A novel parallel approach to the likelihood-based estimation of admixture in population genetics

Ambra Giovannini, Gaetano Zanghirati, Mark A. Beaumont, Lounès Chikhi, and Guido Barbujani

ABSTRACT

Summary: Inferring population admixture from genetic data and quantifying it is a difficult but crucial task in evolutionary and conservation biology. Unfortunately state-of-the-art probabilistic approaches are computationally demanding. Effectively exploiting the computational power of modern multiprocessor systems can thus have a positive impact to Monte Carlo-based simulation of admixture modeling. A novel parallel approach is briefly described and promising results on its message passing interface (MPI)-based C++ implementation are reported.

Availability: The software package ParLEA is freely available at http://dm.unife.it/parlea.

Contact: ambra.giovannini@unife.it

Supplementary information: Additional information, including instructions for installation/use the original sequential LEA code and the data used in this paper, are also available in the web site.

1 INTRODUCTION

The process of population admixture has received in recent years an increasing attention due to its important implications in research areas such as evolutionary genetics (Belle et al., 2006) and conservation genetics. Likelihood-based approaches for estimating admixture rates are very appealing (Chikhi et al., 2001), but their effectiveness depends on a sufficiently large amount of Monte Carlo simulations. Some computational tools for admixture estimation are available for scalar environment, but they usually require quite long running time (see e.g. Langella et al., 2001). Here, we describe the innovative parallel software ParLEA that speeds-up the estimation of admixture proportions in large genomic datasets.

2 METHODS AND IMPLEMENTATION

The likelihood-based approach to the admixture modeling assumes that two independent parental populations \( P_1 \) and \( P_2 \), sized \( N_1 \) and \( N_2 \), contribute to the gene pool of a new admixed population \( P_H \), sized \( N_H \), with different proportions \( p_1 \) and \( p_2 \). The populations mixed some time \( T \) in the past, measured in generations. The Monte Carlo-based Griffiths–Tavaré method is used to estimate the data likelihood under pure drift using an importance sampling algorithm.

2.1 The novel parallel approach

The parallel code is an extension of the original LEA scalar code by Langella et al. (2001). The parallelization is based on splitting the loci among the available PEs (processing elements): this is because the main loops run over the loci and these computations are independent for the most part of the program. The main parallelization phases are the following:

(1) all processors initialize the parallel environment;
(2) the master PE (root) reads the input files and distributes the input data to other PEs;
(3) the parallel computations start asynchronously on each PE. Locally, each PE computes the likelihood of the data based on the allele frequency in the ancestral population, the corresponding frequency of the admixed population, the admixture contribution of \( P_1 \) and the time since the admixture event, scaled by the population size;
(4) a synchronization point is then needed to allow the root PE to collect the evolution results from each PE and to compute the new parental population frequencies.

Phases 1 and 3 are PE independent and are thus performed in parallel, while phases 2 and 4 require communications from/to the root processor; the first and last phases only need synchronization.

The code exploits the standard MPI (message passing interface) paradigm and it distributes the data among the available processors by partitioning the structure storing the loci in a uniform way and giving to each PE one of these subsets. Hence, all computational loops involving this structure can be performed simultaneously and locally on each PE. Each structure subset does not need to be replicated in the local memory of any other PE: this clearly allows for a much larger number of loci to be loaded by the program into the whole memory available to the bunch of allocated processors.

1440 © The Author 2009. Published by Oxford University Press. All rights reserved. For Permissions, please email: journals.permissions@oxfordjournals.org
3 RESULTS AND DISCUSSION

We tested the ParLEA package on the IBM CLX/1024 Linux cluster at the CINECA Supercomputing Center (Bologna, Italy). This is a strictly coupled distributed-memory MIMD multiprocessor system with each PE being an Intel Xeon 64 bit processor at 3.04 GHz with 1 GB local RAM (see https://hpc.cineca.it/docs/HPCUserGuide for additional details).

The software performances are evaluated by the following standard measures: relative speedup, \( sp_r(N_{pe}) = T_1 / T_{N_{pe}} \); relative efficiency, \( eff_r(N_{pe}) = sp_r(N_{pe}) / N_{pe} \); Kuck’s function, \( f_K(N_{pe}) = sp_r(N_{pe}) eff_r(N_{pe}) / (N_{pe} / N_{pe}) \). Here \( N_{pe} \) is the number of processors, \( T_1 \) is the running time of the parallel algorithm on one processor only and \( T_{N_{pe}} \) is the running time of the parallel algorithm on \( N_{pe} \) processors. In our case, \( T_1 \) is essentially the running time of the original scalar code LEA.

The relative speedup accounts for how much a parallel algorithm is faster than a sequential one. Best performances (i.e. linear speedup) are obtained when \( sp_r = N_{pe} \). Given the communication overhead, this is usually just an ideal level, but sometimes it can be reached and even improved (superlinear speedup) by a very good exploitation of the parallel resources. The relative efficiency estimates how well the given processors are exploited in solving the problem, compared with the overhead due to communications and synchronizations. Best performances are obtained when \( eff_r \) is close to 1, superoptimal behavior when \( eff_r > 1 \). Finally, the Kuck’s function refers to how advantageous the parallel implementation remains as the number of processors increases. The maximizer of the Kuck’s function is interpreted as the largest number of processors suitable for the parallel implementation to solve the given particular problem.

The test problem (available in the web page) we used to benchmark the program has 377 loci and 3357 alleles. The program runs over 50,000 iterations, each one performing 500 Monte Carlo steps. Note that these are the default values in the software and are only used here for the speed tests.

Relative speedup and Kuck’s function are plotted in Figure 1, while the execution times (in seconds) and the relative efficiency are as follows:

<table>
<thead>
<tr>
<th>( N_{pe} )</th>
<th>Sec.</th>
<th>( eff_r )</th>
<th>( N_{pe} )</th>
<th>Sec.</th>
<th>( eff_r )</th>
<th>( N_{pe} )</th>
<th>Sec.</th>
<th>( eff_r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>300.692</td>
<td>8</td>
<td>43.811</td>
<td>0.86</td>
<td>64</td>
<td>8072</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>122.983</td>
<td>1.14</td>
<td>24.533</td>
<td>0.77</td>
<td>128</td>
<td>7000</td>
<td>0.43</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>77.335</td>
<td>0.97</td>
<td>12.755</td>
<td>0.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Speedup, efficiency and Kuck’s function are all quite good up to 32 processors: in this case the table shows that the parallel program is more than 20 times faster than its scalar version (compare the time for \( N_{pe} = 1 \)). We still get quite acceptable performances on 64 processors: the execution time is about 2 h and 15 min, against almost 4 days for the scalar version. However, the Kuck’s function clearly shows that up to 32 PEs can be effectively exploited by the parallel program on the given simulations. Little improvement is obtained by providing additional PEs, even though the performance degrades because the computational load on each PE becomes too small (too few loci) compared with the communication time. The final results are qualitatively comparable with the results of the scalar code, even if some decimal digits are different because the current random numbers generator does not allow interprocessor control of the sequence (see the web site for more details).

4 CONCLUSIONS

We presented a novel parallel approach for estimating proportions of population admixture, based on a well-established likelihood-based method. The parallel program implementing the proposed approach showed to be very effective on a quite large, meaningful test problem. The coarse-grain parallelization strategy is based on a suitable data and loops distribution. The very promising performances we obtained will allow much larger datasets to be analyzed in reasonable time. Moreover, they encourage further improvements: future developments will include a careful storage-structures revision as well as the use of strongly trusted random numbers generation libraries, suitable for distributed memory parallel environments.

ACKNOWLEDGEMENTS

The authors thank E. Belle for useful discussions at the early stages of the project. The authors are also greatly thankful to I. Girotto and G. Erbacci of the CINECA’s Supercomputing Team, for their extremely effective support.

Conflict of Interest: none declared.

REFERENCES

