msPar: A Parallel Coalescent Simulator

Carlos Montemuiño[1], Antonio Espinosa[2], Juan-Carlos Moure[3],
Gonzalo Vera-Rodríguez[4], Sebastián Ramos-Onsins[4], and Porfidio Hernández Budé[3]

[1] Universitat Autònoma de Barcelona, Bellaterra, Spain
cmontemu@acm.org
[2] Universitat Autònoma de Barcelona, Bellaterra, Spain
antonio.espinosa@caos.uab.es
[3] Universitat Autònoma de Barcelona, Bellaterra, Spain
{juancarlos.moure,porfidio.hernandez}@uab.es
[4] Centre de Recerca en Agrigenòmica, Bellaterra, Spain
{gonzalo.vera,sebastian.ramos}@cragenomica.es

Abstract. We implemented a parallel version (hereafter referred as "msPar") of the coalescent simulation program ms, providing the same functionality and output, parallelized using a Master-Worker scheme with on-demand scheduling and MPI to run on an HPC cluster. To our knowledge this is the first time such parallelization has been applied to ms, and shown to be effective in using all computational resources of an HPC cluster, performing up to 42 times faster than original ms when using 72 logical processors. We propose msPar as an alternative to ms and other simulators using approximations to the standard coalescent approach. Source code is available at https://github.com/cmontemuino/mspar

Keywords: HPC, parallel, master-worker, MPI, coalescent.

1 Introduction

Population genetics, a field of biology that studies the molecular diversity within and between species [1], is playing a major role in human genetic research linking hypothesis on sequence variation with empirical observations [2].

As genome-scale data is becoming a common place in genetic research, disagreements between expectation and observation have become clear [3], forcing geneticist to work with more complex evolutionary scenarios [4].

Simulation software is the key tool used in population genetics to generate genetic data. Much effort is continuously put in designing efficient simulation programs that aim to deal both with genome-scale data and complex evolutionary models [2][4].

The more complex the evolutionary model, the less efficient the simulation and vice versa [4]. From the two types of simulation applications, coalescent-based (backward in-time) and forward in-time, the standard coalescent approach shown to be computationally intractable when working at genome scale [4][5][6].

The common approach to overcome this issue is to design new algorithms, compromising the flexibility for the sake of efficiency, but some authors are starting to make use of new technologies and computer clusters [4][7], as with forward in-time simulators, SimuPop [8], Nemo [9] and Mendel's Accountant [10].

Parallel programming is a promising approach to improve the performance of a broad range of applications, especially bioinformatics applications. In this paper we present an approach to parallelize a coalescent simulator, enabling it to be run in an HPC cluster. We parallelized the Hudson's *ms* coalescent simulator [11], the most classical and widely used coalescent simulator [2][12][13], not only by the community research, but also as a point of comparison when new coalescent simulators are released. We coined this simulator as *msPar*.

We propose to parallelize the coalescent simulator *ms*, without affecting its flexibility when working at genome-scale or to perform analysis requiring high number of iterations, as long as more computational resources are available. We measured execution time of *ms* to generate 528 replicas in an HPC cluster, with a mutation rate of 640 and a recombination rate of 5120 (both scaled to 4N generations), with a sample size of 200 chromosomes and a region of 1e6 bp, resulting in 24 hours. Doubling the recombination rate to 10240 we found *ms* took 7.75 days to complete.

The organization of the article is as follows. In next section we present the background and related work. In section 3, we present the *ms* application and describe its structure. Section 4 describes our proposed solution. We present and discuss the performance results in section 5, and end with some conclusions from this research in section 6.

2 Related Work

Researching in genetics population field was supported by mathematical modeling for over 75 years. Empirical testing of theoretical models is practically impossible for organisms with long generation times. Continuous advances in numerical simulation and wide availability of computational resources allow researchers to use numerical simulation to test mathematical models in virtual populations, and even to analyze genetic data [10][14]. As a direct consequence there is a high number of simulators available, each one tailored to a specific scenario, implying geneticists must decide which simulator to use depending on the research being conducted [12].

From the two approaches to simulation algorithms, forwards in-time (also known as individual based-simulation) and backwards in-time (also known as coalescent simulation), coalescent simulations are most widely used because of its efficiency and flexibility [5][8][12]. Forward in-time simulations have been mostly used for cases where coalescent approach does not suit, as to observe the evolution of allelic spectra from the founder to the current generation [15], when the interest is focused on the evolutionary process itself [2], or teaching purposes [7].

The standard coalescent approach showed to be extremely efficient for short sequences, but becomes computationally demanding and very slow for large genome regions (> 100Mb) with increasing recombination rate. New approaches emerged to overcome this issue, as the sequentially Markov coalescent [4][16][17] and Markov chain Monte Carlo [18]. Among other coalescent simulators developed last years

(GENOME [6], mlcoalsim [19] and MSMS [13]), MaCS [16] and fastsimcoal [17] appeared to be the most efficient ones for large genomic regions and high recombination rates.

3 MS Coalescent Simulator

Hudson's *ms* [11] is a Monte Carlo application widely used by population geneticists and molecular biologists to analyze populations and DNA sequences, and estimate demographic and mutational parameters from a theoretical population evolving under a neutral Wright-Fisher model. Implementing a continuous-time approximation of the coalescent process, *ms* bases its processing in first generating an ARG (Ancestral Recombination Graph) for an initial sample of chromosomes and then placing random mutations over the resulting ARG. The ARG is constructed by stochastically simulating evolutionary events (such as coalescence, migration or recombination) occurring on the ancestral material of sampled alleles backwards in time, until the MRCA (Most Recent Common Ancestor) to all the alleles is found.

We show in Fig. 1 global replica generation process. Starting from bottom, each replica sample is independently and sequentially generated by an orchestration module split up in two independent steps: Genealogy Construction and Mutation Assignment. Genealogy construction is driven by three routines simulating a random set of recombination, migration and coalescence events. The mutation assignment step works over the ARG produce by step 1, generating the different segregation loci in order to produce the final chromosomes. The stochastic feature of the coalescence process is based on an implementation of the Monte Carlo method.

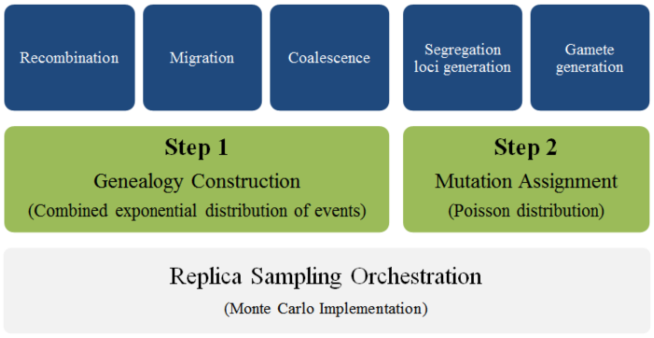
Recombination
Migration
Coalescence
Segregation loci generation
Gamete generation
Step 1
Genealogy Construction
(Combined exponential distribution of events)
Step 2
Mutation Assignment
(Poisson distribution)
Replica Sampling Orchestration
(Monte Carlo Implementation)

Fig. 1. Schematic representation of the Hudson's ms process to generate a set of replica samples

The first step is to construct the ARG and put it into an array-based data structure. In Fig. 2, we show how the history of each chromosome's segment is represented by the ARG nodes. Each node points to the start segment, the end segment, history is represented by stating the ARG node this node is the parent and the ARG node this node is a child. This structure is shared by the three event routines from step 1 (recombination, migration and coalescence), updating it as soon as one event occurs.

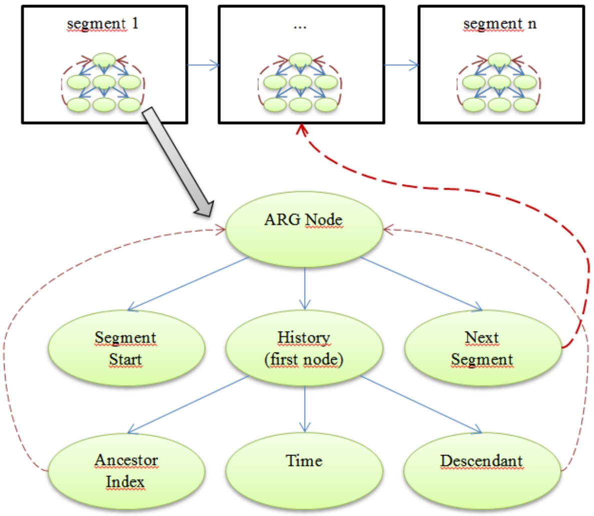

Fig. 2. Data structure of the ARG node. Given n samples, the first n nodes are the tips of the tree. The remaining nodes are the ancestral to the sampled chromosomes.

The complete ARG structure is the input of step 2, where mutations are randomly assigned (following a Poisson distribution) and the replica samples data is finally produced.

The spatial and temporal complexity of *ms* is mainly affected by the population size (n), region (l), and two weighting parameters greater than one, w_1 and w_2, that incorporate the effect of recombination and mutation rates. Temporal complexity is $O(nl + w_1n + w_2l)$ and spatial complexity is $O(nl)$ [20].

3.1 Sequential Application Characterization

In this section we want to show how computation is distributed among the algorithm steps and the output process of the sequential application, in order to determine whether a fine-grained parallelization approach could make sense. We measured the time spent by the routines related to the algorithm steps and took note of the memory consumption of the whole application execution. The test environment was a box with two Intel Xeon X5660 six-core processor with Hyper-Threading Technology running at 2.80 GHz, 12MB L3 cache, and 96 GB of DDR-RAM

In Fig. 3, we show how much time is spent by the algorithm steps and I/O processing. As long as the recombination rate is bigger, it is the percentage of total time needed to construct the ARG due to the occurrence of more events to find the MRCA. The mutation rate is what determines the size of the output, meaning more I/O processing time, but the sum of the mutation assignment step and I/O processing quickly becomes practically negligible when the recombination rate is bigger than 400.

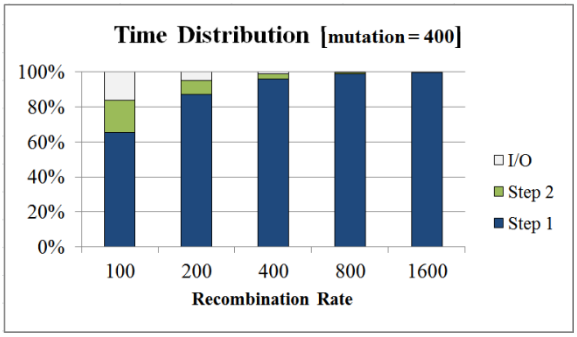

Fig. 3. Time spent by ARG construction step, mutation assignment step, and input/output processing

In Fig. 4, we show how much memory the application uses to generate one single replica as a function of recombination and mutation rates. We observed that mutation rate does not impact memory usage, irrespectively of the recombination rate value. This happens because all data structures are created at genealogy construction step (where recombination events take place), and the mutation rate applies to the mutation assignment step where no new data structures are created.

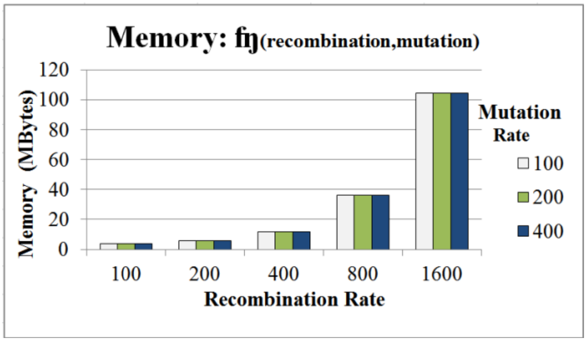

Fig. 4. Maximum resident set size as a function of recombination and mutation parameters

4 Proposal

For analysis requiring millions of population replica samples, each one independently generated, means sequentially running steps 1 and 2 from *ms* millions of times. The result of each of these computations does not depend on the results from any other computation. This is a pleasingly[1] parallel problem [21][22][23].

The high degree of data dependency among routines from step 1 (e.g. shared most variables with global scope), and between steps 1 and 2 prevents the use of a fine-grained parallelization approach without completely refactoring the source code.

[1] Also termed as “embarrassingly”.

We propose to apply the Master-Worker pattern to parallelize the *ms* application, a widely used form of parallel application programming, and a natural fit for Monte Carlo applications as well [24][25].

The replica sampling orchestration from Fig. 1 assigned to the master. Its algorithm is as follow:

```
masterProcessingLogic(){
  initialize and distribute RNG seeds to workers
  while there are replicas to generate {
    find an idle worker
    if there is an idle worker {
      assign work to worker
    } else {
      retrieve generated replica from workers
    }
  }
}
```

The master also maintains the idle workers pool. In the beginning, all workers are idle and the pool is full. When one replica is assigned to one idle worker, the worker is removed from the pool. And after one worker has generated its assigned replica, it sends a message to the master and master adds this worker to the pool. When the master reads the generated replica sent by workers, it determines if there are more replicas to be generated and send a signal to the worker to let it know whether to wait for more request or to stop working.

Each worker is waiting for a request of replica generation, and then run both step 1 and 2 (see Fig. 1) to generate one replica and then transmits it back to the master. Following is the pseudo-code for worker's processing:

```
workerProcessingLogic(){
  receive seeds and initialize local RNG
  read experimentation parameters
  while worker is active {
    run step 1
    run step 2
    send replica to master
    receive activation signal
  }
}
```

The inline commands and output is the same as in the original *ms*, but includes additional arguments to setup the parallelization strategy. If the target environment consists of m cores, the parallelization is performed by dividing the N replicas evenly amongst m-1 cores, and remaining core plays the role of the orchestrator.

Master-Worker ecosystem is approached with the MPI model, to spawn worker processes and map them to hardware processors in the system, each one using its own local memory. We specifically use MPI parallel library [26].

Quality of random number streams employed by the master-worker is guaranteed by using the RNG (Random Number Generator) twice. The RNG seeds specified as input parameters are used to initialize the master's RNG. Then the master generates a set of random numbers ultimately used by each worker as seeds to initialize their own RNG.

5 Analysis and Results

In this section we give a description of the experimental setup for gathering and analyzing based on an HPC cluster.

5.1 Experimental Setup

We compare our application with respect to *ms* in terms of execution time performance metric.

Test cases were designed to be simple enough (i.e. without population structure), taking into consideration only recombination, mutation, genetic region and population size, focusing our attention on the performance evolution when the recombination ratio changes. The mutation rate is not the principal factor affecting execution time (see Fig. 3), but as it is still important in the standard coalescent process, we decided to use a scaled[2] mutation rate of 640, that being not too high, it is big enough to let the mutation assignment step to get some computation.

We selected a population size of 200 chromosomes and a region of 1e6 bp (base pairs), considered by geneticists as big for genetic analysis and quite close to what is required in genomic analysis [16].

If evolutionary parameters do not change from one replica to another (as it is in this setup), then differences in the execution time and memory consumption for each replica generation will be negligible. For the recombination rate we decided to start with a scaled value of 2560 because combined with the other parameters makes *ms* consume enough memory (1.3 GB). We doubled it until we reached 10240, which is considered a very big recombination rate.

The number of generated replicas was set in 528 for two reasons: first we wanted ms to execute during enough time (2.75 hours in the minimal setup) for measuring, and second because it allows our Master-Worker to distribute work fairly well in the case of 72 workers.

[2] All neutral parameters are given by per-site rates 4N, where N is the current population size.

5.2 Hardware and Software

We implemented the parallel application in ANSI C, and compiled it using the Linux GNU Compiler 4.4.6. Inter-process communication was implemented using OpenMPI [27] 1.4.3.

We run the binaries on a cluster using up to three nodes each with two Intel Xeon X5660 six-core processor with Hyper-Threading Technology running at 2.80 GHz, 12MB L3 cache, and 96 GB of DDR-RAM. This configuration gives us 12 physical processors per compute node, but due to the Hyper-Threading technology we can get 24 logical processors per node.

5.3 Discussion

As a way to evaluate the *msPar* performance we investigated the speedup and efficiency as a function of increasing numbers of processors. 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. The efficiency is defined as $E_p = S_p/p$.

We overcommitted the MPI processes per node from 12 to 24. This has a direct impact in the efficiency numbers, as it will be taken into account the 24 logical processors due to the Hyper-Threading technology, instead of the 12 physical processors in each compute node.

In Fig. 5 (a), we show the normalized speedup of *msPar* compared to the sequential version. For the case of maximal recombination we observed a performance degradation. This happened because the node memory was exhausted by the MPI processes. Generating a replica in this scenario means a consumption of ~16 GB, and having 24 MPI processes running with this problem size implies to use more memory (~380 GB) than available physical memory at the node (96 GB), meaning a penalty because the page swapping. In Fig. 5 (b), we show the efficiency we obtained as long as more processes are added for computation. Besides there is a considerable drop-off when the recombination rate is 10240, the efficiency does not significantly change its values in each one of the problem sizes.

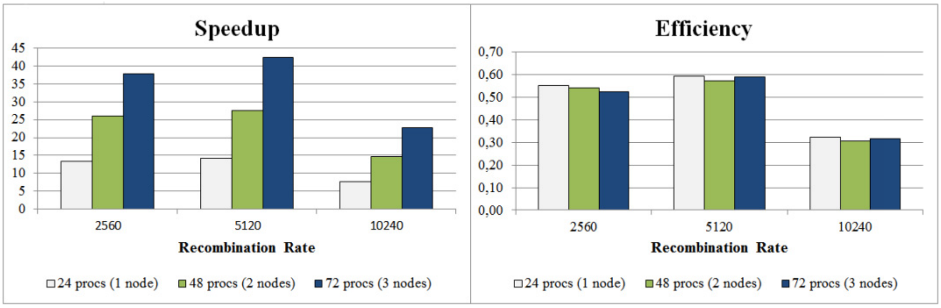

Fig. 5. a) Speedup obtained with 24 processes (1 node), 48 processes (2 nodes) and 72 processes (3 nodes). b) Efficiency showing how well the parallelization goes as long as more processes are added for computation.

The most important contribution for the geneticist is the reduction of the overall running time. Table 1 shows the overall running time *ms* and *mspar* take to complete different problem sizes, demonstrating a significant improvement in the time a researcher should wait to run the experiments.

Table 1. Execution times (in 'd' days, 'h' hours and 'm' minutes) for both ms and msPar

		msPar		
Recombination	ms	72	48	24
2560	2h45m	4m	6m	12m
5120	24h40m	35m	53m	1h44m
10240	7d18h5m	8h11m	12h36m	24h2m

6 Conclusions

Initial results indicate that using the proposed parallel approach we can achieve significant speedup values and much better execution times on an HPC cluster. This can allow population genetic scientists to use *ms* with large genome regions or analysis requiring high number of iterations (e.g. Approximate Bayesian Computation, ABC [28]) otherwise intractable due to time restrictions.

The next step is to refactor the *ms* code in order to remove the data structure dependency and then to apply an hybrid approach (MPI - SMP/GPGPU) depending on the input parameters (genomic region, population size and recombination rate)..

Acknowledgments. This work has been supported by projects number CGL2009-09346 and TIN2011- 28689-C0201-01 of Spanish Ministerio de Ciencia y Tecnologia (MICINN).

References

1. Hudson, R.: Gene genealogies and the coalescent process. Oxford Surveys in Evolutionary Biology 7, 1–44 (1990)
2. Carvajal-Rodríguez, A.: Simulation of Genomes: A Review. Current Genomics 9, 155–159 (2008)
3. Schaffner, S., Foo, C., Gabriel, S., Reich, D., Daly, M., Altshuler, D.: Calibrating a coalescent simulation of human genome sequence variation. Genome Res. 15, 1576–1583 (2005)
4. Carvajal-Rodríguez, A.: Simulation of Genes and Genomes Forward in Time. Current Genomics 11, 58–61 (2010)
5. Kim, Y., Thomas, W.: Simulation of DNA sequence evolution under models of recent directional selection. Brief. Bioinform. 10(1), 84–96 (2009)
6. Liang, L., Zöllner, S., Abecasis, G.: GENOME: a rapid coalescent-based whole genome simulator. Bioinformatics 23(12), 1565–1567 (2007)

7. Peng, B., Chen, H.-S., Mechanic, L., Racine, B., Clarke, J., Clarke, L., Gillanders, E., Feuer, E.: Genetic Simulation Resources: a website for the registration and discovery of genetic data simulators. Bioinformatics, 1–2 (2013)
8. Peng, B., Kimmel, M.: simuPOP: a forward-time population genetics simulation environment. Bioinformatics 21(18), 3686–3687 (2005)
9. Guillaume, F., Rougemont, J.: Nemo: an evolutionary and population genetics programming framework. Bioinformatics 22, 2556–2557 (2006)
10. Sanford, J., Baumgardner, J., Brewer, W., Gibson, P., Remine, W.: Mendel's Accountant: A biologically realistic forward-time population genetics program. SCPE 8(2), 147–165 (2007)
11. Hudson, R.: Generating samples under a Wright-Fisher neutral model of genetic variation. Bioinformatics 18(2), 337–338 (2002)
12. Hoban, S., Bertorelle, G., Gaggiotti, O.: Computer simulations: tools for population and evolutionary genetics. Nature Reviews Genetics 13, 110–122 (2012)
13. Ewing, G., Hermisson, J.: MSMS: a coalescent simulation program including recombination, demographic structure, and selection at a single locus. Bioinformatics 26(16), 2064–2065 (2010)
14. Sanford, J., Nelson, C.: Studies in Population Genetics, pp. 117–135 (August 2012)
15. Peng, B., Kimmel, M.: Simulations Provide Support for the Common Disease–Common Variant Hypothesis. Genetics 175(2), 763–776 (2007)
16. Chen, G., Marjoram, P., Wall, J.: Fast and flexible simulation of DNA sequence data. Genome Res. 19, 136–142 (2009)
17. Excoffier, L., Foll, M.: fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. Bioinformatics 27(9), 1332–1334 (2011)
18. Grünwald, N., Goss, E.: Evolution and population genetics of exotic and re-emergin pathogns: novel tools and appoaches. Annual Review of Phytopathol. 49, 249–267 (2011)
19. Ramos-Onsins, S., Mitchell-Olds, T.: Mlcoalsims: multilocus coalescent simulations. Evol. Bioinform. Online 3, 41–44 (2007)
20. Yuan, X., Miller, D., Zhang, J., Hirrington, D., Wang, Y.: An Overview of Population Genetic Data Simulation. J. Comput. Biol. 19(1), 42–54 (2012)
21. Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A.: Source book of parallel computing. Morgan Kaufmann (2003)
22. Breshears, C.: The art of concurrency: a thread monkey's guide to writing parallel applications. O'Reilly Media (2009)
23. Mattson, T., Sanders, B., Massingil, B.: Patterns for parallel programming. Addison-Wesley Professional (2004)
24. Shao, G.: Adaptive scheduling of master/worker applications on distributed computational resources. PhD thesis, University of California at San Diego (2001)
25. Basney, J., Raman, R., Livny, M.: High throughput Monte Carlo. In: Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing (1999)
26. Pacheco, P.: Parallel programming with MPI. Morgan Kaufmann (1996)
27. Open MPI: open source high performance MPI
28. Beaumont, M., Zhang, W., Balding, D.: Approximate bayesian computation in population genetics. Genetics 162, 2025–2035 (2002)