Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2013, Article ID 345297, 9 pages
http://dx.doi.org/10.1155/2013/345297

Research Article

Using Multicore Technologies to Speed Up Complex Simulations of Population Evolution

Mauricio Guevara-Souza and Edgar E. Vallejo

ITESM-CEM, Carretera a Lago de Guadalupe km 3.5, Col. Margarita Maza de Juarez, 52956 Atizapan de Zaragoza, MEX, Mexico

Correspondence should be addressed to Mauricio Guevara-Souza; a00456476@itesm.mx

Received 11 December 2012; Accepted 18 February 2013

Academic Editor: Cheng-Jian Lin

Copyright © 2013 M. Guevara-Souza and E. E. Vallejo. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We explore with the use of multicore processing technologies for conducting simulations on population replacement of disease vectors. In our model, a native population of simulated vectors is inoculated with a small exogenous population of vectors that have been infected with the *Wolbachia* bacteria, which confers immunity to the disease. We conducted a series of computational simulations to study the conditions required by the invading population to take over the native population. Given the computational burden of this study, we decided to take advantage of modern multicore processor technologies for reducing the time required for the simulations. Overall, the results seem promising both in terms of the application and the use of multicore technologies.

1. Introduction

We are part of a research program whose main purpose is to develop computational tools and models that are useful to gain insights on population dynamics of disease vectors and its potential application to develop new strategies to control vector borne diseases such as malaria and dengue [1].

The introduction of transgenic vectors that are refractory to the disease into wild native populations to achieve the replacement of disease carrying populations is a promising strategy for disease control but so far has only been explored to a limited extent [2, 3].

One possible alternative that seems promising is the introduction of mosquitoes infected with the *Wolbachia* bacteria into wild populations for dengue or malaria disease control. The bacteria produces a variety of fitness and reproduction altering mechanisms that contribute to the establishment of immune populations [4].

Wolbachia pipientis is a bacteria that infects a wide variety of invertebrate taxa. It is estimated that approximately 20% of the insects are all infected with this bacteria [5].

The bacteria can spread rapidly over an uninfected population due to the cytoplasmic incompatibility that it induces in its hosts [6, 7].

This mechanisms cause the progeny of a female that is not infected with *Wolbachia* and a male that is infected to die. If the female is infected, the offspring will survive and will be infected with *Wolbachia* no matter the infection status of the male (see Figure 1).

Moreover, it has been shown that *Wolbachia* provides some virus resistance to their hosts and thus contribute to overcoming loss of fitness. For all this, this mechanism results in the rapid invasion of the host population.

Theoretical models on the dynamics of *Wolbachia* infection have been developed in the past [8]. In principle, these models should be able to explain the dynamics of the infection in native populations. However, theoretical population dynamics models often require strong assumptions, such as unbounded population sizes, probability calculations that are difficult in practice, among others. In this context, computational simulations have proven to be useful for confirming the validity of the theory in practical scenarios. We believe that successful experimental work should rely crucially on sound theoretical work validated by thorough computer simulations.

Based on this premise, we have been developing and testing computer simulation models for several years for

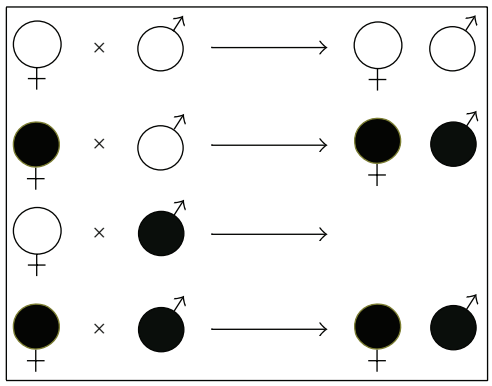


Figure 1: Cytoplasmatic incompatibility induced by *Wolbachia* infection.

a variety of gene drive mechanisms such as transposable elements and the maternal effect dominant embryonic arrest (MEDEA) in order to cast a prediction about the effectiveness of such population replacement strategies for disease control [9, 10].

We are confident that these mechanisms hold much promise in the effective replacement of wild populations in a way that does not affect the food chain or other important variables of the ecosystem. The environment is in such a subtle balance that any external intervention can break the equilibrium with severe consequences.

In this paper, we present a series of experiments of wild population replacement using vectors infected with the *Wolbachia* bacteria. A variety of scenarios are explored using different parameters for each one. The main objective of the simulations has been to determine the conditions required by *Wolbachia* infection to take over the entire native population.

Given the computational intensive simulations required for this study, we used several computer technologies both sequential and parallel to contrast all of them in terms of efficiency in the usage of computer resources.

We believe this kind of benchmark would be valuable for scientists that are not versed in computer technologies to help them choose the most appropriate technology for their studies.

Towards this end, we also explored to what extent we can speed up the simulations by using a variety of multicore technologies, including OpenMP and Java Threads. It is important to point out that we programmed the four simulators as close as possible using the same design and coding standards to be fair in the comparisons between technologies.

In terms of the application, the experimental results presented here provided valuable insights related to the conditions required by population replacement to occur, including the size of the native and the invading populations, among others.

As mentioned above, these results presented here are very preliminary and we only used the simulator to examine whether multicore technologies are applicable to real problems but we want to focus on the benchmark between technologies and not on the results of the computational model.

In terms of the technologies employed for this study, we found that Java threads is a more capable technology for reducing the time required by the simulations and also since it is a computer language that is relatively easy to learn, the time needed to build the software is significantly less.

The organization of the paper is as follows. In Section 2 we describe very briefly some theory about programming languages and programming paradigms for those who are not up to date in the matter. Section 3 describes the computer model we use in our research program. In Section 4, we present the preliminary results obtained with the simulator and the performance benchmark of the different technologies. Section 5 presents our conclusions, both about the experiments performed with the simulator and the performance benchmark.

2. Background

2.1. Computer Technologies

2.1.1. Compiled Programming Languages versus Interpreted. According to their implementation, there are two main categories to classify programming languages: compiled and interpreted. With compiled languages, the code entered by the programmer is reduced to a series of machine language instructions that are stored in an executable file. On the other hand, with interpreted languages, the code is saved in the same language the programmer introduced it or as an alternative it is stored in an intermediate language to be interpreted by a virtual machine.

It is believed that compiled programs run faster than interpreted ones because interpreted programs have to be translated to machine instructions at runtime, but there is a tradeoff: interpreted languages are more flexible. To give an example, interpreted programs can add or modify functions and variables at runtime, compiled programs can not. Another advantage of interpreted languages is that they are usually easier to learn and debug.

Nowadays, the preferred languages in scientific computing are the compiled ones. This preference is due to the belief that compiled languages are more efficient and make a better memory usage. However, with the advance of virtual machine technologies, the efficiency gap has been narrowing.

In this work we made experiments with two different programming languages: one compiled and one interpreted. The languages we used are C++ and Java; both of them are widely used.

2.1.2. Serial versus Parallel Processing. When computers first made their appearance, and until recently, serial processing was the dominant paradigm. The instructions in the computer were processed mainly in a first in first out manner with some hierarchies between the instructions to get sure critical operations were performed before less critical ones.

When CPUs turned fast enough, multitasking operating systems were developed. This multitasking is only an illusion, actually only one task was performed at a time. The processor switched tasks so fast that the user could not notice the

switching between tasks. In serial processing, there is only one path for instruction execution and data flow between processes.

One advantage of serial processing is that the results of a program are very predictable and usually is easier to build a serial program. Computer hardware nowadays is becoming cheaper and more powerful very rapidly. Some years ago, personal computers were very limited in primary memory and processor power, but today, anyone can get a computer that is more powerful than the supercomputers thirty years ago at an affordable price.

It is common that personal computers are equipped with more than one processor and each processor with more than one core. This opened the door to a different computing paradigm called parallel processing.

Parallel processing is the use of more than one CPU or core to execute a program, generally by using threads. Ideally, parallel processing helps the program to execute faster because more hardware is allocated to execute the instructions but this is not always the case.

To take advantage of multiple processing units, the program has to be designed in such a way that several instructions can be executed at once and it is important to state that not every program can be broken into code segments that can be processed in parallel.

The main disadvantage of parallel processing is the complexity of the programs, they are harder to write and debug. Another disadvantage is the power consumption and hardware required, not suitable for mobile devices for the time being.

In this work we wrote the same program in a sequential and parallel programming. For the sequential programs we used Java and C++, for the parallel programs we used Java threads and C++ with OpenMP library, respectively.

2.1.3. Java Virtual Machine. The Java virtual machine (JVM) is responsible of executing the intermediate language of Java, called Java bytecode. The JVM also provides an environment in which the program can be executed providing some utilities like error handling. The main advantage of using the JVM is the portability of the programs written in Java. The same code can be executed in any computer or device that has a JVM enabled.

The JVM provides hardware access protection. Every Java program has assigned a maximum amount of memory it can use. The amount of memory is configured in the JVM parameters. If the program uses more memory than the established, the JVM throws an exception and then kills the process. This protection mechanism is implemented to protect the computer's memory for being corrupted by a malicious program or programming error.

In the mid 90s when Java made its appearance, the main complaints were about the performance of the programs. They were very slow compared with the popular languages at that time like C or C++. In general, scientists still have this idea but at present the JVM has had significant improvements in that aspect, closing the gap in efficiency between compiled languages and Java.

Moreover, since every JVM is made according to the operating system and the hardware, nowadays JVM are designed to fully exploit the computer resources available, feature that a traditional compiled language like C++ does not have.As an example of this, JVM can recognize the pieces of code that are more processor consuming and compile them into machine language to increase efficiency, while the rest of the program is kept in Java bytecodes [11].

2.1.4. Java Threads. Java Threads are incorporated in the core of the language so there is no need for additional libraries to use them. The concept of threads is easy to grasp. In simple terms, a thread is another path in execution through program code so more than one instruction can be executed simultaneously. There are some programs that need to execute several processes at a time, so threads can be very useful in these situations.

All Java programs start with one thread that is the main one. In the body of the program, several new threads can be instantiated and used, each thread with its own call stack. Programming with threads introduces new challenges, especially if they share some memory that needs to be read or written by more than one thread at a time. Several mechanisms are available to prevent shared memory corruption, concurrency problems or deadlock, but thread management is out of the scope of this work.

In summary, threads can be a very valuable tool to speed up and increase efficiency of Java programs but they have to be used with caution, the tradeoff is efficiency against complexity in the design and coding [12].

2.1.5. OpenMP. OpenMP is a third party library that can be used with C, C++, or Fortran programming languages. It is extensively used in scientific and commercial computing. OpenMP application programming interface (API) uses the well-known fork-join model to allow parallel execution in a program that otherwise would be entirely sequential. With OpenMP, multiple threads of execution can perform different task using the directives available in the library. The directives are designed to use all the cores available in the computer.

One advantage of using OpenMP is the capacity of correctly execute the same program both in a sequential or parallel manner. However, there are some programs that would execute correctly as a parallel program but not as a sequential one. Furthermore, when there are mathematical calculations involved, the same program running with different number of threads can yield different results due to the associations of numbers.

As in Java threads, the program starts with a single thread of execution called the initial thread. This thread executes the instructions sequentially until it reaches a region enclosed by a parallel directive. When this happens, the thread creates a team of threads to execute the instructions enclosed inside the parallel directive. The number of threads can be specified in the OpenMP directive. At the end of the parallel block, the team of threads joins to one thread and the execution of the program continues sequentially [13, 14].

3. Materials and Methods

The computer model proposed for this work is a set of mosquito populations connected via migration. Each population evolves independently from each other. We simulated the *Wolbachia* mechanism during reproduction to observe if it is possible for a transgenic mosquito population to replace a native one with the help of *Wolbachia* infection.

To conduct the simulations, two variations of the computer model were used. The difference between them is merely computational. The first one runs in a sequential way while the other one uses multicore technologies to evolve each population on parallel using all the cores available. The simulated biological processes the populations suffer are identical in both variations.

3.1. Mosquito Representation. Each mosquito of the population is represented by a set of attributes that model the most important features of a living being in nature. The first characteristic of the mosquito is the chromosome that is represented by an array of letters. All letters belong to the DNA alphabet (A, G, C, T).

The length of the chromosome was kept small to save computational resources and since *Wolbachia* infection is bacterial and not a DNA modification, the length of the chromosome does not affect the results of the simulations.

The second feature we included in the model is the sex. This distinction between individuals is very important since we want to emulate the reproduction process as closely to the nature as possible. As expected, the sex attribute can assume only two values: male and female.

Another important characteristic is the location of the individuals in the population. For this purpose we used a couple of variables. Location is important specially in the reproduction stage and population structure.

To simulate the *Wolbachia* infection process we used a boolean flag that indicates whether the individual was infected or not.

Finally, we kept track of the fitness of the individual. This value always fell in a range of 0 to 100. It is one of the most important characteristics of an individual. A high fitness value increases the chances of an individual to produce offspring and it also controls the number of offspring that it can produce.

3.2. Population Structure. The individuals are organized in a two-dimensional toroidal square grid that is used to simulate their natural habitat. The position of each individual is important for reproduction because we restricted the females to only mate with males that are located inside a fixed neighborhood (see Figure 2).

The composition of the population is approximately half males and half females but since the sex of the individual is generated randomly at the moment of breeding, this composition can vary from generation to generation.

We decided to use this population structure to get closer to how the reproduction takes place in natural populations where the location of the individuals is not random, it relies heavily in the location of the individuals. In many populations it is known that the best adapted individuals tend to get together in the center of the population and the less favored ones are relegated to remote locations.

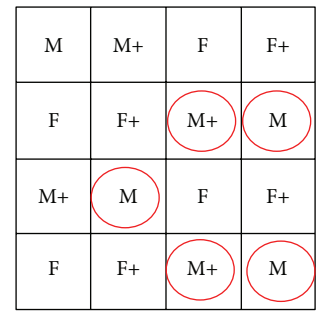


Figure 2: Population structure. Red circles indicate the possible mates of the female in the center of the neighborhood. Individuals with the plus sign are infected with *Wolbachia* bacteria.

3.3. Wolbachia Infection. In this work, the *Wolbachia* infection manifests in the individuals as cytoplasmic incompatibility. The infection process is carried out at the beginning of the simulation. Different percentages of infected mosquitoes are introduced in all the populations and we tracked how many mosquitoes are infected with *Wolbachia* at the end of each generation. Then, we calculate the percentage of infection of every population. In our experiments as in nature, the *Wolbachia* infection does not involve a penalty in the fitness.

3.4. Mating Restriction. In ideal populations, reproductions take place with random mating. In nature, very few or none populations reproduce like that. There are a lot of circumstances that make random mating impossible. In our model, we use geographic restriction in order to make the reproduction more alike as wild populations reproduce.

3.5. Genetic Operators

3.5.1. Selection. The selection of suitable parents is done using tournament selection but with two significant restrictions. The first important restriction is the sex of the individuals. The first parent we selected is the female. After selecting the female, the next step is to find a suitable male.

The second constraint comes in at this point. We used a neighborhood restriction so only the males that are inside the neighborhood can be selected as the female mate. We used tournament selection for choosing both the female and male.

3.5.2. Crossover. In our model, we used a single point crossover to generate the chromosome of the offspring. Before performing the crossover, we checked the *Wolbachia* status of the parents. As described in the *Wolbachia* section, if the female is not infected with *Wolbachia* and the male is, the cytoplasmic incompatibility would kill all the offspring so there is no need to generate the chromosome of the offspring.

If the offspring is feasible, the crossover operation takes place with a probability of 100%.

After the crossover is done, the *Wolbachia* flag is updated in the offspring according to the parents infection status. We believe that always carrying out the crossover operation is closer to how reproduction takes place in nature since the offspring always inherits genetic material from both parents.

3.5.3. Mutation. In our experiments we used uniform mutation in all cases. Since the chromosome of the individual is not binary but contains letters, we had to apply little modifications to the mutation process. For every letter in the chromosome we generated a random number and if it surpassed a threshold the letter was mutated. For every letter that has to be mutated, we used another random generated number. Based on that number we picked another letter of the DNA alphabet to replace it with uniform probability distribution.

3.5.4. Migration. Migration is the movement of some individuals of one population to another one. In nature, this mechanism is important because it introduces some genetic diversity to the populations. In our model, before each new generation is generated, we gather a percentage of individuals of one population and move them to another population to mimic the migration process.

The newcomers participate in the reproduction process, introducing new genetic material and in some cases more probabilities of *Wolbachia* infection.

4. Results and Discussion

4.1. Population Replacement Driven by Wolbachia Infection. Genetic engineering has made great strides in the last few years. It is thought that a transgenic mosquito can be engineered in order to replace the malaria carrying alleles with inoffensive ones. If this is true, founding the more feasible scenarios and mechanisms that can lead to the replacement of a wild population would be very valuable. The goal of this experiment is to found possible scenarios to replace a wild mosquito population with another population infected with *Wolbachia*.

To determine the parameters for the experiments we divided them to two groups. The first group are parameters that were fixed in all the experiments. We decided to keep these parameters unchanged because we think they do not have an important impact in the outcome of the experiments, so varying them would not affect the results in a definitive manner.

These parameters and its values are shown in Table 1. The values stated in this table were used for all experiments.

The second group of parameters were determined using the widely used statistical method known as Latin hypercube sampling (LHS) [15]. LHS is used in computer simulations because it generates a distribution of plausible collection of parameter values from a multidimensional distribution. These parameters were chosen among others because we think they are the most important ones. Additionally, in experiments with real mosquitoes, we can have some control over them.

Table 1: Fixed parameters.

Parameter	Value
Chromosome length	20 bases
Generations	100
Mutation probability	1%
Migration	3%
Tournament size	3
Maximum offspring	10
Neighborhood size	30
Crossover probability	100%

Table 2: Variable parameters.

Parameter	Value
Population size	2,500–250,000
Number of populations	2–4
Percentage of infection	1%–5%

These parameters are shown in Table 2 and are represented as a range, not as a simple value like the other parameters.

We made exhaustive experiments combining the values of the different parameters in order to cover as much scenarios as possible. Due to the large number of combinations of parameters, we obtained a lot of data that we analyzed and synthesized so we are only presenting the results that we considered the most important.

In all cases, several runs with the same parameters were made, and the results presented are an average of all the runs. We are separating the results of our experiments in three parts. We considered each of the parts is important to understand the effect of the different parameters in the infection process.

4.1.1. Population Size Effect. In this scenario we wanted to observe the effect of the population size in the fixation of the *Wolbachia* infection. The results of the experiment are shown in Figure 3.

As can be seen in Figure 3, as populations become larger, the infection expedites its infection. In large populations, the infection spreads through all the individuals in only 80 generations. In contrast, in small populations the infection remains near zero. We think this behavior is produced by the influence of genetic drift.

It is important to denote that a very small percentage of infection were used for this experiment. The results suggest that it is more probable to infect a large population using a small infection percentage than to try to infect a small population even with a larger percentage of infection.

4.1.2. Percentage of Infection. In this result, we show how the percentage of infection affects the speed of the infection process. In this case we are using a medium sized population

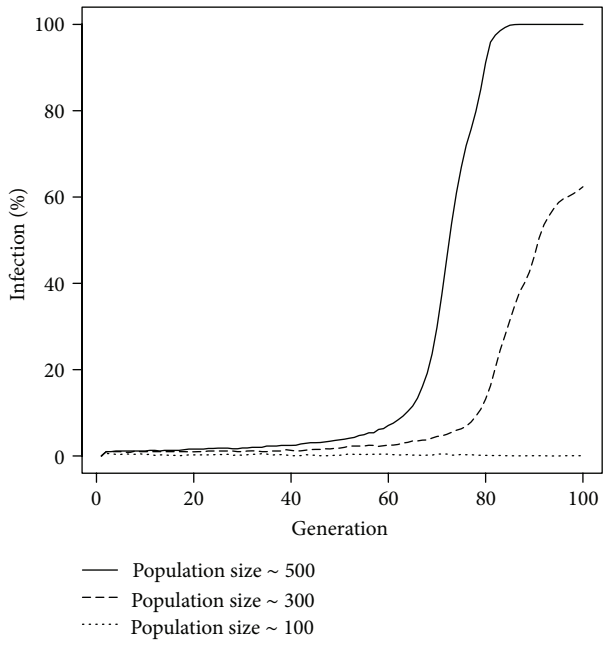


Figure 3: Effect of population size.

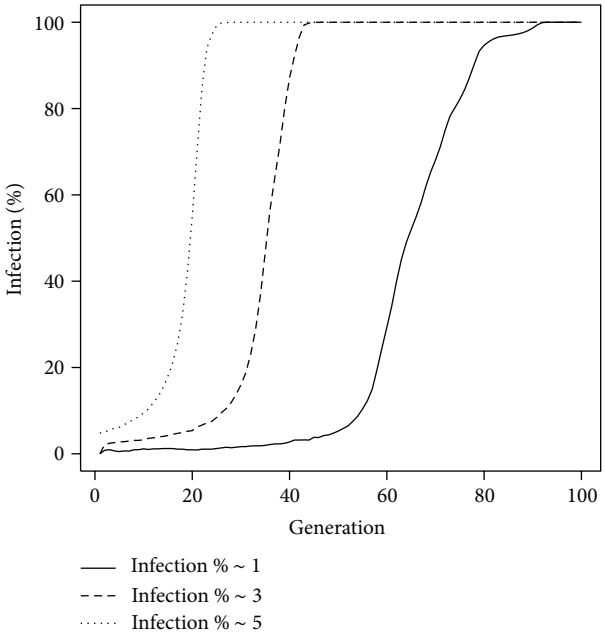


Figure 4: Percentage of infection.

of about 90,000 individuals to observe the effect of different percentages of infection.

The results of this experiment are shown in Figure 4.

There are no surprises in the results obtained in this experiment. In all cases, the infection spread in the entire population, but the spread is faster with a higher percentage of infection. The valuable information obtained with this experiment is that even with a very small percentage of infection, if the population is large enough, the replacement of the native population is feasible.

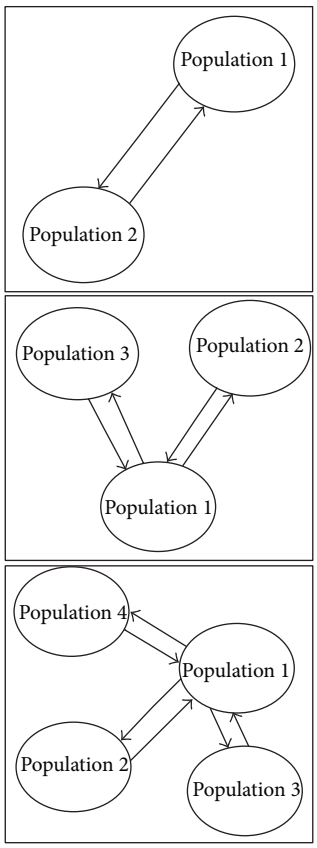


Figure 5: Simulation Environments.

4.1.3. Number of Populations. In this experiment, we wanted to observe if the number of populations can affect the infection process. We are comparing the results obtained with two, three, and four populations connected via migration (see Figure 5). In this case, there is one central population and one, two, or three subpopulations. The migration is between the central population to the subpopulation and vice versa. There is no migration between subpopulations.

The results of this experiment are shown in Figure 6.

The figure shows that the number of populations does not have a direct impact in the infection process. If the number of population increases, there is a small perturbation due to the migration process but is not important enough to consider the number of population a crucial parameter. Conversely, we would like to explore in the future if inoculating just one population would be sufficient for propagating the *Wolbachia* infection to the entire neighbor populations.

4.2. Computer Technologies Benchmark. The goal of this experiment is to benchmark different computer technologies to observe the efficiency of each of them when executing the exact same computer program. For this experiment, we are using a compiled language and a semi-interpreted language, both of them with a sequential program and a parallel one. We are considering Java a semi-interpreted language because in a strict sense it is compiled to bytecodes, but those bytecodes have to be interpreted by the JVM.

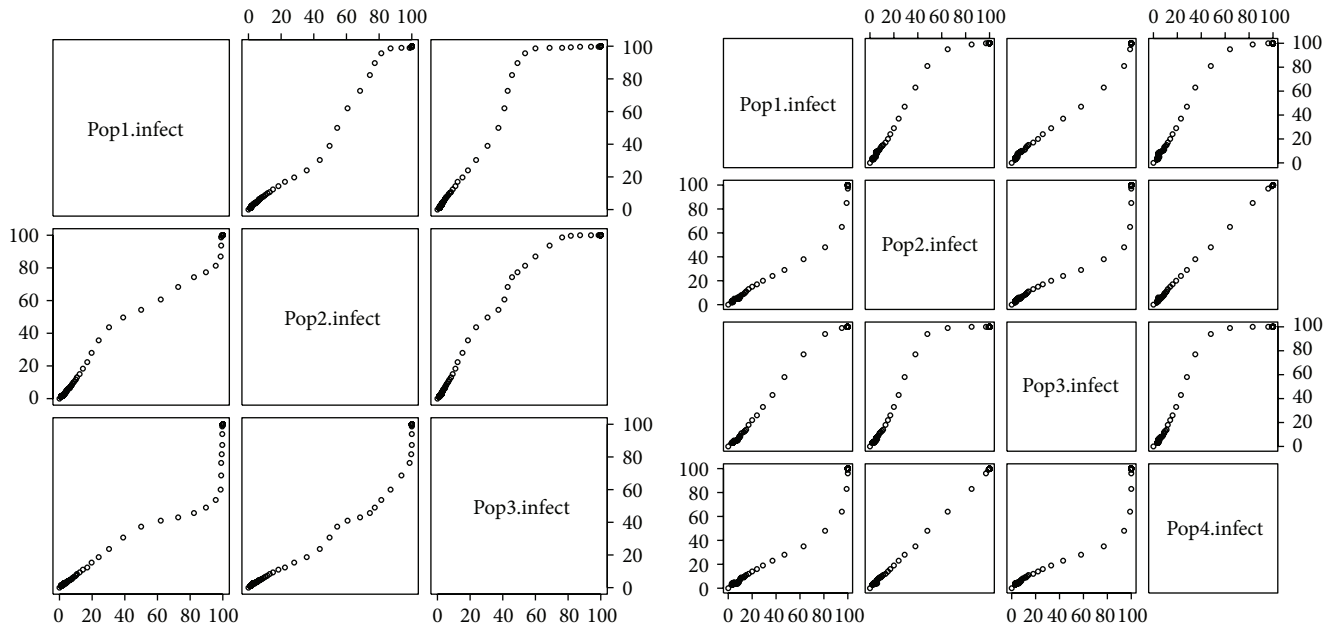


Figure 6: Number of populations.

Table 3: Simulation parameters.

Parameter	Value
Chromosome length	20 bases
Generations	100
Mutation probability	1%
Migration	1%
Tournament size	3
Maximum offspring	5
Infection percentage	1%
Neighborhood size	30
Crossover probability	100%

We used two scenarios. The first one with two populations and the second one with four populations. We used three different population sizes, all of them structured as a two-dimensional square grid. The parameters for the simulation are specified in Table 3 and were the same for all the simulations in this experiment.

We choose these parameters in order to keep the simulations as close as possible as how the populations and the individuals behave in nature. The only parameter that is out of tune is the length of the individual's chromosome. In nature, the chromosome is about thousands of bases but we considered that using such a large chromosome would result in a waste of memory.

The computer used to run the experiments has an Intel Core i5 dual core processor capable of executing four tasks at a time using hyper threading. The machine has four Gigabytes of random access memory (RAM), but the JVM is configured to use at most one Gigabyte of memory for every experiment.

To track the time that each of the experiments took to complete we used two measurements. The first one was the clock of the integrated development environment (IDE). The second one was a clock inside each program that started when the first instruction was executed and ended when all the computation was finished. Both of them yielded the exact same time in milliseconds.

To perform a benchmark like this, it is important to isolate the program as much as possible. When the experiments were running, none of the other programs was executed by the computer. Also, the computer was disconnected from the internet to prevent the unintentional downloading of internet content like operating system updates.

Additionally, we killed all the user and system processes that could bias the result, but we could not kill all the processes due to operating system restrictions. The results of the experiments are shown in Table 4.

For every scenario, 30 runs were performed. The values of the table are the average of all the runs. The values are expressed as a percentage and reflect the speed up of every technology against the slowest one, in this case C++.

As can be seen in Table 4, Java Threads were very superior compared to the other three technologies in all experiments.

Java and OpenMP yielded similar results in overall, in some experiments Java obtained a better speed up but in the rest OpenMP was better.

C++ was far behind all other technologies. Considering that C++ is a compiled language and in general it is considered a very efficient language, the results of these experiments are surprising.

Java Threads speed-up was very constant in all experiments. It does not matter the size of the population or the number of them, the results were almost the same.

On the other hand, OpenMP apparently started to close the gap against Java Threads as the number and size of the populations were increased. Maybe for a simulation with a

Table 4: Simulation results. The slowest technology has a score of 0. The other three technologies have a percentage according to how faster they were compared with the slowest one.

	Two populations			Four populations		
	50 × 50	100 × 100	200 × 200	50 × 50	100 × 100	200 × 200
C++	0	0	0	0	0	0
Java	14.7%	43.1%	36.6%	54.9%	41.7%	34.8%
OpenMP	24.1%	33.1%	38.1%	41.9%	46%	48.6%
Java threads	63.7%	61.6%	60.7%	65.6%	66.1%	61.7%

very large number of populations or with populations with millions of individuals, OpenMP would be the more efficient technology.

In nature, the size of *Anopheles gambiae* mosquito populations rarely is above the 40,000 individuals. Using this premise, Java Threads would be the technology of choice to simulate populations of *Anopheles gambiae* mosquitoes.

Another reason to chose Java or Java Threads among C++ or OpenMP is the complexity of the design and coding of the programs. Programming with Java technologies is far easier than C++ due to the automatic management of memory, the more extensive and useful API and the more robust IDEs available in the open source community.

5. Conclusion

The work presented here showed that computer modeling and simulations simplify the study of population-based phenomena. Particularly, we showed that computational simulations could provide important insights on the conditions required to implement population replacement strategy for controlling vector borne diseases such as malaria and dengue.

The determination of such conditions experimentally would be extremely onerous in practice. Furthermore, we believe that computational simulations are capable of modeling reasonably well evolutionary and genetic aspects of population biology, such as genetic drift and cytoplasmic incompatibility, among others.

The availability of such computer simulation tools is rapidly coming to be crucial as several research groups have already started the actual release of *Wolbachia*-infected mosquitoes. Countries such as China, Australia, Indonesia, Vietnam, Brazil, among others, have joined to the Eliminate Dengue Program, which aims to control dengue by *Wolbachia* induced population replacement. Information about the field release of *Wolbachia*-infected mosquitoes for dengue fever control can be found at (http://www.eliminatedengue.com/).

In addition, we demonstrated that using modern multicore technologies can be extremely useful for reducing the time required for conducting population based simulations, both the experimentation and the programming of the simulator. In effect, simulations on population biology are highly parallelizable in general, so these applications and modern hight throughput computing technologies will be a very good fit.

We also found that computer technologies are advancing so rapidly, that most preconceptions (such as the superiority of compiled versus interpreted programming languages) are no longer valid, at least for a particular set of problems.

References

[1] M. Guevara-Souza and E. E. Vallejo, "Computer simulation on disease vector population replacement driven by the maternal effect dominant embryonic arrest(medea)," in *Software Tools and Algorithms for Biological Systems*, pp. 335–344, Springer, 2011.

[2] J. M. Marshall and C. E. Taylor, "Malaria control with transgenic mosquitoes," *PLoS Medicine*, vol. 6, no. 2, Article ID e1000020, 2009.

[3] A. A. Hoffmann, B. L. Montgomery, J. Popovici et al., "Successful establishment of *Wolbachia* in Aedes populations to suppress dengue transmission," *Nature*, vol. 476, no. 7361, pp. 454–459, 2011.

[4] C. J. McMeniman, R. V. Lane, B. N. Cass et al., "Stable introduction of a life-shortening *Wolbachia* infection into the mosquito Aedes aegypti," *Science*, vol. 323, no. 5910, pp. 141–144, 2009.

[5] P. R. Crain, J. W. Mains, E. Suh, Y. Huang, P. H. Crowley, and S. L. Dobson, "*Wolbachia* infections that reduce immature insect survival: predicted impacts on population replacement," *BMC Evolutionary Biology*, vol. 11, no. 1, article 290, 2011.

[6] S. L. Dobson, C. W. Fox, and F. M. Jiggins, "The effect of *Wolbachia*-induced cytoplasmic incompatibility on host population size in natural and manipulated systems," *Proceedings of the Royal Society B*, vol. 269, no. 1490, pp. 437–445, 2002.

[7] D. C. Presgraves, "A genetic test of the mechanism of *Wolbachia*-induced cytoplasmic incompatibility in *Drosophila*," *Genetics*, vol. 154, no. 2, pp. 771–776, 2000.

[8] V. A. A. Jansen, M. Turelli, and H. C. J. Godfray, "Stochastic spread of *Wolbachia*," *Proceedings of the Royal Society B*, vol. 275, no. 1652, pp. 2769–2776, 2008.

[9] M. Guevara and E. E. Vallejo, "A computer simulation model of gene replacement in vector populations," in *Proceedings of the 8th IEEE International Conference on BioInformatics and BioEngineering (BIBE '08)*, pp. 1–6, IEEE, October 2008.

[10] M. Guevara and E. E. Vallejo, "Computer simulation on the maternal effect dominant embryonic arrest (MEDEA) for disease vector population replacement," in *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference (GECCO '09)*, pp. 1787–1788, July 2009.

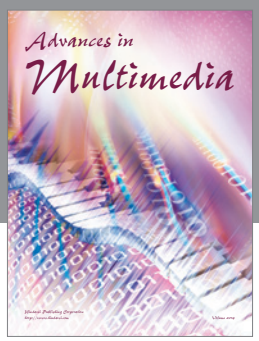
[11] C. Horstman and G. Cornell, *Core Java*, Sun Microsystems Press, San Diego, Calif, USA, 2007.

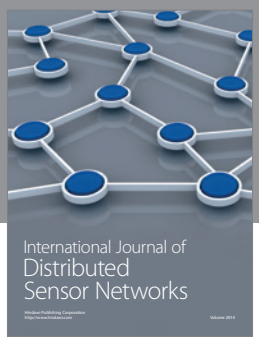
[12] B. Goetz, *Java Concurrency in Practice*, Addison-Wesley, New York, NY, USA, 2006 .

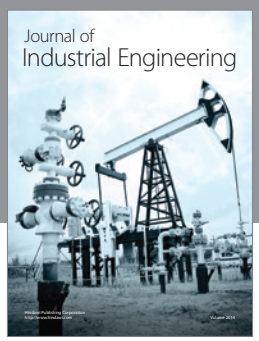
[13] B. Chapman, *Using OpenMP: Portable Shared Memory Parallel Programming*, MIT Press, Boston, Mass, USA, 2008.

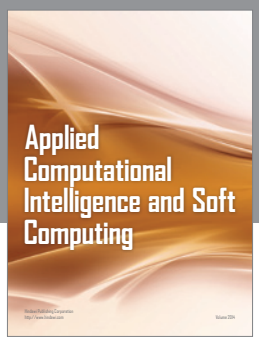
[14] W. Savitch, *Problem Solving with C++*, Addison-Wesley, New York, NY, USA, 2011.

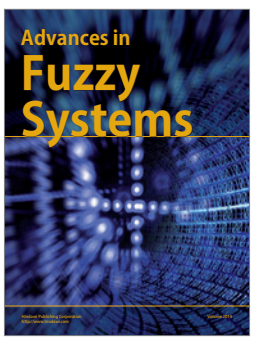
[15] T. Santner, B. Williams, and W. Notz, *The Design and Analysis of Computer Experiments*, Springer Series in Statistics, Springer, 2003.











Hindawi

Submit your manuscripts at
http://www.hindawi.com