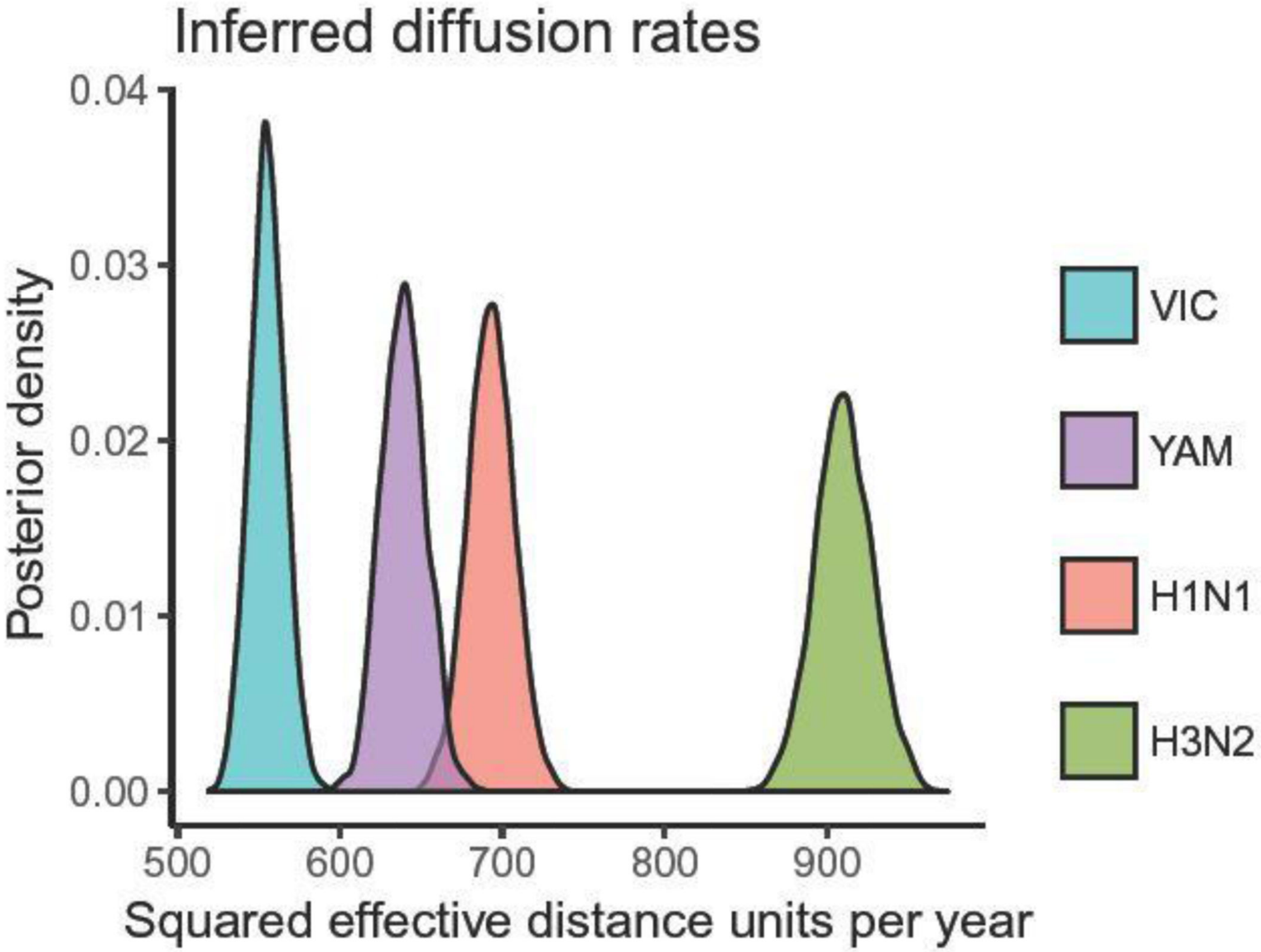


Fig. 5.

Posterior distributions of strain-specific diffusion rates inferred from 6-dimensional Bayesian phylogenetic multidimensional scaling with effective worldwide air traffic space distances for data.

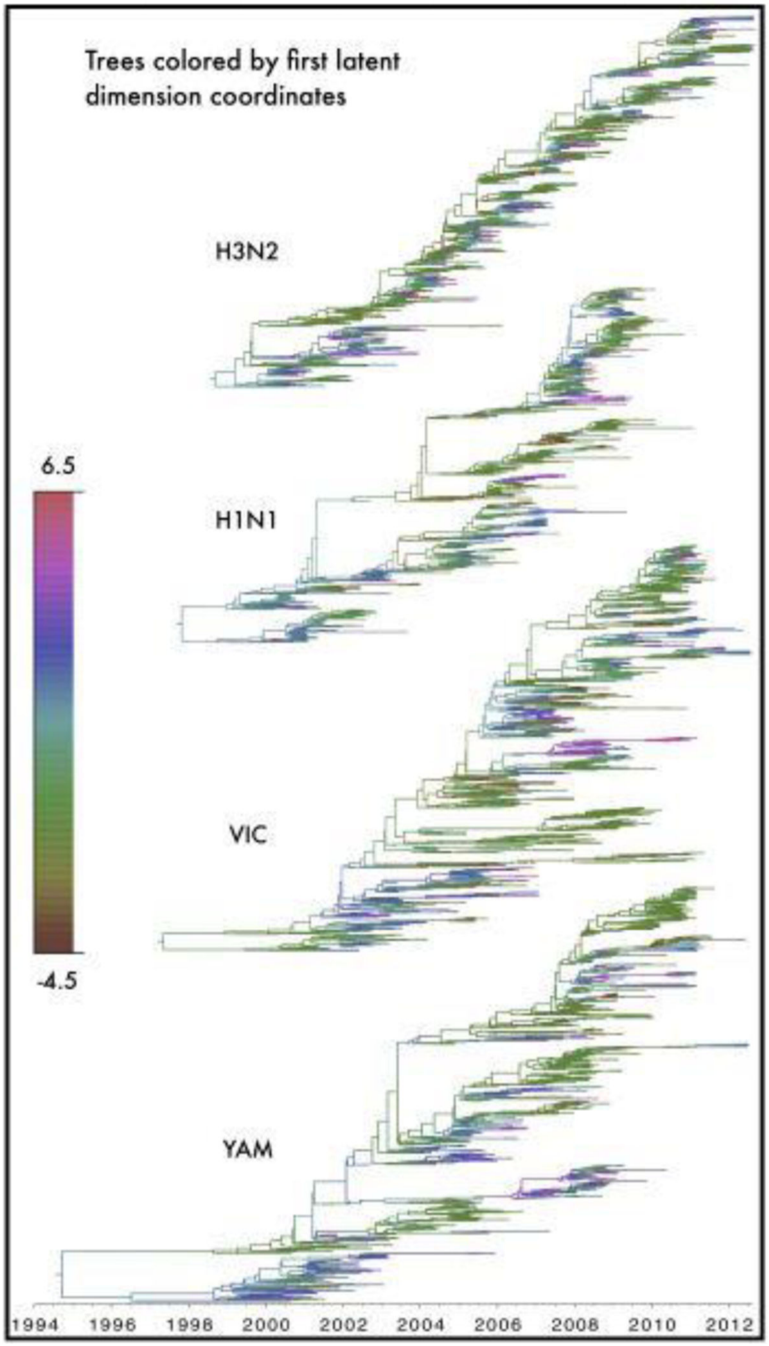


Fig. 6.

Posterior modes for trees from each strain, colored along first latent dimension of worldwide air traffic space.

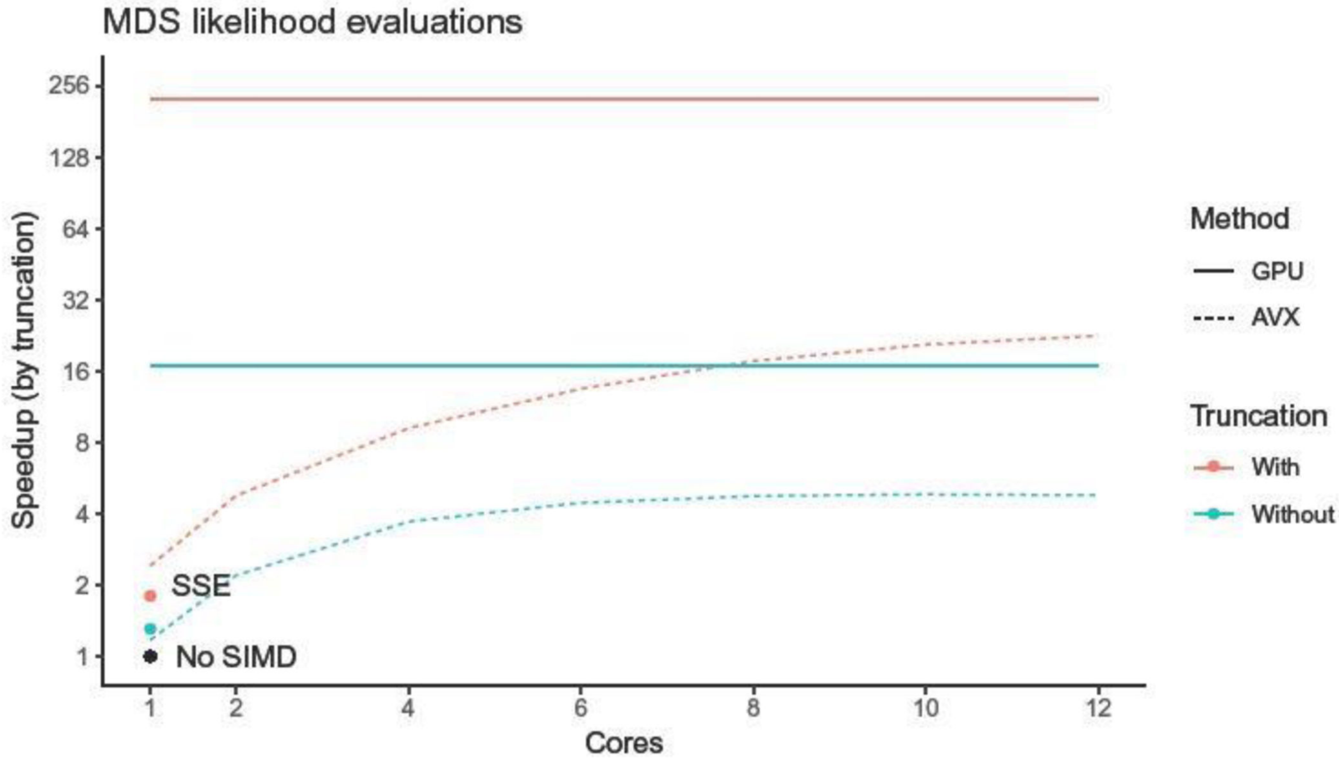


Fig. 7.

Speedup of graphics processing unit (GPU) and multi-core advanced vector extensions (AVX) computations over single core implementations of multidimensional scaling (MDS) likelihood with and without truncation. No single instruction, multiple data (No SIMD; baseline, black) implementation and streaming SIMD extensions (SSE; colored) occupy the bottom left corner.

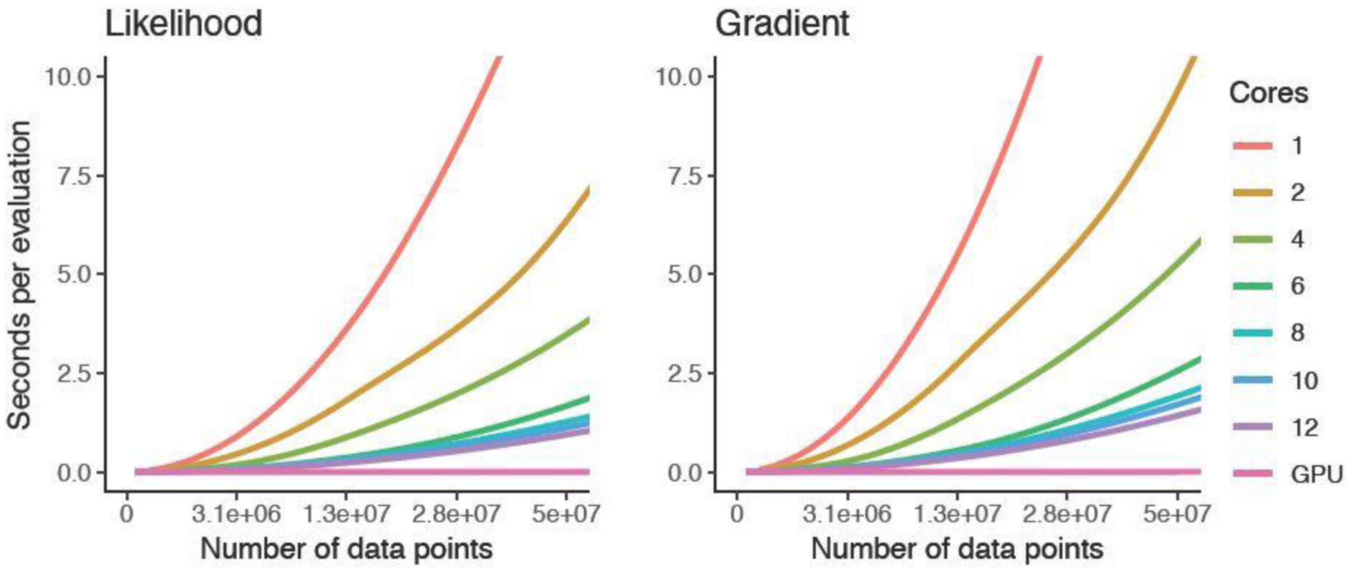


Fig. 8.

Seconds to evaluate likelihood and gradient using central processing unit (CPU) and graphics processing unit (GPU) as a function of data size. For both likelihood and gradient, computation time grows exponentially for CPU and logarithmically for GPU implementation.

Table 1

Speedup of graphics processing unit (GPU) and multi-core advanced vector extensions (AVX) computations relative to single core AVX computing. Single core implementations without single instruction, multiple data (SIMD) and with streaming SIMD extensions (SSE) occupy the bottom left corner.

Cores	**1**		**2**	**4**	**6**	**8**	**10**	**12**	**GPU**
Vectorization	None	SSE	AVX						
Likelihood	0.41	0.71	1.98	3.80	5.57	7.29	8.55	9.31	92.25
Gradient	0.42	0.75	1.96	3.73	5.37	7.14	8.46	9.18	177.77