Approaching Long Genomic Regions and Large Recombination Rates with msParSm as an Alternative to MaCS

Carlos Montemuiño[1], Antonio Espinosa[1], Juan C. Moure[1], Gonzalo Vera[2], Porfidio Hernández[1] and Sebastián Ramos-Onsins[2]

[1]Computer Architecture and Operating Systems Department (CAOS), Universitat Autònoma de Barcelona, Bellaterra, Spain. [2]Centre for Research in Agricultural Genomics (CRAG) Consortium CSIC-IRTA-UAB-UB Edifici CRAG, Campus UAB, Bellaterra, Spain.

ABSTRACT: The msParSm application is an evolution of msPar, the parallel version of the coalescent simulation program ms, which removes the limitation for simulating long stretches of DNA sequences with large recombination rates, without compromising the accuracy of the standard coalescence. This work introduces msParSm, describes its significant performance improvements over msPar and its shared memory parallelization details, and shows how it can get better, if not similar, execution times than MaCS. Two case studies with different mutation rates were analyzed, one approximating the human average and the other approximating the *Drosophila melanogaster* average. Source code is available at https://github.com/cmontemuino/msparsm.

KEYWORDS: coalescence, recombination, sequential Markov coalescent, HPC, MPI

CITATION: Montemuiño et al. Approaching Long Genomic Regions and Large Recombination Rates with msParSm as an Alternative to MaCS. *Evolutionary Bioinformatics* 2016:12 223–228 doi: 10.4137/EBO.S40268.

TYPE: Software or Database Review

RECEIVED: May 31, 2016. **RESUBMITTED:** July 19, 2016. **ACCEPTED FOR PUBLICATION:** July 21, 2016.

ACADEMIC EDITOR: Liuyang Wang, Associate Editor

PEER REVIEW: Two peer reviewers contributed to the peer review report. Reviewers' reports totaled 450 words, excluding any confidential comments to the academic editor.

FUNDING: This work has been supported by projects (number: AGL2013-41834-R and TIN2014-53234-C2-1-R) of Spanish Ministerio de Ciencia y Tecnologia. The authors confirm that the funder had no influence over the study design, content of the article, or selection of this journal.

COMPETING INTERESTS: Authors disclose no potential conflicts of interest.

CORRESPONDENCE: cmontemu@acm.org

Introduction

The field of population genetics endeavors to ascertain how basic evolutionary forces, such as natural selection, recombination, and mutation, shape the patterns of genetic variation within and between populations.[1,2] Computer simulation software has traditionally been used to explore analytically intractable genetic models.[3,4] Continuous advances in numerical simulation and the wide availability of computational resources allow researchers to use numerical simulation to test mathematical models in virtual populations, and even to analyze genetic data.[5,6] As a direct consequence, there is a plethora of simulators available, each one tailored to a specific scenario. This forces geneticists to choose a simulator depending on the research being conducted.[3]

The challenge is to provide flexible simulation programs with the capacity for dealing with complex and realistic demographic/evolutionary models, using efficient algorithms to deal with genome-wide data sets processing complexity.[4,7,8] The availability of cost-effective next-generation sequencing technologies has democratized whole-genome sequencing, making large genomic data sets available to most researchers.[2,9,10] Among the two approaches to simulation algorithms, forward-in-time and coalescent-based, the latter is most widely used because of its efficiency and flexibility.[3,11,12] In particular, the standard coalescent approach was shown to be extremely efficient for short sequences (less than few mega base pairs), but it becomes computationally demanding for simulating long genome regions with large recombination rate.[13] New approaches have emerged to overcome this issue, such as the sequential Markov coalescent and Markov Chain Monte Carlo methods[5,8,14–16] and alternative implementations of the exact coalescent, including algorithmic and data structure optimizations, such as scrm and msprime.[17,18]

In this article, we focus on coalescent simulators, which are shown to be computationally intractable when working at the genome scale,[8,11,19] and more specifically on msPar,[20] which is the parallel version of Hudson's ms coalescent simulator,[21] ie, the most classical and widely used coalescent simulator.[3,7,22] msPar addresses both the problem of sampling long genomic regions with large recombination rates and running analysis requiring a large number of samples.[20] We have previously shown that msPar execution time is orders of magnitude faster than Hudson's ms when simulating large samples of long genomic sequences and large recombination rates using a High Performance Computing (HPC) cluster,[20] but it is less efficient than the other approximated standard coalescent simulators such as MaCS and fastsimcoal.[13–15]

Our interest in further developing the standard coalescent simulators resides in the underlying side effects shown by approximated methods, like the one implemented by MaCS:

1. We can suffer loss of accuracy for certain evolutionary models as when sampling from populations separated by reduced gene flow.[23]
2. Ignoring type 2 recombination events (from Marjoram and Wall's classification[24]) may impact on the mean and variance of most recent common ancestor times when long sequences are simulated.[25]

Regarding the implementation details, we have considerably improved msPar's memory management, improved its master–worker implementation, removed the data structure dependencies inherited from Hudson's ms, and made use of the Vader shared memory byte transport layer (BTL) provided by OpenMPI v1.8.[26] By applying a message-passing/shared-memory parallelization strategy, msParSm is able to achieve similar execution times to those of MaCS, simulating long stretches of DNA sequences with large recombination rates. Thus, we significantly reduce the computational burden of the standard coalescent method. For the purpose of comparison, we have used two case studies, with fixed mutation rates approximating the human and *Drosophila melanogaster* average values.

Accuracy of simulations is maintained to be the same as the one from Hudson's ms. It is ensured by the fact that the code in charge of sample generation was maintained unchanged from original Hudson's ms code.

It is also important to note that msParSm, as same as msPar and Hudson's ms, only generates samples under a Wright–Fisher neutral model of genetic variation.

Methods

In this section, we describe the parallelization approach taken over msPar, the experimental setup we designed for evidence gathering, and the hardware/software used to run the experiments.

Parallelization approach. We present a new master–worker application design that includes the following: removing the data structure dependencies inherited from Hudson's ms, refactoring msPar's memory management, and improving the communication patterns used in the msPar's master–worker implementation.

First, we have taken the advantage of the OpenMPI's new transport layer (Vader shared-memory BTL) for transferring data between communication end points, which was introduced in version 1.8. The Message Passing Interface (MPI) transport layer used by msPar is known as sm BTL (shared-memory BTL), which follows a copy-in/copy-out pattern: when an MPI process X sends a message to a process Y, the message is first copied from the X's buffer to the shared memory, and then the receiver (ie, the process Y) copies the message from the shared memory into a buffer.[27] The Vader shared memory BTL provides support for XPMEM Linux kernel module and allows to directly transfer messages between sender and receiver buffers when both MPI processes are in the same node, thereby saving one copy operation compared against the previous transport layer used included in OpenMPI (ie, sm BTL).

To take the advantage of Vader BTL, we have changed the master–worker topology used by msPar. Given N as the total number of available processes for parallel computation, we define one single global master process coordinating the N–1 remaining worker processes. In msParSm, we have one master process per compute node (henceforth referred to as node master) and one global master process that coordinates the M node masters (where M is the number of compute nodes). Each master node coordinates the worker processes located in its node, returning a consolidated message with all of the generated samples back to the global master when computation is performed. It is important to note that there is going to be one node hosting two master processes: the global master and the node master.

By using non-blocking MPI collectives, we enabled node masters to also generate samples while waiting for local worker communications; therefore, the loss of worker processes compared with msPar is compensated as much as possible. In a setup with M compute nodes and P cores per node, in msPar, we have a total number of $M^*P - 1$ *dedicated worker processes*, while in msParSm the number is $M^*(P - 1) - 1$.

For the use case of the scientist using a single node for the computation, eg, a single fat node or even a workstation, the application is not going to use a global master but a single master that is also going to generate replica samples.

Another improvement we made is to allow MPI processes located in the same compute node as the global master, to directly output the generated samples avoiding the use of MPI point-to-point communications with a master process.

Summarizing the major improvements, we

- greatly decreased the number of memory allocation calls by favoring memory reusing and reallocation as much as possible;
- refactored the master–worker strategy to leverage the use of the Vader BTL;
- included the master process in the replica generation process;
- made worker processes more autonomous to reduce the number of MPI communications;
- removed all global structures inherited from Hudson's ms, making it possible to explore a fine-grained parallelization approach (eg, using OpenMP and/or CUDA).

Experimental setup and case studies. We compared our msParSm with Hudson's ms and MaCS in terms of execution time, using two case studies with fixed population mutation rate values (ie, $\theta = 4N\mu$, where N is the current population size and μ is the mutation rate per site), one of them approximating the estimated human average (case 1) and the other one approximating the estimated *D. melanogaster* average

(case 2; $\theta = 0.001$ and 0.005 values, meaning that 1 and 5 are differences between two random individuals for each 1000 sites, respectively).

The number of generated replicas was set at 300, each one with 100 chromosomes, a genetic region of 1×10^6 bp, and population recombination per site ($4N\rho$, where ρ is the mutation rate between two contiguous sites) *ranging* from 0.0005 to 0.08 values. Both evolutionary models exhibit no population genetic structure.

The parameters used in each of the simulations performed in this work are given in Supplementary Table 1.

Hardware and software. We implemented the parallel application in ANSI C and compiled it using the Linux GNU Compiler 4.9.1. Inter-process communication was implemented using OpenMPI 1.10.1.[28] We ran the binaries on a cluster using up to eight nodes each with two Intel Xeon X5660 six-core processors using hyper-threading technology running at 2.80 GHz, 12 MB L3 cache, and 96 GB of Double data rate synchronous dynamic-random access memory (DDR-RAM). This configuration provided us 12 physical processors per compute node.

Results and Discussion

We evaluated the performance of msParSm by analyzing the speedup and efficiency as a function of increasing numbers of compute nodes. The speedup is defined as $S_p = T_1/T_p$, where p is the number of processors, T_1 is the execution time of the sequential application, and T_p is the execution time of the parallel application with p processors. Efficiency is defined as $E_p = S_p/p$.

In Figure 1, the speedup of msParSm compared with the sequential Hudson's ms version is shown, which was registered in case study 1. We observed a high parallel efficiency of msParSm and a strong increase in speed in relation to sequential Hudson's ms version. Nevertheless, for the case of maximal recombination (Fig. 1D), we observed a slight performance degradation. This happened because the node memory was mostly exhausted by the MPI processes (90 GB in use out of 96 GB available), implying a penalty because of the page swapping.

In Figure 2, the same graphic data as before are shown, but for the case study 2. Compared with the case study 1, we observed a worse performance for lower recombination rate (Fig. 2A), but there was a similar performance for upper values (ie, 2000, 4000, and 8000). An explanation for this behavior is associated with the message payload exchanged between MPI processes. Average message with sample data is 4.5 times higher for the case study 2, suggesting that the time spent in communication is less relevant as long as the recombination rate increases.

In Figure 3, the average consumed memory in function of the recombination rate and cores used for the computation is shown, which is registered in case study 1. In Figure 4, the same exact data are shown, but for the case study 2. From both figures, we can observe how the node's main memory is exhausted when the maximal recombination rate is reached (Fig. 4D), irrespective of the number of involved cores, explaining why the efficiency is impacted given the current experimental setup.

The most important contribution for the geneticist is the reduction in the overall running time of the simulation process. In Table 1, the average time Hudson's ms, MaCS, msPar, and msParSm spent running both case studies using different problem sizes (ie, recombination rate values) is given. We observe that how the execution time of msParSm is orders of magnitude faster than Hudson's ms, faster by a factor than msPar, and significantly better than MaCS in most configurations, mainly when mutation rate is higher.

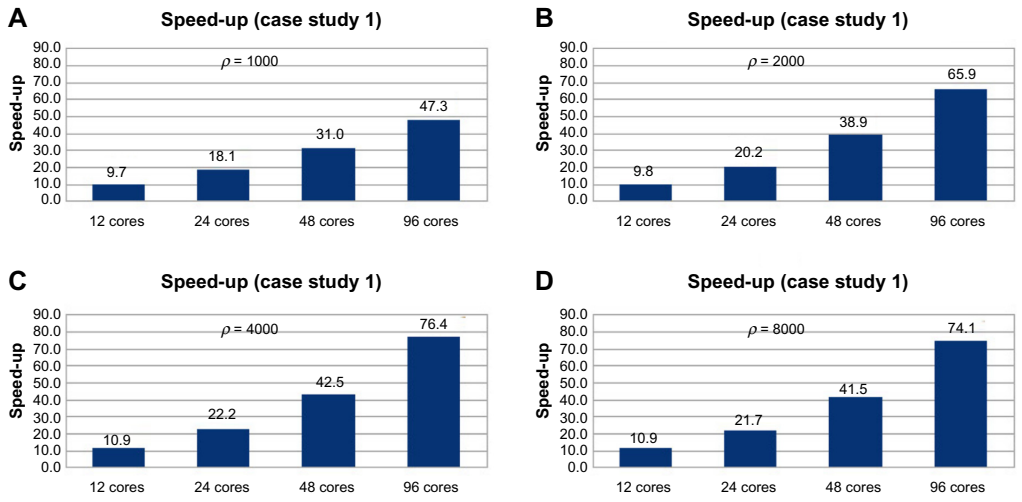

Figure 1. Speedup analysis of case study 1 in function of the number of cores used for computation, grouped by a scaled recombination rate (**A** to **D**), registered in case study 1. Subfigures show speedup of the application increases adding more computational cores, slowly decaying when considering recombination rates of $4N\rho = 8000$.

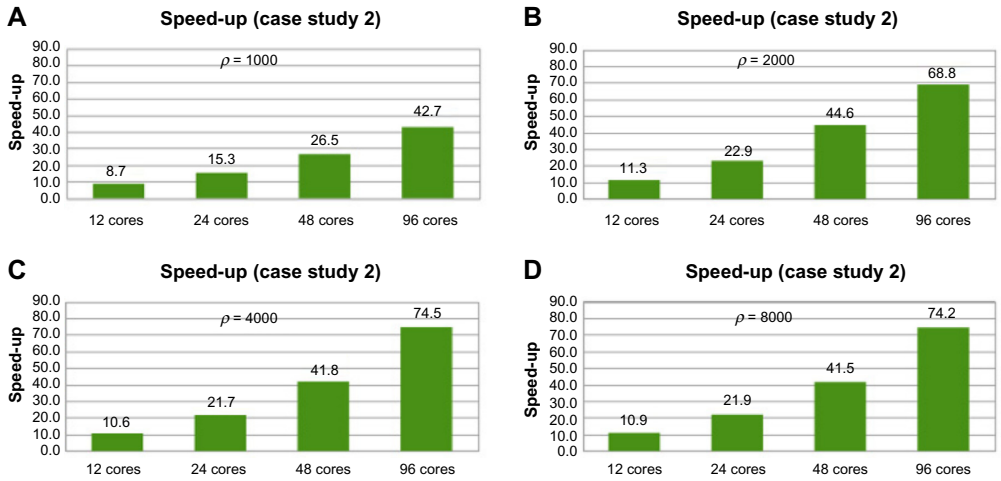

Figure 2. Obtained speedup of case study 2 in function of the number of cores used for computation, grouped by used recombination rate (**A** to **D**). As in the previous case study 1, this figure shows increasing speedups when adding more computational resources.

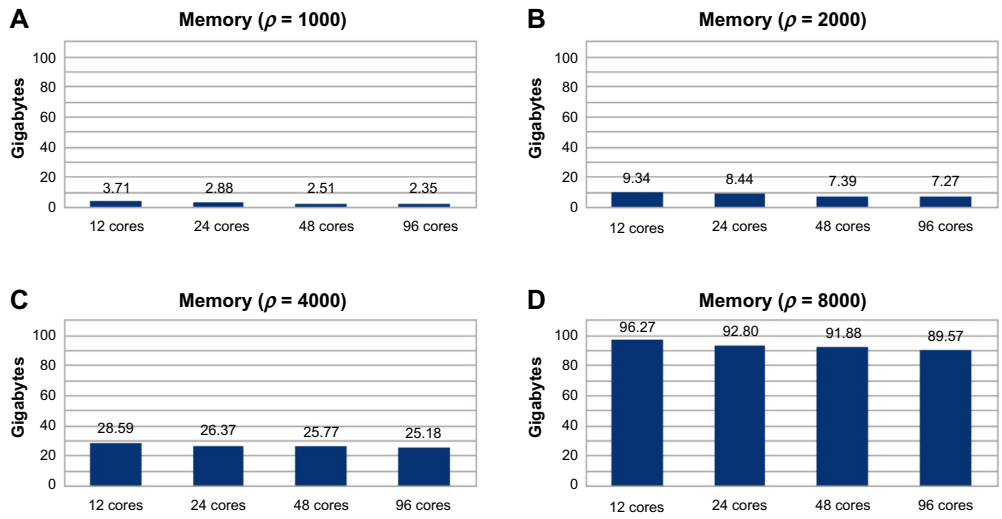

Figure 3. Average consumed memory for case study 1 in function increasing the number of computational cores and recombination values, registered in case study 1. Each subfigure shows the consumed memory values when running the simulation using 12, 24, 48, and 96 cores, starting with a scaled recombination of $4N\rho = 1000$ in (**A**), and doubling it through the remaining subfigures (**C** to **D**) until reaching a scaled recombination of $4N\rho = 8000$.

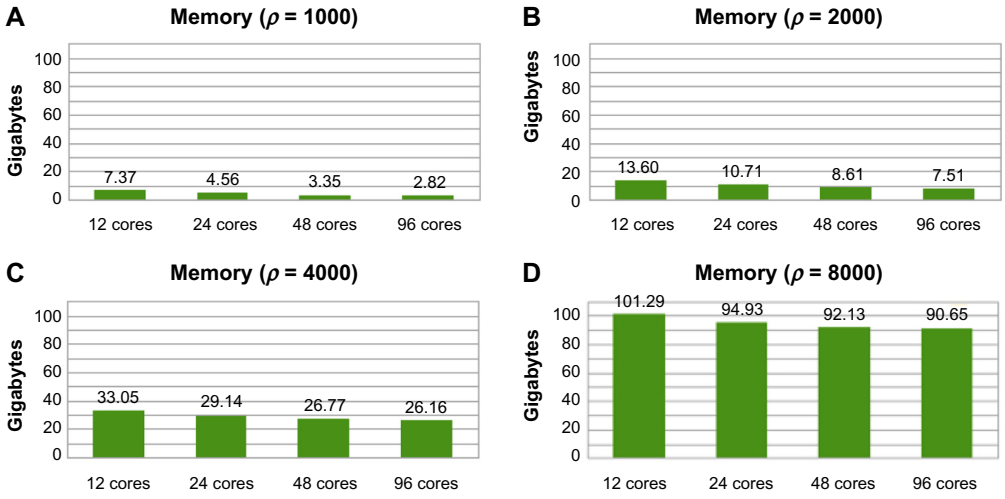

Figure 4. Average consumed memory for case study 2 in function increasing the number of cores used for computation. Each subfigure (**A** to **D**) shows the same information as in Figure 3, but with data registered when running the case study 2.

Table 1. Comparison of average time cost (in minutes) between ms, MaCS, msPar, and msParSm. The execution times of both msParSm and msPar are grouped in function of the number of cores used for computation. A combination of font styles and background color is used to facilitate the reading: values in italics are related to msPar, while bold style is used for msParSm; shaded table cells indicate cases that the execution time of either msParSm or msPar is worse than MaCS.

Rho	Ms	MaCS	msParSm/*mspar*							
			96 CORES		48 CORES		24 CORES		12 CORES	
Case study 1										
1000	3.81	3.82	**0.08**	*0.44*	**0.12**	*0.44*	**0.21**	*0.64*	**0.40**	*0.59*
2000	29.77	7.16	**0.45**	*2.75*	**0.76**	*2.75*	**1.47**	*3.91*	**3.05**	*3.42*
4000	322.80	13.76	**4.23**	*19.57*	**7.60**	*26.07*	**14.57**	*28.48*	**29.64**	*30.52*
8000	2605.58	26.86	**35.15**	*149.48*	**62.78**	*214.36*	**120.02**	*223.98*	**240.12**	*249.5*
Case study 2										
1000	3.95	5.80	**0.22**	*1.14*	**0.21**	*1.22*	**0.26**	*1.61*	**0.44**	*1.37*
2000	34.85	9.14	**0.51**	*3.33*	**0.78**	*3.53*	**1.52**	*4.89*	**3.09**	*4.23*
4000	320.30	15.88	**4.30**	*20.02*	**7.67**	*27.61*	**14.78**	*29.43*	**30.12**	*32.03*
8000	2602.00	29.32	**35.05**	*151.35*	**62.75**	*222.60*	**119.03**	*224.18*	**239.47**	*249.27*

Although the parallel execution of msParSm is behind MaCS, the scalability of presented msParSm is generally good when adding more computational resources with the possible limitation of the available memory of the nodes.

Conclusion

In this work, we developed a new application for parallelizing the Hudson's ms, a standard coalescent simulator, coined as msParSm, addressing the sampling of long stretches of DNA sequences with large recombination rates, and we showed that this new application has comparable performance with MaCS when working with large recombination rate, without the side effects of MaCS.

Interestingly, the application achieves high parallel efficiency figures (>70%) when working with large recombination rate, suggesting that it could be run on more nodes than used for this work and get better execution times. In addition, our application can outperform MaCS in most of the cases, even when running the application in one single node.

We believe that this new application will facilitate simulating coalescent processes with long genomic regions and large recombination rates in population genomics and evolutionary biology, without compromising the accuracy of the standard coalescence.

Author Contributions

Conceived and designed the experiments: SRO, CM. Analyzed the data: CM. Wrote the first draft of the manuscript: CM. Contributed to the writing of the manuscript: PH, SRO. Agreed with the manuscript results and conclusions: SRO, PH, AE, GV, JCM, CM. Jointly developed the structure and arguments for the article: SRO, PH, CM. Made critical revisions and approved the final version: SRO, PH, AE, GV, JCM. All the authors reviewed and approved the final manuscript.

Supplementary Material

Supplementary Table 1. Parameters used in each of the simulations performed.

REFERENCES

1. Hudson R. Gene genealogies and the coalescent process. In: Futuyama D, Antonovics J, eds. *Oxford Survey in Evolutionary Biology*. Vol 7. Oxford: Oxford University Press; 1991:1–44.
2. Akey JM, Shriver MD. A grand challenge in evolutionary and population genetics: new paradigms for exploring the past and charting the future in the post-genomic era. *Front Genet*. 2011;2:47.
3. Hoban S, Bertorelle G, Gaggiotti OE. Computer simulations: tools for population and evolutionary genetics. *Nat Rev Genet*. 2011;13(2):110–22.
4. Arenas M. Simulation of molecular data under diverse evolutionary scenarios. *PLoS Comput Biol*. 2012;8(5):e1002495.
5. Sanford J, Baumgardner J, Brewer W, Gibson P, ReMine W. Mendel's accountant: a biologically realistic forward-time population genetics program. *Scalable Comput Pract Exp*. 2001;8(2):147–65.
6. Sanford J, Nelson C. The next step in understanding population dynamics: comprehensive numerical simulation. In: Fust MC, ed. *Studies in Population Genetics*.InTech;2012.Availableat:http://www.intechopen.com/books/studies-in-population-genetics/the-next-step-in-understanding-population-dynamics-comprehensive-numerical-simulation. Accessed March 29, 2016.
7. Carvajal-Rodríguez A. Simulation of genomes: a review. *Curr Genomics*. 2008;9(3):155–9.
8. Carvajal-Rodríguez A. Simulation of genes and genomes forward in time. *Curr Genomics*. 2010;11(1):58–61.
9. Liu DJ, Leal SM. Replication strategies for rare variant complex trait association studies via next-generation sequencing. *Am J Hum Genet*. 2010;87(6):790–801.
10. Pool JE, Hellmann I, Jensen JD, Nielsen R. Population genetic inference from genomic sequence variation. *Genome Res*. 2010;20(3):291–300.
11. Kim Y, Wiehe T. Simulation of DNA sequence evolution under models of recent directional selection. *Brief Bioinform*. 2009;10(1):84–96.
12. Peng B, Kimmel M. simuPOP: a forward-time population genetics simulation environment. *Bioinformatics*. 2005;21(18):3686–7.
13. Yang T, Deng H-W, Niu T. Critical assessment of coalescent simulators in modeling recombination hotspots in genomic sequences. *BMC Bioinformatics*. 2014;15:3.
14. Chen GK, Marjoram P, Wall JD. Fast and flexible simulation of DNA sequence data. *Genome Res*. 2009;19(1):136–42.
15. Excoffier L, Foll M. Fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. *Bioinformatics*. 2011;27(9):1332–4.
16. Grünwald NJ, Goss EM. Evolution and population genetics of exotic and re-emerging pathogens: novel tools and approaches. *Annu Rev Phytopathol*. 2011;49(1):249–67.

17. Staab PR, Zhu S, Metzler D, Lunter G. scrm: efficiently simulating long sequences using the approximated coalescent with recombination. *Bioinformatics*. 2015;31(10):1680–2.
18. Kelleher J, Etheridge AM, McVean G. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS Comput Biol*. 2016;12(5):e1004842.
19. Liang L, Zöllner S, Abecasis GR. GENOME: a rapid coalescent-based whole genome simulator. *Bioinformatics*. 2007;23(12):1565–7.
20. Montemuiño C, Espinosa A, Moure JC, et al., eds. MsPar: a parallel coalescent simulator. *Euro-Par 2013: Parallel Processing Workshops*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer; 2013:321–30. Available at: http://link.springer.com/chapter/10.1007/978-3-642-54420-0_32. Accessed February 24, 2016.
21. Hudson RR. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*. 2002;18(2):337–8.
22. Ewing G, Hermisson J. MSMS: a coalescent simulation program including recombination, demographic structure and selection at a single locus. *Bioinformatics*. 2010;26(16):2064–5.
23. Eriksson A, Mahjani B, Mehlig B. Sequential Markov coalescent algorithms for population models with demographic structure. *Theor Popul Biol*. 2009;76(2):84–91.
24. Marjoram P, Wall JD. Fast "coalescent" simulation. *BMC Genet*. 2006;7:16.
25. Wang Y, Zhou Y, Li L, et al. A new method for modeling coalescent processes with recombination. *BMC Bioinformatics*. 2014;15:273.
26. FAQ: Tuning the Run-Time Characteristics of MPI Sm Communications. Available at: https://www.open-mpi.org/faq/?category=sm. Accessed March 25, 2016.
27. Hjelm NT, Gutierrez SK, Gorentla Venkata M. *On the Current State of Open MPI on Cray Systems*. Oak Ridge National Laboratory (ORNL); Oak Ridge Leadership Computing Facility (OLCF); 2014. Available at: http://www.osti.gov/scitech/biblio/1150898-current-state-open-mpi-cray-systems. Accessed April 4, 2016.
28. Open MPI: Open Source High Performance Computing. Available at: https://www.open-mpi.org/. Accessed March 29, 2016.