BIOINFORMATICS

Vol. 00 no. 00 2014
Pages 1–3

High Performance Comparative Methods using GPUs and multicore CPUs

Igor Siveroni [1], Andrew Meade [1,*] and Mark Pagel [1]

[1] School of Biological Sciences, Lyle Building, University of Reading, Berkshire RG6 6BX, United Kingdom

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: We present a OpenCL/OpenMP-based GPU and multi-core implementation of Bayestraits, a popular state-of-the-art bayesian comparative methods package. In particular, we outline the OpenCL/OpenMP implementation of the likelihood functions associated to the analysis of continuous and discrete data under the GLS and continuous-time Markov models, respectively. Speedups are obtained by new implementations of matrix inversion (based on Cholesky factorization) , matrix exponentation, and the tree partial likehood function used for discrete traits. Performance tests show 3x and 4.8x speedups of the OpenCL and OpenCL/OpenMP implementations of the models associated with phylogetic analysis of continuous and discrete data, respectively.

Availability: BayesTraits executables for Windows, Linux and OS X systems, as well as test data sets, are avaiblable at http://www.evolution.rdg.ac.uk/BayesTraits.html.

Contact: a.meade@reading.ac.uk

1 INTRODUCTION

Comparative methods for analysing data across species have over the past twenty years moved from being a speciality subject confined largely to applications in organismal evolutionary biology to being an important component of analyses in nearly all biological disciplines. New rapidly growing genomic and bioinformatic databases, combined with phylogenetic trees of hundreds or even thousands of organisms mean that comparative methods can now test evolutionary hypotheses with unprecedented accuracy and generality. But these large datasets and phylogenies often require extraordinary amounts of computing power, placing them off-limits to most researchers, or requiring unfeasible amounts of time. Conventional high performance computers offer a solution, but they have a number of limitations for the biological community, including cost and the requirement of specialized installations and equipment. On the other hand, Graphical Processor Units (GPUs) are affordable and remarkably fast, and the recent development of frameworks and programming languages that enable General Purpose computing on GPUs, such as CUDA and OpenCL, make them ideal candidates to potentially make affordable high-performance computing available on the desktop. Furthermore, multithreading parallel computing APIs such as Cilk and OpenMP offer a simple and low cost way to exploit the parallel capabilities of current multi-core desktop computers.

BayesTraits (Pagel *et al.*, 2013) is a popular state-of-the-art bayesian Markov chain Monte Carlo (MCMC) comparative methods package. We have extended BayesTraits to run on GPUs using OpenCL and multiple-cores using OpenMP, in order to show the applicability of GPU and multi-core technology to tackle the current performance problems inherent to sequential implementations of comparative methods. In this paper, we present an overview of the GPU and multicore implementation of BayesTraits and, in particular, outline the implementations of the likelihood functions associated to the analysis of continuous and discrete data under the Generalized Least Squares (OpenCL) and continuous time Markov models (OpenCL/OpenMP), respectively.

2 METHODOLOGY

BayesTraits is a comparative methods package that performs analyses of trait evolution among groups of species for which a phylogeny or sample of phylogenies is available. BayesTraits can analyse continuous or discrete data in either a maximum likelihood (ML) or Markov Chain Monte Carlo (MCMC) statistical framework. The statistical models used by BayesTraits can be roughly classed into two categories, depending on the data being analysed. Continuous data are analysed in a phylogenetic generalised least squares (GLS) framework, while discrete data are analysed using a continuous time Markov model (Pagel *et al.*, 2004). Model parameters are estimated using likelihood approaches. For both continuous and discrete data, calculating the likelihood of observing the data given a statistical model and evolutionary history, accounts for over 99% of the total computational run time. In the remaining of this section, we define the likelihood function associated to the GLS and continuous time Markov models used by BayesTraits, and outline their OpenCL/OpenMP implementations.

2.1 Continuous traits and GLS

Analyses performed on continuously varying traits based on the generalised least squares (GLS) approach use the phylogenetic tree to derive a variance-covariance matrix to account for the non-independence among species (Felsenstein, 1973; Pagel, 1997). Given a phylogenetic tree containing n species (taxa) and a data set containing observations for k traits, the model takes as input the $n \times n$ variance-covariance matrix V built using the path and branch

[*] to whom correspondence should be addressed

lengths of the tree, and the nk-vector z containing the n observed values of each of trait. Let Σ be the $k \times k$ variance-variance matrix of the k traits, and α the k-vector with the notional origin or predicted root associated to each trait. The logarithm of the likelihood function is defined by:

$$ln(L) = -\frac{1}{2}[(nk)\ln(2\pi) + |V||\Sigma| + z_\alpha^{\mathrm{t}}(V^{-1} \otimes \Sigma^{-1})z_\alpha] \quad (1)$$

where $z_\alpha = z - \alpha\mathbf{1}$, $|A|$ denotes the determinat of matrix A and $\otimes$ is the Kronecker product. The computation of $ln(L)$ is largely dominated by the calculation of the inverse of V, followed by, to a significantly less degree, the vector product of z_α with the Kronecker product of the inverses of V and Σ. It is possible to compute the tree likelihood using a method that avoids computing the inverse of V (Freckleton, 2012). However, the algorithm - initially proposed in Felsenstein (1973) - cannot be applied to cases when, for example, α needs to be estimated using the MCMC approach.

Our GPU implementation takes advantage of the fact that V is positive-definite. This means that it is possible to apply Cholesky decomposition and write $V = LL^{\mathrm{t}}$, the product of a lower triangular matrix L and its transpose. Thus, the inverse of V can be expressed as $(L^{-1})^{\mathrm{t}} * L^{-1}$ and computed as a sequence of simpler operations. Matrix multiplication and its variations (e.g. multiplication of triangular matrices) can be efficiently implemented in GPUs. Cholesky factorization and triangular matrix inversion are implemented using hybrid blocked algorithms (Bientinesi *et al.*, 2008). A blocked algorithm partitions the matrix into submatrices and expresses the result in terms of operations between the matrix blocks; one of them being usually a call to a non-blocked version of the algorithm. A hybrid version of the blocked algorithm is implemented as a sequence calls to the GPU interleaved with CPU code. In the case of Cholesky decomposition, the CPU code corresponds to a call to a non-blocked version of the algorithm while the GPU part is made of calls to two kernels that perform column and matrix updates. All operations are performed in-place; no extra memory is required. Non-blocked versions of Cholesky and L^{-1} are implemented as calls to the corresponding functions in LAPACK.

2.2 Discrete traits and the continuous-time Markov model

The calculation of the tree likelihood under a continuous-time Markov model of trait evolution for discrete traits requires the computation of the probability matrix (P-Matrix) and partial likelihood associated to each node of the tree. Let n be the number of nodes of the phylogenetic tree and nos the numbers of states of the Markov process. Given rate matrix Q and a vector t of n real values, the probability matrix $P(N_k)$ associated to the k-th node of the tree is defined by e^{Qt_k}, where t^k is the length of the branch reaching N_k. We compute the exponential of a matrix by first decomposing Q into $E\Lambda E^{-1}$, where Λ is a $nos \times nos$ diagonal matrix containing the eigenvalues of Q, and E is a square matrix containing the corresponding eigenvectors. We can then express e^{Qt_k} as $Ee^{\Lambda t_k}E^{-1}$. Matrix diagonalisation is computationally expensive but since we are dealing with relatively small matrices ($nos < 100$) and the decomposition only needs to be performed once (per tree/iteration), its relative cost becomes neglible. Furthermore, using the fact that the exponential of a diagonal matrix can be obtained by replacing its diagonal terms with their exponentials, e^{Qt_k} can be written as the matrix product $E\Lambda' E^{-1}$ where Λ' is a diagonal matrix with $\Lambda'_{i,i} = e^{t_k\lambda_i}$ and $\lambda_i = \Lambda_{i,i}$ the i^{th} eigenvalue of Q. This result makes it possible to compute all P-matrix elements, for each branch of the tree, in parallel. Our general GPU implementation follows this approach, calculating each P-matrix cell independently using the following definition:

$$P(N_k)_{i,j} = E_{i,\text{-}}\Lambda' E^{-1}_{\text{-},j} = \sum_{l=1}^{nos} E_{i,l}(e^{t_k\lambda_l})E^{-1}_{l,k} \quad (2)$$

Full calculation of each of the P-Matrices associated to all tree nodes (with the exception of the root) is achieved by spawning $(n)nos^2$ workitems, one per matrix element, grouped into $n(nos)$ workgroups (one per row). Besides allocating the neccesary global buffers needed to read input and write the final output, the GPU implementation uses local memory (shared only between elements of the same workgroup) to store the (intermediate) results of performing $E_{i,\text{-}}\Lambda'$.

The second step in the calculation of the tree likelihood requires the calculation of the partial likelihood associated to each node of the tree. Let $L(N_k)$ denote the vector of likelihoods associated to each state in node N_k. The likelihood vector at the tips of the tree is determined by setting $L(N_k, s)$ to 1 if state s was observed in the species associated to tip (node) N_k, and 0 otherwise. The partial likelihood for the internal nodes is defined by:

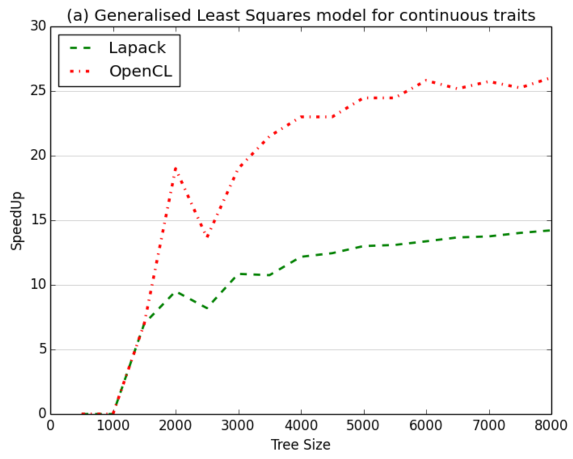
$$L(N_k, s) = \prod_{N_i \in C_k} L(N_i)P(N_i)_{-,s} \quad (3)$$

where C_k is the set of immediate descendants of N_k.

The calculation described in (3) offers limited parallelization due to the fact that the partial likelihood of an internal node requires the values of the likelihood vector of each of its descendants. We are forced to split the computation into batches, one batch per level of the tree (the tips being level 0), and work our way up the tree. We implemented different versions of this approach, where the GPU was used to calculate the likelihood of each node/state pair in parallel, for each tree level. This is effective for the lower levels of the tree (where most of the internal nodes reside) but quickly becomes inefficient as we progress up the tree, where the problem size decreases considerably and the setup costs become too large in comparison. Nevertheless, we were able to take advantage of the structure of the computation in (3) by implementing a multi-threaded OpenMP version that computes one level of the tree at a time (bottom up) but calculates the partial likelihoods in parallel by assigning a thread to each node in the group.

3 RESULTS AND DISCUSSION

We executed the GPU and multi-core implementations of the GLS and continuous time Markov models used by Bayestraits in a system made of an Intel core i7-3770k processor (4 cores, 8 threads) and an NVIDIA GeForce GTX Titan GPU card. The speedup of the double precision GPU OpenCL version of the GLS model against the sequential CPU implementation is shown in Figure 1(a). The graph plots execution speedup against tree size. In order to put our results into context, we have included the speedups obtained from the execution of the LAPACK version of the program, which is also based in a blocked implementation of Cholesky decomposition. According to the data shown, the LAPACK implementation is 14 times



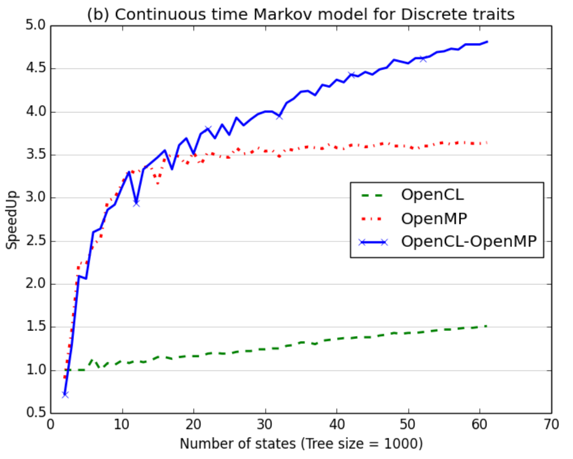


Fig. 1. Speedup of OpenCL/OpenMP implementations

faster (for a tree of 8000 taxa) than the CPU version - based on LU decomposition followed by backward and forward substituions - while the GPU implementation offers a combined 26x speedup. This means that the addition of the GPU provides a 1.8x speedup relative to LAPACK. This number reflects the difficulties in obtaining two-digit speedups in parallel versions of matrix inversion. For example, the clMagma library obtains a 1.2x speedup on the inversion of general dense matrices, while a 5x speedup is reported on Cholesky factorization. We have obtained similar results on the matrix decomposition phase but uneven performance on the second part of the process.

The results in Figure 1(b) show the speedups obtained by the parallel implementations (OpenCL/OpenMP) of the continuous time Markov model against the execution of the sequential CPU version. The model is run on a randomly-generated tree of 1000 taxa and discrete data varying from 2 to 61 states. The OpenCL version, which performs the calculation of the P-matrix on the GPU, achieves a top speedup of 1.5x while the combined OpenCL/OpenMP implementation reaches a maximum speedup of 4.8x. The pure OpenMP version achieves speedups (up to 3.5x) quickly when run on 18 states or less, while further increments on the number of states result in very little performance improvement (3.6x). It is from this point - 20 states - that the use of the GPU pays off, resulting on constant speedup improvements as the number of states grows.

The speedups shown in Figure 1(b) are in line with the differences on the time complexitiy of the two versions of the P-matrix calculation,: the sequential version requires nos^3n steps while the kernel of the OpenCL implementation requires to performs a dot product on vectors of size nos (linear). Clearly, larger number of states will provide better speedups, as shown by the positive slope of the OpenMP graphs, while the effect of increasing the tree size would be less dramatic.

As mentioned in section 2.2, OpenCL implementations of the calculation of partial tree likelihoods proved unsuccesful. It could be argued that incrementing the number of sites, which improves the problem's parallelism, would make an OpenCL implementation feasible. Unfortunately, a high number of states (as needed by the first part of the problem), requires the use of a special library to handle Quad precision, currently unavailable (or computationally expensive) on GPUs.

4 CONCLUSIONS AND FUTURE WORK

We have presented preliminary results showing the benefits of using GPU and multicore CPUs applied to the implementation of phylogenetic comparative methods. We have used OpenCL and OpenMP to implement performance-critical computations required by the calculation of the likelihood functions required by the GLS and continuous-time Markov models. Performance tests show 3x and 4.8x speedups of the OpenCL and OpenCL/OpenMP implementations of the models associated with phylogetic analysis of continuous and discrete data, respectively. Experiments show that, as expected, OpenCL works well with linear algebra operations involving large matrices while OpenMP provides a better solution for computations involving tree traversal. OpenCL programming require careful resource planning and fine-tuning. Our current framework will allow us to improve the current implementation - by, for example, making better use of local card memory - and extend it's GPU and multi-core functionality to other parts of the code.

ACKNOWLEDGEMENT

Funding: Biotechnology and Biological Sciences (BBSRC). Grant BB/K004344/1.

REFERENCES

Bientinesi,P.,Gunter,B. and Geijn, R. van de (2008) Families of algorithms related to the inversion of a Symmetric Positive Definite matrix. *ACM Transactions on Mathematical Software*, vol. 35, issue 1, pp. 3:1-22.

Felsenstein, J. (1973) Maximum-llikelihood estimation of evolutionary trees from continuous characters *American Journal of Human Genetics*, vol. 25, pp. 471-684.

Freckleton, R. (2012) Fast likelihood calculations for comparative analyses *Methods in Ecology and Evolution*, vol. 3, pp. 940-947.

Pagel, M. (1997) Inferring evolutionary processes from phylogenies *Zoologica Scripta*, vol. 26, pp. 331-348.

Pagel,M.,Meade,A. and Barker, D. (2004) Bayesian estimation of ancestral character states on phylogenies. *Systematic Biology*, vol. 53, pp. 673-684.

Pagel,M. and Meade,A. (2013) BayesTraits V2.0 (Beta). July 2013. http://www.evolution.rdg.ac.uk/BayesTraits.html.