Care HPS: A High Performance Simulation Methodology for Complex Agent-Based Models

Francisco José da Silva Borges de Santana

Care HPS: A High Performance Simulation Methodology for Complex Agent-Based Models

Francisco José da Silva Borges de Santana

Advisor: Dr. Remo Suppi Boldrito

Computer Architecture & Operating Systems Department (CAOS)
Universitat Autònoma de Barcelona
Cerdanyola del Vallès, Barcelona

In partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science.

July 2016

Declaration

Care HPS: A High Performance Simulation Methodology for Complex Agent-Based Models

Thesis submitted by Francisco José da Silva Borges de Santana in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the subject of Computer Science by Universitat Autònoma de Barcelona. This work has been developed in the Computer Architecture and Operation Systems department under the supervision of Dr. Remo Suppi Boldrito.

Advisor
Dr. Remo Suppi Boldrito.

July 2016

I would like to dedicate this thesis to the three most important women of my life: Elza Borges, Daniela Bitencourt and Alice Bitencourt Borges.

If one does not know to which port one is sailing, no wind is favorable.

Seneca.

Acknowledgements

PhD is a long journey with several challenges which must be overcome. Throughout this wonderful learning journey , I had spiritual, academic, financial and emotional support from several people and institutions. I dedicate this space to express my deepest and most sincere gratitude to them.

I thank God and my spiritual friends for giving me faith, strength and determination to finish this work and overcome problems that seemed insurmountable.

I thank my parents: José Roque and Elza for investing in my studies all my life. They told me when I was younger that my inheritance would be my studies. Thank you very much, today I am a rich man. Also, I thank all my family for the emotional support.

I thank my beautiful, friend, lifemate and wife Daniela. Unfortunately, I do not have enough space here to express all my gratitude, love and affection to her. Daniela, throughout these years gave me her joy and love. Thank you my love.

I would like to thank the DACSO teachers that contributed to my academic education. Especially, to Prof. Dr. Emílio Luque and Prof. Dra. Dolores Rosário (Lola) for their insightful scientific questions that helped me reflect and question this work. To my PhD candidate friends, especially Viola Saveta, César Azevedo, Pilar Gomez, Francisco Javier Panadero, Albert Gutiérrez Millà, Zhengchun Liu, Alvaro Wrong. And to my Brazilian friends: Aprígio Bezerra and João Gramacho who helped me since I arrived in Barcelona.

I thank Dr. Roberto Solar that helped me understand the Fish Schooling simulator and many pieces of the code.

I thank Dra. Dra Marylene Brito. She taught me and still teaches me so much about epidemiology and the Aedes Aegypti mosquito.

I thank the Brazilian Government, on behalf of President Dilma Rousseff, Federal Institute of Bahia, and my department colleagues for my leave of absence for educational qualification overseas. Investment in education, like this, is fundamental and required for the development of our Brazil.

And finally, to my adviser Dr. Remo Suppi. Thank you for believing in me. Without your emotional, scientific and ACaDemiC support this journey would have been much harder. I am a lucky man to have you as my adviser. Thank you very much for everything.

Thank all of you.

Agradecimentos

O doutorado é uma jornada longa com diversos desafios que devem ser superados. Durante esta maravilhosa jornada de aprendizado tive o suporte espiritual, acadêmico, financeiro, emocional e afetivo de diversas pessoas e instituições para as quais reservo esse espaço para os meus mais profundos e sinceros agradecimentos.

Agradeço a Deus e aos meus amigos do plano espiritual pela fé, força e determinação proporcionada para finalizar esse trabalho e superar problemas que muitas vezes pareciam intransponíveis. Agradeço aos meus pais: José Roque e Elza por toda vida ter investido em meus estudos. Eles falavam, quando eu era mais jovem, que a educação era a herança que eles iriam deixar para mim. Meu profundo agradecimento por me transformar em um homem rico. Estendo minha gratidão a toda minha família pelo apoio e suporte emocional e afetivo.

A minha linda, amiga, companheira e esposa: Daniela. Infelizmente, não tenho aqui espaço suficiente para registrar todo meu agradecimento, amor e carinho por ela. Daniela, que durante esses anos me deu sua alegria, carinho e amor.

Gostaria de agradecer aos professores do DACSO que contribuíram para a minha formação. Em especial, aos professores Dr. Emílio Luque e Dra Dolores Rosário (Lola) por seus úteis questionamentos científicos que me auxiliaram a refletir e questionar o meu trabalho. A todos os meus companheiros de doutorado do departamento, em especial a: Viola Saveta, Cesar Azevedo, Pilar Gomez, Francisco Javier Panadero, Albert Gutiérrez Millà, Zhengchun Liu, Alvaro Wrong. Aos meus amigos brasileiros, Aprígio Bezerra e João Gramacho, que me ajudaram desde a minha chegada em Barcelona.

Ao Dr. Roberto Solar pelo suporte no entendimento do simulador Fish Schooling.

Agradeço muito a Dra Marylene Brito que me ensinou e continua ensinando sobre epidemiologia e o Aedes Aegypti.

Agradeço ao Governo Brasileiro, em nome da Presidente Dilma Rousseff, Instituto Federal da Bahia e aos meus colegas de departamento pelo meu afastamento para capacitação no exterior. Investimentos na educação são fundamentais e necessários para o desenvolvimento do nosso querido Brasil.

E finalmente ao meu orientador Dr. Remo Suppi. Muito obrigado por ACreDitar e Confiar em mim. Sem seu suporte emocional, motivacional, científico e acadêmico essa jornada teria sido muito mais difícil. Tive muita sorte em ter você como orientador. Muito obrigado por tudo.

Obrigado a todos vocês.

Abstract

Parallel and distributed simulation is a powerful tool for developing realistic agent-based modeling and simulation (ABMS). ABMS can allow scientists to reach conclusions and gain knowledge about the system under study. But this is only possible if these simulations offer realistic results, meaning simulations whose results are validated in reality and that can be used for prediction or to explain some phenomenon. These simulations require reliable results through statistical approaches. In addition, they have a high computational complexity because thousands of agents are used in order to model them. For these reasons, this kind of simulation requires a long execution time. Consequently, one possible solution to solve these simulations is to use parallel and distributed simulations that take advantage of the powerful architecture available nowadays. Thus, for the advance of computing science, it is important that High Performance Computing (HPC) techniques, solutions and approaches be proposed and studied. In the literature, we can find some agent-based model tools that use HPC to execute agent-based modeling and simulations. However, none of these tools are designed to execute HPC experiments in order to propose new approaches, techniques and solutions for ABMS that required high performance solutions without great programming effort.

In this thesis, we introduce a methodology to do research on HPC for complex agent-based models that demand high performance solutions. This methodology, named Care High Performance Simulation (HPS), enables researchers to: 1) develop techniques and solutions of high performance parallel and distributed simulations for agent-based models; and, 2) study, design and implement complex agent-based models that require high performance computing solutions. This methodology was designed to easily and quickly develop new ABMs, as well as to extend and implement new solutions for the main issues of parallel and distributed simulations such as: synchronization, communication, load and computing balancing, and partitioning algorithms in order to test and analyze. Also, we developed in Care HPS some agent-based models and HPC approaches and techniques which can be used by researchers in HPC for ABMs that required high performance solutions.

We conducted some experiments with the aim of showing the completeness and functionality of this methodology and evaluate how the results can be useful. These experiments focus on: 1) presenting the results of our proposed HPC techniques and approaches which are used in Care HPS; 2) showing that the features of Care HPS reach the proposed aims; and, 3) presenting the scalability results of Care HPS. As a result, we show that Care HPS can be used as a scientific instrument for the advance of the agent-based parallel and distributed simulations field.

Resumo

Simulação paralela e distribuída é uma poderosa ferramenta para o desenvolvimento de simulação e modelagem baseada em agentes (ABMS). ABMS pode permitir aos cientistas tirar conclusões e obter conhecimentos sobre um determinado sistema em estudo. No entanto, isso somente é possível se essas simulações produzem resultados realistas, validados na vida real e que podem ser usados para predição ou para explicar algum fenômeno. Essas simulações requerem resultados confiáveis obtidos por meio de abordagens estatísticas. Além disso, elas possuem alta complexidade computacional já que milhares de agentes são utilizados para modelá-las. Por essas razões, esse tipo de simulação demandam longo período de execução. Uma possível alternativa para executar essas simulações seriam simulações paralelas e distribuídas que podem tirar vantagens da poderosa arquitetura disponível nos dias atuais. Assim, para o avanço da computação científica, é importante que técnicas, soluções e abordagens de Computação de Alto Desempenho (CAD) sejam propostas e estudadas. Na literatura podemos encontrar algumas ferramentas de modelagem baseada em agentes que utilizam CAD para executar simulação e modelagem baseada em agentes. Entretanto, nenhuma delas são projetadas para executar experimentações de CAD a fim de propor novas abordagens, técnicas e soluções para ABMS que necessitam de computação de alto desempenho sem que seja necessário um grande esforço de programação. Dessa forma, para o avanço da computação científica, é importante que técnicas, soluções e abordagens de CAD sejam propostas e estudadas. Na literatura podemos encontrar algumas ferramentas para modelos baseados em agentes que usam CAD para executar simulação e modelagem baseada em agentes. Entretanto, nenhuma dessas ferramentas são projetadas para executar experimentos em CAD com o objetivo de propor novas técnicas, soluções e abordagens para ABMS que necessitam de soluções de alto desempenho sem que seja necessário um grande esforço de programação.

Nesta tese apresentamos uma metodologia para executar pesquisas em CAD para modelos baseados em agentes complexos que necessitam de soluções com alto desempenho. Essa metodologia, denominada Care High Performance Simulation (HPS), permite a pesquisadores a: 1) desenvolver técnicas e soluções em simulações de alto desempenho paralelas e distribuídas para modelos baseados em agentes; e, 2) estudar, projetar e implementar modelos baseados em agentes complexos que requerem soluções de computação de alto desempenho. Essa metodologia foi projetada para desenvolver novos ABMs fácil e rapidamente, bem como estender e implementar novas soluções para as principais questões de simulação paralela e distribuída como por exemplo: sincronização, comunicação, balanceamento de carga e processamento, e algoritmos de particionamento a fim de testar e analisar. Também, nós desenvolvemos no Care HPS alguns modelos baseados em agentes, e abordagens e técnicas em CAD as quais podem ser usadas por pesquisadores em CAD para ABMs que necessitam de soluções com alto desempenho.

Nós efetuamos alguns experimentos com o objetivo de mostrar a completude e funcionalidade dessa metodologia e avaliar como os resultados podem ser úteis. Estes experimentos focaram em: 1) apresentar os resultados das técnicas e abordagens de CAD propostas que são usadas no Care HPS; 2) mostrar que as características do Care HPS cumprem os objetivos propostos; 3) apresentar os resultados de escalabilidade do Care HPS. Como resultado, nós mostramos que o Care HPs pode ser usado como instrumento científico para o avanço do campo de pesquisa em simulações paralelas e distribuídas baseadas em agentes.

Resumen

La simulación paralela y distribuida es una potente herramienta para el desarrollo realista de modelos basados en agentes y su simulación (ABMS). Esta herramienta permite que científicos de diferentes áreas puedan realizar conclusiones y adquirir conocimientos acerca del sistema bajo estudio. Sin embargo, esto sólo es posible si las simulaciones realizadas ofrecen resultados realistas, es decir si los resultados se asemejan a la realidad y si estas simulaciones pueden ser utilizadas para la predicción o para explicar algún tipo de fenómeno emergente. Por ello, estas simulaciones requieren resultados confiables a través de métodos estadísticos y presentan una alta complejidad computacional debido a que miles de agentes independientes se utilizan para modelar el sistema. Por estas razónes, este tipo de simulación requiere de largos tiempos de ejecución y de gran potencia de cómputo. Una posible solución para resolver este tipo de simulaciones es la utilización de sistemas paralelos y distribuidos que aprovechan la potencia de la arquitectura subyacente disponible en las infraestructuras actuales y es importante, para el avance de la ciencia de la computación, el desarrollo de técnicas, algoritmos y enfoques que permitan analizar estos sistemas ejecutándose sobre infraestructuras de cómputo de altas prestaciones. En la literatura, se pueden encontrar algunas herramientas que permiten el modelado basados en agentes y que utilizan HPC pero ninguna de estas herramientas están diseñadas para ejecutar experimentos con el fin de incluir y analizar nuevos enfoques, técnicas y soluciones para ABMS que requieran un alto rendimiento y poco esfuerzo de programación.

En el presente trabajo, se introduce una metodología para realizar investigaciones en modelos complejos basados en agentes que demandan soluciones de alto rendimiento (HPC). Esta metodología, llamada Care High Performance Simulation (HPS) permite a los investigadores: 1) desarrollar técnicas y soluciones de alto rendimiento y simulaciones distribuidas para modelos basados en agentes; y, 2) permite el estudio, diseño e implementación de modelos complejos basados en agentes que requieren soluciones de computación de alto rendimiento. Esta metodología ha sido diseñada para desarrollar de forma fácil y rápida nuevos ABM, así como para extender y aplicar nuevas soluciones a los diferentes módulos funcionales que afectan a una simulación paralela y distribuida, tales como la sincronización, la comunicación, la carga y el balance de la computación y/o los algoritmos de partición de datos. Dentro del presente trabajo, y como prueba de concepto, se han desarrollado además en Care HPS diferentes modelos basados en agentes y técnicas/algoritmos que pueden ser utilizados por los investigadores en ABMS y que requieran soluciones HPC para realizar sus investigaciones.

Para validar la propuesta se han realizado un conjunto de experimentos con el objetivo de mostrar la completitud y funcionalidad de esta metodología y evaluar la bondad de los resultados obtenidos. Estos experimentos se centran en: 1) validar los resultados de las técnicas propuestas y enfoques que se utilizan en Care HPS; 2) mostrar que las características de diseño de Care HPS satisfacen los objetivos propuestos; y finalmente, 3) verificar los resultados de escalabilidad de Care HPS como infraestructura de simulación distribuida para modelos basados en agentes. En conclusión, Care HPS puede ser utilizado como instrumento científico en el desarrollo de modelos basado en agentes y en el área de simulaciones distribuida en arquitecturas HPC.

Resum

La simulació paral·lela i distribuïda és una potent eina per al desenvolupament realista de models basats en agents i la seva simulació (ABMS). Aquesta eina permet que científics de diferents àrees puguin realitzar conclusions i adquirir coneixements sobre el sistema sota estudi. Tanmateix, això solament és possible si les simulacions realitzades ofereixen resultats realistes; és a dir, si els resultats s'assemblen a la realitat i si aquestes simulacions poden ser utilitzades per a la predicció o per a explicar algun tipus de fenomen emergent. Per això, aquestes simulacions requereixen resultats confiables mitjançant mètodes estadístics i presenten una alta complexitat computacional pel fet que milers d' agents independents s'utilitzen per a modelar el sistema. Per aquesta raó, aquest tipus de simulació requereix de llargs temps d'execució i de gran potència de computació. Una possible solució per a resoldre aquest tipus de simulacions és la utilització de sistemes paral·lels i distribuïts que aprofiten la potència de l'arquitectura subjacent disponible en les infraestructures actuals i és important, per l'avanç de la ciència de la computació, el desenvolupament de tècniques, algoritmes i enfocaments que permetin analitzar aquests sistemes executant-se sobre infraestructures de computació d'altes prestacions. En la literatura, es poden trobar algunes eines que permeten el modelatge basat en agents i que utilitzen HPC però cap d'aquestes eines estan dissenyades per executar experiments amb el fi d'incloure i analitzar nous enfocaments, tècniques i solucions per a ABMS que requereixin un alt rendiment i poc esforç de programació.

En el present treball, s'introdueix una metodologia per a realitzar investigacions en models complexos basats en agents que demanen solucions d'alt rendiment (HPC). Aquesta metodologia, anomenada Care High Performance Simulation (HPS) permet als investigadors: 1) desenvolupar tècniques i solucions d'alt rendiment i simulacions distribuïdes per a models basats en agents; i, 2) permet l'estudi, disseny i implementació de models complexos basats en agents que requereixen solucions de computació d'alt rendiment. Aquesta metodologia ha sigut dissenyada per desenvolupar de forma fàcil i ràpida nous ABM, així com per a estendre i aplicar noves solucions als diferents mòduls funcionals que afecten a una simulació paral·lela i distribuïda, tals com la sincronització, la comunicació, la carrega i el balanç de la computació i/o els algoritmes de partició de dades. Dintre del present treball, i com a prova de concepte, s'han desenvolupat a més en Care HPS diferents models basats en

agents i tècniques/algoritmes que poden ser utilitzats per als investigadors en ABMS i que requereixen solucions HPC per a realitzar les seves investigacions.

Per a validar la proposta s'han realitzat un conjunt d'experiments amb l'objectiu de mostrar la completitud i funcionalitat d'aquesta metodologia i avaluar la bondat dels resultats obtinguts. Aquests experiments es centren en: 1) validar els resultats de les tècniques proposades i enfocaments que s'utilitzen en Care HPS; 2) mostrar que les característiques de disseny de Care HPS satisfacin els objectius proposats; i finalment, 3) verificar els resultats d'escalabilitat de Care HPS com infraestructura de simulació distribuïda per a models basats en agents. En conclusió, Care HPS pot ser utilitzat com a instrument científic en el desenvolupament de models basats en agents i en l'àrea de simulacions distribuïda en arquitectures HPC.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

ABM	Agent-Based Model
ABMS	Agent-Based Modeling and Simulation
ABS	Agent-Based Simulation
ANOVA	ANalysis Of VAriance
API	Application Program Interface
DES	Discrete-Event Simulation
GIS	Geographical Information Systems
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDF	Hierarchical Data Format
HPC	High Performance Computing
HPS	High Performance Simulation
IBM	Individual-Based Modeling
IDE	Integrated Development Environment
IOM	Individual-Oriented Modeling
MIMD	Multiple Instruction Multiple Data
MPI	Message Passing Interface

MVC	Model-View-Controller
NUMA	Non-Uniform Memory Access
OOP	Object-oriented programming
PDS	Parallel and Distributed Simulation
SD	System Dynamics
SPMD	Single Program Multiple Data
UMA	Uniform Memory Access
XML	Extensible Mark-up Language

Chapter 1

Introduction

1.1 Motivation and justification

Over the last 15 years, our group has been researching High Performance Computing (HPC) in parallel and distributed agent-based simulation. From the beginning, our main aim has been to get more realistic simulations, because many researchers in application areas, such as ecology, biology and physics, have expressed the need for more realistic results. The problem is that realistic simulations require a lot of computing resources. Thus, our research line is directed to study how to provide these simulations in a more efficient and scalable way.

From the point of view of scalability and efficiency in the Agent-Based Model (ABM) context, we have noted that the HPC approaches used to reach a good scalability and efficiency depend on the characteristics of the model. We understand that the characteristics of the ABM and HPC techniques must fit together in order to extract the best solution performance. As an example, whether the simulation type is communication or computing demand. ABM simulations that have a high communication volume require a different distribution of data throughout the architecture compared with ABM simulations that are computing demand. This reasoning is applicable for other important HPC and ABM issues, such as synchronization, load balancing, partitioning, among others.

The initial idea of Care HPS came up as a methodology to support the aims of this research line of our research group. For this reason, Care HPS basically comes from the answers to two questions: 1) How can we make a generalization of our HPC techniques for ABM simulations that demand high performance solutions? and 2) How can we develop other ABM models rapidly and test our HPC techniques with other ABM roles? These questions are important because in computer science, as well as in general science, the generalization of results is a fundamental scientific aspect. The research must give results that can be applied in other contexts in order to have a bigger impact on and importance to the studied field. From previous simulation analysis and from initial works, we can conclude that our previous contributions had an impact in a very-specific context because our solutions were developed always using the same ABM case study (Fish Schooling - see 4.2.1). Therefore, these proposed solutions can be generalized for ABMs which have similar behaviors and characteristics such as the flocking of birds. Applying these solutions to other contexts would require a huge effort in programming, since other models with different computing characteristics must be implemented. These are problems that our methodology and proposal may solve. In this direction, we hope that the next step in our research will provide solutions for a greater range of models. For that, tools that we can easily add new models to using the same HPC feature are needed, as well as a tool in which the extended features could easily be tested using the same model. Figure 1.1 shows two hypothetical scenarios with the objective of presenting the applicability of proposing and developing a methodology. In Scenario 1, suppose we have implemented a load balancing (LB) algorithm. The methodology proposed enables the user to use different ABMs (ABM_A, ABM_B and ABM_C), which have different computing requirements in order to analyze the hypothetical load balancing algorithm proposed. These scenarios enable the HPC expert to evaluate the efficiency of the load balancing hypothetical implemented, or another HPC issue, with different ABM solutions. Therefore, Scenario 1 reflects an HPC expert that develops an HPC approach and wants to test it with other ABM contexts. In the other scenario, suppose we have an ABM, and we want to analyze which load balancing algorithm (LB_A, LB_B and LB_C) will fit best with ABM requirements. This scenario enables the application area researcher to identify which LB algorithm offers the best execution times without a great effort in implementation, by just connecting the HPC approach and ABM on Care HPS. Therefore, Scenario 2 reflects an application area researcher that studies an ABM and wants to test which HPC feature fits best with the ABM under study.

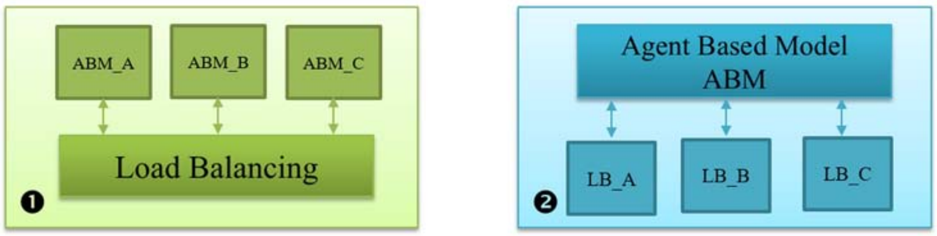


Figure 1.1: Different scenarios of research.

One of the biggest weaknesses for developing scalable and efficient solutions, in previous works of our research group, is the high effort in the programming and full knowledge of the software architecture. Until the end of 2013, a lot of programming effort was required in order to apply new HPC approaches to ABM, such as a new partitioning algorithm. This meant too many changes and too much restructuring of the simulator because its architecture was overly coupled. Figure 1.2 presents how it was: the ABM model (in this case, fish schooling), the kernel of simulation and the HPC solutions were all connected. Therefore, we were spending more time programming than doing science. Now, we have developed a methodology that enables us to focus more on the generation of knowledge in HPC to ABM.

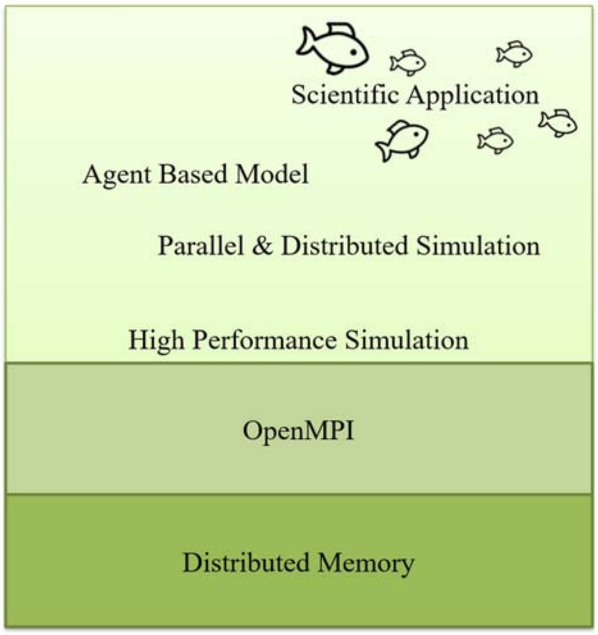


Figure 1.2: First architecture of our fish schooling simulator.

In the literature, we find several tools in which ABM had been very well tested in specific projects and presented a high maturation level. As examples: FLAME in EURACE project [45, 81] which simulates the European economy; Netlogo [163] has a huge community and a large library of sample models; Pandora is being used in SimulPast, which is an archaeological project [131, 132, 92]; and Repast HPC has presented excellent weak scalability on Argonne National Laboratory's IBM Blue Gene [110]. But most of these tools address and support HPC features using generic approaches to execute an ABM model. In addition, they do not allow the user to extend the HPC features without a huge programming time investment. Thus, Care HPS was born as an instrument that supports HPC experimentations for ABM simulation that demand by high performance solutions. The basic premises are:

1. To support the addition of new HPC features;
2. To allow the extension and reuse of HPC techniques proposed for ABM simulation;

3. To permit the development of new ABM models faster; and,
4. To enable the user with less programming know-how to develop Parallel and Distributed Simulation (PDS) without previous knowledge of HPC.

1.2 Research questions

This research will address the following question:

- how to generalize our HPC techniques and approaches for agent-based models that demand high performance solutions?

To make a generalization in science on the field under study, the solution must be analyzed in other contexts. This is not different for HPC techniques and approaches for agent-based models that demand high performance solutions. Thus, it is possible for a solution to have a scope of validation with a wider range. One of the biggest problems in experimental research is the time invested in the implementation of solutions in order to test or prove a theory or hypothesis through scientific experiments. Therefore, in this thesis we present a methodology that contributes in this sense.

1.3 Problem

We can find in the literature several ABM tools that support several issues, features, and contexts. A few of them use HPC to execute agent-based models. Although exist many consolidated ABMS tools none of them enable the experimentation in HPC. None of these tools are designed with the aim of being a scientific instrument that facilitates the research of HPC for agent-based models that demand high performance solutions. Of course, many of these tools are open source and, consequently, other features can be added, but the time invested from programming is too high and requires high level knowledge of the tool and programming. Therefore, this research will fill this gap in the literature.

1.4 Objective

In this thesis, we introduce a methodology called Care High Performance Simulation (HPS). The main objective of Care HPS is:

- to be a scientific instrument for research on HPC for agent-based models that demand high performance solutions.

Its sub-objectives are:

- to enable application area researchers to study, design and implement complex ABM models that require an HPC solution; and,
- to enable HPC experts to develop techniques and solutions of high performance distributed simulation for agent-based models.

Care HPS was idealized with the aim of being a methodology, in a wider meaning, and a tool that enables us to propose new algorithms, models, and methods that enable us to cover the whole cycle of design and development of HPS for ABM. Also, it is important to say that Care HPS was not conceived to be just another ABMS tool that executes ABMS simulations. Of course, Care HPS implements this very important feature, but its purpose is to enable HPC experimentations for ABMS. So, this proposed methodology enables us to answer research questions for HPC approaches in an ABMS context, such as:

- What is the best communication pattern for a specific ABM whose agents move constantly in the environment?
- What is the best communication pattern: synchronous or asynchronous for a determined ABM that is communication bound?
- What if the load balance strategy considers the number of neighbors of an agent?
- Can a partitioning approach that uses a specific algorithm offer better results for a determined model than other partitioning approaches?
- Is a load balance strategy that take into account the number of agents in a partitioning for a specific ABM whose agents move constantly efficient?
- What is the behavior of a determined ABM like if the synchronization occurs for each 1000 steps instead of step by step?
- How scalable is the partitioning algorithm if each partition is executed per 4 and 8 threads?
- and so on and so forth.

And many other research questions can be carried out by Care HPS.

1.5 Contextualization of HPS

Performance, scalability, and efficiency are recurring problems in parallel and distributed simulation that must be analyzed and studied so that the ABMs can be executed faster using HPC. Care HPS emerges as a methodology that allows for the development of studies of different approaches for several problems in these areas. Care HPS allows us to develop and simulate ABMS as well as other well-know tools. But its main novelty is that it allows for the experimentation of ABM with HPC. This means that new approaches can be verified without changes in its kernel or huge programming efforts. Other ABMS tools, such as D-Mason [42], FLAME [69], Pandora [130], and Repast HPC [40] were developed with the purpose of allowing for the development of ABMS that require HPC. The models developed in these tools use the built-in HPC techniques available in them. These tools were not designed to enable the user to propose new HPC techniques. It is important to emphasize that these tools were designed to enable the user to develop their ABM and to use the HPC techniques available in the tools. Of course, the user can propose new HPC techniques in these tools because they are open source though this is not the aim of these tools. For that, a very good understanding of the tool code is required, and changes are necessary in its kernel. On the other hand, Care HPS allows the user to add and extend features without a huge programming effort through software engineering.

Figures 1.3 and 1.4 try to synthesize the objective of Care HPS. For didactic effects, Care HPS is represented simply in these figures, with the kernel of this tool painted in yellow. Also, we represent only partitioning, communication, and synchronization as examples; therefore the other HPC features are not represented in these figures. The cracks represent interfaces between Care HPS and other pieces of the user code. Figure 1.3 presents a very simplified view of the design of Care HPS and a conceptual design of other ABM tools. In Care HPS, the user can attach, through the use of interfaces, their ABM and HPC solutions to the kernel of Care HPS. In the other ABM tools, the user can only attach their ABM solution and has to use the HPC solutions available in the kernel of the tool.

Giving a zoom in Care HPS, Figure 1.4 shows that the interfaces allow the user to fit these pieces of code with the kernel of Care HPS. To do so, these pieces must comply strictly with the specifications defined by the interface. The interface depends on the purpose of the piece. These pieces can be agent-based models 1.4(a) and/or HPC approach 1.4(b). In Care HPS, the user can propose and implement new ABMs, change the roles and behaviors of ABM or use previously developed ABMs 1.4(a). Also, Care HPS allows the user to propose and implement new techniques, modify built-in techniques and use HPC techniques such as partitioning, communication and synchronization 1.4(b,c). The user might implement their ABM, partitioning, communication, synchronization and other HPC solutions, then

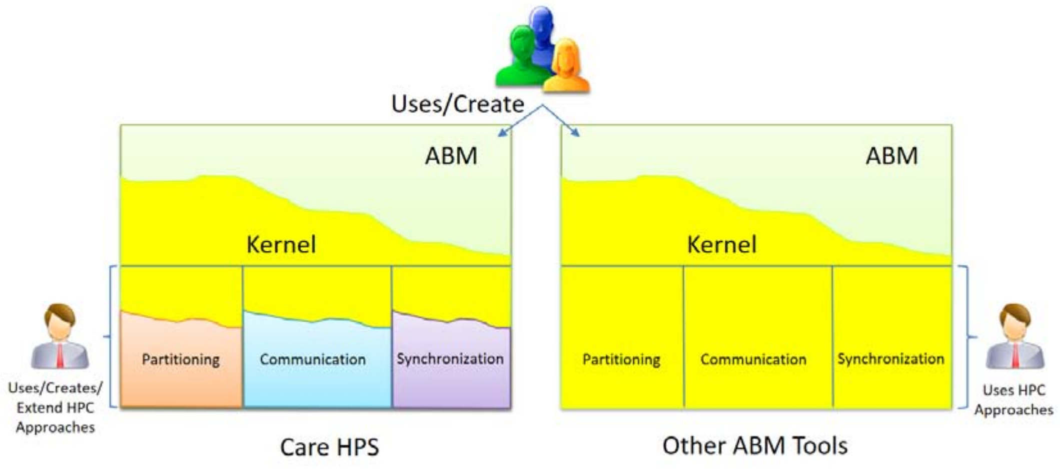


Figure 1.3: Conceptual representation of Care HPS design and other ABM tools design.

attach their solutions to the kernel of Care HPS. This design, basically supported by software engineering techniques, enables the user to experiment with several approaches 1.4(d) without a lot of codification. As examples of use, the user can test which partitioning techniques offer the best performance for a specific ABM; or, the user can analyze and study a new proposed algorithm of synchronization using ABMs which require different computational characteristics.

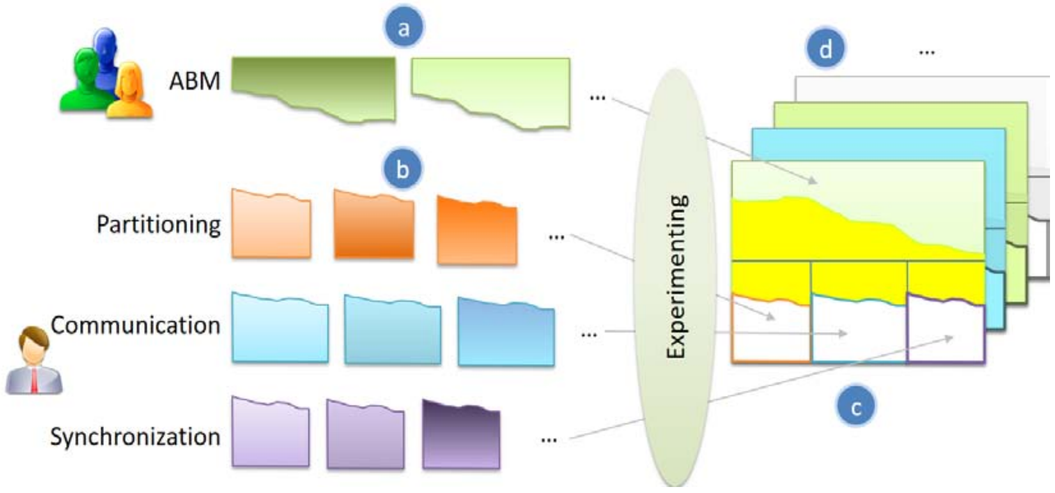


Figure 1.4: Objective of Care HPS .

1.6 Methodology

In this section, we describe the methodological steps followed in order to develop this thesis.

- Study of previous works, related works and state of the art. At the beginning, the initial background about ABMS, HPC and HPS were obtained from the previous related works of the research group. These works helped us to gain a basic understanding of the problem and give autonomy to find more related works and improve the literature review. In order to map and verify the academic contributions on ABMS tools, we checked the following surveys [4, 109, 151]. We have listed nominally all of the tools and checked which tools have some features and purposes of support simulations. We have found around 80 different tools referenced in the literature. After that, we then selected the tools that fit with the criteria: support parallel and/or distributed simulation and/or support for agent-based modeling and simulation. We could find several ABM tools that support several issues, features, and contexts. A few of them use HPC to execute their models. Therefore, we concentrated our study and analyzed the following: D-Mason [42], FLAME [69], Pandora [130], and Repast HPC [40]. This literature review was very important because of two reasons: 1) we could not find a methodology that supported HPC experiments for ABMs; and 2) we checked the weaknesses and strengths of the main ABMs tools that support HPC executions.

- After that, we identified our research problem as presented in Section 1.3 using the motivation and justification that were discussed previously (see Section 1.1).

- From this point on, we used this information in order to propose and define our methodology.

- We developed the models, and algorithms, and we implemented the proposed methodology, which becomes a tool.

- And finally, we executed the tests, validation and experiments. The main findings of this research were obtained through experimental work. These findings can be categorized in: ABMs implemented; HPC approaches proposed and implemented; the methodology proposed; and the implementation of the methodology, thus providing Care HPS as a scientific instrument:

  - For ABMs, we execute a simple computational alignment [8] between the original model and our implemented model.

  - For HPC findings, we tested them with ABMs. The performance measures of the HPC findings come from statistical approaches.
  - We conducted functional experimentation of Care HPS in order to prove Care HPS is able to be used as a scientific instrument. For that, we conducted some experiments where we showed that Care HPS can model and simulate an agent-based model; and we show that new HPC approaches for ABM can be developed and compared using Care HPS.

In these methodological steps, the development of ABMs, HPC approaches, and the design and implementation of the methodology occurred in cyclic and progressive processes.

1.7 Threats to validity of research

This work produced promising outcomes, specifically in the area of HPC experimentation for ABM that demands parallel and distributed simulation. However, there are also limitations that need to be addressed. These limitations to validity are discussed below.

- Comparison with other ABMS tools - Care HPS was idealized with the aim of being a scientific instrument that facilitates the research of HPC for agent-based models that demand high performance solutions. Even though Care HPS has a different scope, the comparison with other ABMS tools is unavoidable. More specifically: whether the output of the simulation is similar; and which tool produces better performance measures for a specific model and technique.

- Quantity of built-in solutions available - Care HPS currently has some implemented ABM and HPC solutions, but it would be interesting to have more solutions in order to offer application area researchers more options to execute more tests in their ABM.

- Manual output analysis - as we said before, new HPC approaches can be developed and tested with ABMs. The comparison processes of the simulation output are manual. This means that the user has to execute their statistical method in order to check whether the results are reliable, and this process can create bias. None of the platforms listed in this work provide a complete tool for statistical output [123]. Therefore, it would be an important contribution.

All these limitations demand a high number of hours of reading, research and programming. Therefore, these limitations were not treated because of time limitations and they will be addressed in future work.

1.8 Contributions

As result of our research throughout these last four years, we can cite as contributions:

- As main contribution, we propose Care High Performance Simulation (HPS). Care HPS is a methodology for agent-based simulations in a parallel and distributed architecture. Care HPS is a scientific instrument that enables both:
  - application area researchers to gain knowledge about the system under study using ABMs that require high performance computing solution. This is possible because Care HPS offers a well defined and simple interface for this type of user in which all HPC complexity is hidden.
  - HPC expert users to develop techniques of high performance parallel and distributed simulation for ABM problems without high programming effort. Care HPS was projected using good object-oriented design practices which enable the extension and reuse of the main HPS features.

Specifically in the HPC field, we can cite the following contributions that were implemented in order to support this methodology:

- We analyzed and compared three communication strategies: synchronous and asynchronous message passing (via MPI) and bulk-synchronous parallel (BSP) for our distributed cluster-based fish school model. We showed that distributed simulations do not always improve the performance when using synchronous communication strategies, and, that asynchronous communications strategies are more efficient. Also, we have verified that the bulk-synchronous parallel method is a scalable [143].
- We study the use of affinity of processes [59] in the simulation using Hwloc [25].
- We developed a hybrid MPI+OpenMP version of cluster-based partitioning. In the hybrid approach developed, we fit our simulation features in the following manner: the communication between the logical processes happens via message passing whereas the computing of the agents by OpenMP threads. In addition, we propose a new data structure for partitioning the fish clusters which avoid the critical section in OpenMP code. As a result, the hybrid version significantly improves the total execution time for huge quantity of individuals, because it decreases both the communication and management of processes overhead, whereas it increases the utilization of cores with sharing of resources [16].

- We proposed a strip partitioning strategy to a spatially dependent problem in agent-based model applications. This strategy avoids sharing resources, and, as a result, it decreases communication volume among the processes. In addition, we develop an objective function that calculates the best partitioning for a specific configuration and gives the computing cost of each partition, allowing for a computing balance through a mapping policy. The results obtained are supported by statistical analysis and experimentation with an Ant Colony application. The partitioning strategy can be chosen dynamically and always returns the lowest total execution time [18].
- We presented a hybrid strip partitioning, based on [18], which was implemented with OpenMPI and OpenMP. This partitioning checks the proportion of the quantity of agents inside a strip and dynamically creates a number of threads.

In the parallel and distributed simulation field, we can cite the following contributions that were implemented:

- We have applied and adapted the simple statistical approach in order to define the optimal simulation length; we propose the approximate approach to normal distribution instead of generate replications sufficiently large; and the method can be used in other kind of non-terminating science simulations where the data either have a normal distribution or can be approximated by a normal distribution [17].
- We have adapted the ant colony model described in [162] for a parallel and distributed approach. In our approach, the environment (stack of food and nest) is distributed by architecture, but the resources consumed by the ants are not shared among the processors. This avoids an increase in communication and barriers [18].
- We have adapted the Shopping Agent model described by [61] for a parallel and distributed approach.
- We develop a serialization schema for agents which enables the migration of the agents among distributed processes. This schema saves additional fields of the agent.

In the agent-based model, we can cite the following contributions that were implemented:

- We propose an Agent-Based Model for assessment of the pupal productivity of the *Aedes Aegypti* mosquito. In this model, the reproduction of the mosquito takes into account the productivity of each type of container. The preliminary results show the effects of considering the pupal productivity for the control and prevention of *Aedes*

Aegypti. As a result, we observed that the prevention methods must consider pupal productivity and that the distance between containers might leverage productivity and increase transmission risk. We verify the completeness and functionality of the model through experimentation using Netlogo [19]. This verification was conducted with parallel execution on the cluster of the model in order to obtain reliable results through statistical approaches. Also, it is important to note that this model was developed in order to verify and validate the proposed methodology in a real problem and to verify if the methodology enables us to implement all elements of a new model.

1.9 Deliverables

Besides this thesis and the published papers listed at Section 7.4, we produced the following items in these research years.

- Care HPS implementation whose code can be used by the scientific community with the Creative Commons license (CC BY-NC-SA). The source code of Care HPS has around 130 files and approximately 25,000 lines.
- The documentation of Care HPS was developed using Doxygen [157]. This documentation has the API specification and examples of how to use it.

1.10 Organization of Thesis

This thesis is organized basically in five main parts that are composed of one or more chapters:

- The introductory part is composed of Chapter 2 that gives basic definitions for readers that are not familiar with some specific concepts. Simulation is a multidisciplinary topic; therefore, it is important to set up some definitions and concepts.
- The literature review part is composed of Chapter 3. We present related works that support our proposed contributions in this thesis. This chapter focuses on the main subjects of this work: agent-based model, simulation and high performance computing. Also, we investigate other ABM methodologies for HPC. In addition, we discuss techniques about how to design an agent-based solution for a parallel and distributed simulation. This chapter supports all work developed in this thesis.
- The contributions part has the contributions that were proposed in this thesis. In Chapter 4, we present the four agent-based models developed and used as case studies in order

to present and explain the methodology. Also, we present the high performance computer approaches proposed to solve problems due to ABM execution in HPC environments. And the methodology Care HPS is presented in Chapter 5.

- The results part is composed of Chapter 6 which presents experiments of the contributions presented in the contributions part. We divide this chapter into four parts. In the first, we present the validation of Assessment of Aedes Aegypti pupal productivity model and we translate this model to Care HPS. After, we present the results of our proposed HPC techniques which are used in Care HPS. Then, we present functional results of Care HPS features; and lastly, we present the scalability results of Care HPS.
- Chapter 7 presents the conclusion, future works, open lines and our publications.

Chapter 2

Theoretical basis

2.1 Introduction

This section presents definitions and explanations about topics that will be frequently used and cited throughout this research. Oftentimes, references will be presented as source for a deeper understanding of the subject. We will discuss these topics to the level required to make this research more understandable.

2.2 Agent-Based Modeling and Simulation

Agent-Based Modeling and Simulation (ABMS) is known by many names: agent-based modeling, agent-based simulation (ABS), individual-based modeling (IBM), individual-oriented modeling (IOM). Although ABM, ABS, IBM and IOM are all widely-used acronyms, we have chosen to use ABMS throughout this discussion to refer to the model and simulation process and ABM for a model without simulation process.

ABMS dates from 1990. According to Chan, Son and Macal [31], the theoretical basis of ABMS lies mainly in cellular automata, complex system modeling, artificial life, and swarm intelligence. Siebers et al. [140] adapt the Shannon [138] definition of simulation in order to define ABMS: "ABMS is the process of designing an ABM of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system. In ABMs a complex system is represented by a collection of agents that are programmed to follow some (often very simple) behavior rules. Macal and North [89], say that ABMS "refers to a model in which the dynamic processes of agent interaction are simulated repeatedly over time, as in systems dynamics, time-stepped, discrete-event, and other types of simulation."

One of usages of ABMS is simulating interactions of autonomous agents to identify, explain, generate, and design emergent behaviors [31]. The emergent behavior comes from the interactions among the agents after they execute local and simple roles. ABMS is associated with a large number of different disciplines: military, physics, biology, chemistry, social science, economics, business etc.

As advantages of agent-based simulations, we can cite: (1) "they allow one to start off with the descriptive power of verbal argumentation and to determine the implications of different hypotheses." [66] Therefore what-if experimentation can be easily conducted with just parameter and configuration adjustments; (2) ABMS can be executed in a parallelized way [66]. It enables the execution of models that require thousand agents in order to represent its complexity; (3) ABMS allows the researcher to produce emergent phenomena; (4) ABMS are a suitable tool to study complex systems

2.2.1 Elements of Agent-Based Model

Agent-based model has three elements[91]: (1) agents, their attributes and behaviors; (2) agent relationships and methods of interaction. An underlying topology of connectedness defines how and with whom agents interact; and (3) agents' environment. Agents live in and interact with their environment, in addition to other agents. There are many definition about ABM:

- "An agent-based model, more generally, is a model in which agents repeatedly interact" [89].
- "Agent-based models can explicitly model the complexity arising from individual actions and interactions that arise in the real world. In other words, ABMS allows people to model their real-world systems of interest in ways that were either not possible or not readily accommodated using traditional modelling techniques, such as Discrete-Event Simulation (DES) or System Dynamics (SD)." [140]

Helbing and Balietti [66] and Macal and North [89] present a vast list of examples where ABM can be applied. Also, there are many studies of spatially-explicit ABM in the literature thus far, such as: bird flocks [126] [133], insect swarms [15] [80][14], mammal herds [64] and fish schools [71] [5] [156] [119] [63]. One common question is when uses ABM instead other modeling technique. Macal and North [89] say that ABM should be used when one or more of the following criteria are satisfied:

- the problem has a natural representation as being comprised of agents.

- there are decisions and behaviors that can be well-defined.
- it is important that agents have behaviors that reflect how individuals actually behave (if known).
- it is important that agents adapt and change their behaviors.
- it is important that agents learn and engage in dynamic strategic interactions.
- it is important that agents have a dynamic relationship with other agents, and agent relationships form, change, and decay.
- it is important to model the processes by which agents form organizations, and adaptation and learning are important at the organization level.
- it is important that agents have a spatial component to their behaviors and interactions.
- the past is no predictor of the future because the processes of growth and change are dynamic.
- scaling-up to arbitrary levels is important in terms of the number of agents, agent interactions and agent states.
- process structural change needs to be an endogenous result of the model, rather than an input to the model.

Also, Macal and North [91] present a list of questions to ask before developing an agent-based model.

Agent-based model also is known as individual-oriented model in the biologic field as referenced in previous works of our research group: [102, 101, 100, 43, 63, 144–146, 143, 16, 17]. Throughout this thesis, we will always use the term agent as a reference to both the agent and the individual. The use of the individual term will only be used when we have a direct citation.

2.2.2 Agent

We could not find in the literature an universal definition for Agent. Especially because this term is used in other correlated computer science areas. We believe the definition that fits best this research is the Macal and North [90]: "The single most important defining characteristic of an agent is its capability to act autonomously, that is, to act on its own without external direction in response to situations it encounters. Agents are endowed with behaviors that allow them to make independent decisions. Typically, agents are active, initiating their actions to achieve their internal goals, rather than merely passive, reactively responding to other agents and the environment."

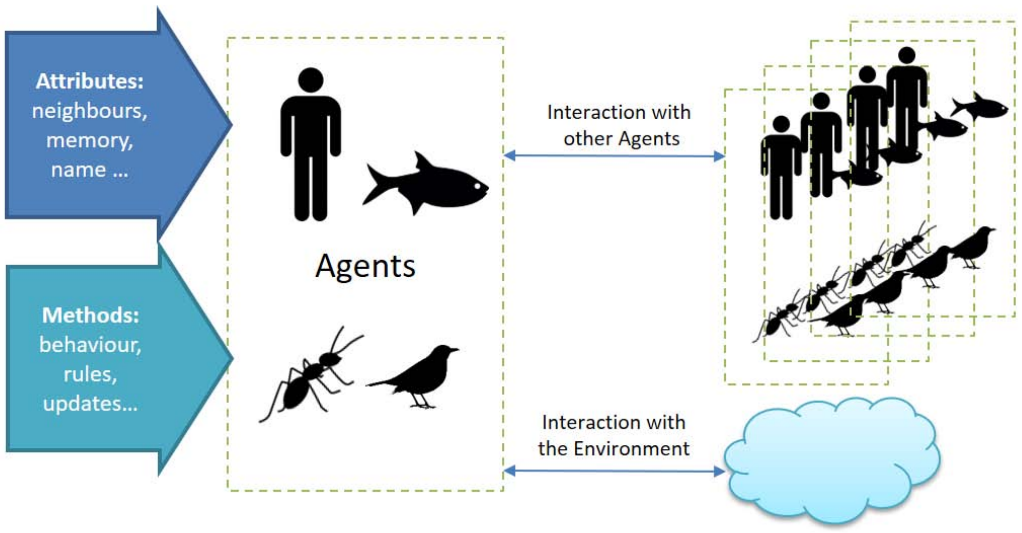


Figure 2.1: Typical agent in accordance with Macal and North.

In addition, Macal and North [90] consider agents to have certain essential characteristics:

- An agent is a self-contained, modular, and uniquely identifiable individual;
- An agent is autonomous and self-directed;
- An agent has a state that varies over time;
- An agent is social having dynamic interactions with other agents that influence its behavior;

Figure 2.1 shows a typical representation of an agent. It is possible to extract an emergent behavior through the interaction among agents and of the interaction of agents with the environment. As a consequence, it enables the researcher to analyze and reach conclusion of the model under study.

2.2.3 Environment

Environment is the space where the interactions of agents take place. The agents can interact with the environment and generate a reaction on agent.

The interactions among agents occur just with a limited number of agents out of all the agents in the population [89]. This subset is agent's neighbors. Agents are connected by a model's topology [90] (see Figure 2.2). The topology is strongly related with environment and the topology used depends on the problem modeled.

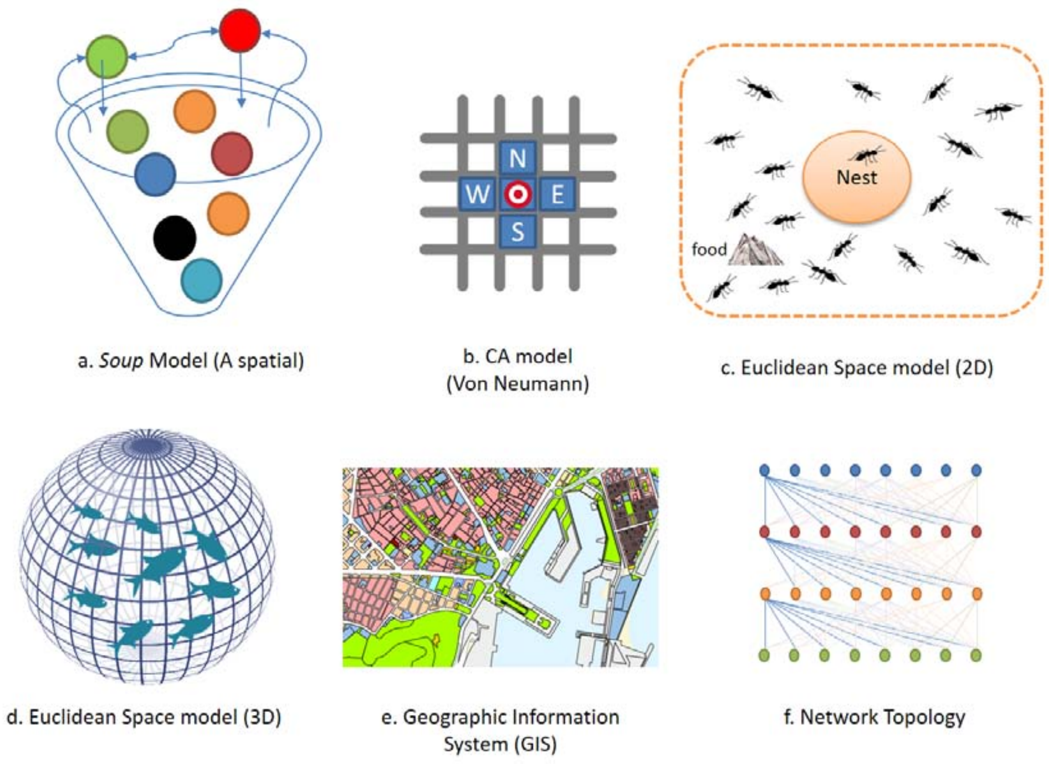

N
W
E
S
Nest
food
a. *Soup* Model (A spatial)
b. CA model (Von Neumann)
c. Euclidean Space model (2D)
d. Euclidean Space model (3D)
e. Geographic Information System (GIS)
f. Network Topology

Figure 2.2: Topologies for agent relationship.

2.2.4 Complex system

In accordance with Helbing and Balietti [66], complex system are systems with many interacting entities and non-linear interactions among them. In the ABM context, the entities are represented by agents and the interactions are defined by the agents' roles.

2.3 High Performance Computing

High performance computing (HPC) is the use of parallel and/or distributed processing techniques for solving complex computational problems such as the study of: universe, cancer, genome, aerodynamic, markets and several other problems with direct applications for humanity. Nowadays, with increase computational power it is possible to create more complex models to achieve results that are closer to reality and consequently help the development of science. In this section, we present basic HPC concepts.

2.3.1 Parallel computer architecture

Flynn's classical taxonomy [49] is a classification of parallel computers. This is one of the more widely used classifications. Flynn's Taxonomy distinguishes multi-processor computer architectures according to two independent dimensions: Instruction Stream and Data Stream. Each of these dimensions can have only one of two possible states: Single or Multiple [14].

MIMD is the architecture used to support this research. It can be classified in accordance with its memory architecture: shared memory, distributed memory and hybrid distributed-shared memory.

Shared memory

Shared memory parallel computers is characterized by the ability for all processors to access all memory as global address space. Multiple processors can operate independently but share the same memory resources. According to memory access times, shared memory can be classified as Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA) [14]. Figures 2.3 and 2.4 are a representation of this architecture.

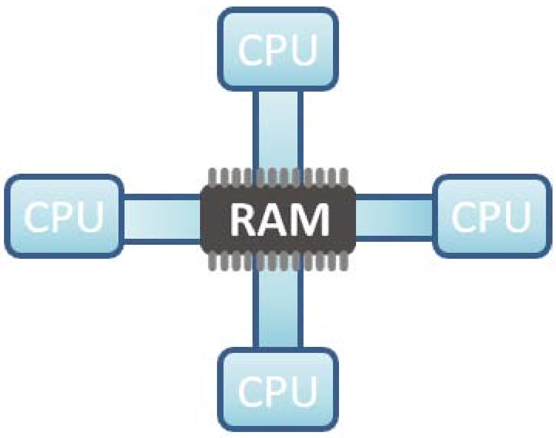


Figure 2.3: Shared memory UMA.

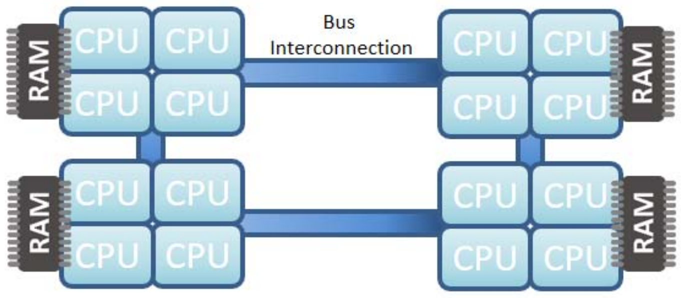


Figure 2.4: Shared memory NUMA.

Distributed memory

Distributed memory systems require a communication network to connect inter-processor memory. Each processor has its own local memory and there is no global address space across all processors [14]. The memory addresses belong to each processor. Mapping the memory addresses of another processor is not possible. The programmer has to define how and when data is communicated when a processor needs access to data in another processor [14].

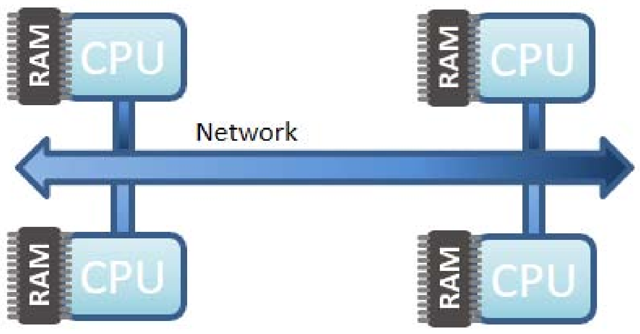


Figure 2.5: Distributed memory.

This architecture brings great advantages in the scalability for high performance computing because adding new units of computing is quite simple. However, the programmer is responsible for communication among processors and exchange of their data. Also, the exchange of data takes longer to access the data when the latest resides on a remote node.

Hybrid distributed-shared memory

Today, the most powerful computers use both shared and distributed memory architectures. The shared component can be a graphics processing units (GPU) and/or shared memory machine [14]. And the distributed memory component is a networking of multiple shared memory/GPU machines where each machine has its own memory and the access to non-local memory occurs by network [14].

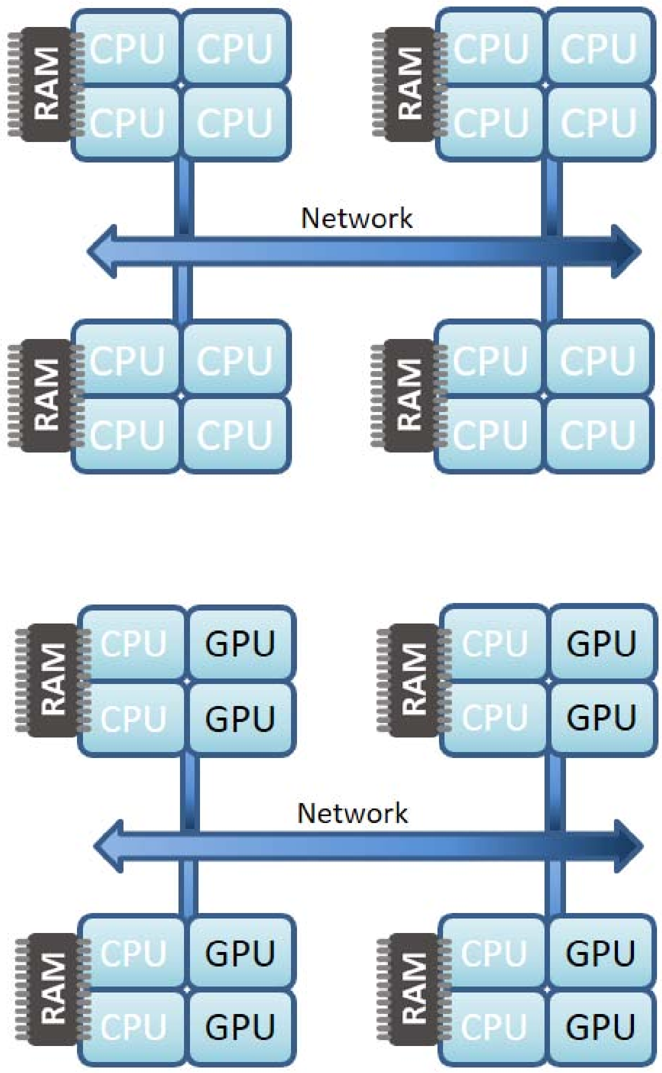


Figure 2.6: Hybrid distributed-shared memory with shared memory/GPU machines.

2.3.2 Parallel programming model

The HPC user disposes of several parallel programming model to access the parallel and distributed architecture. In this work, we take advantage of two parallel systems: shared memory and distributed memory. Therefore, two parallel programming model used in our HPC techniques will be discussed to follow.

Shared memory

Shared memory model is widely implemented using Pthreads or OpenMP [115]. In this section, we will focus only on OpenMP because we used it in some contributions that will be presented at a later time. OpenMP extends the programming language (C or Fortran) with directives to make parallelism. These directives enable users to create threads in order to explore the shared memory architecture. OpenMP follows the Fork-Join Model (Figure 2.7). OpenMP programs start with a single thread called master thread. At start of parallel region, master creates many threads (FORK) [13]. Statements in parallel block are executed in parallel by every thread. At end of parallel region, all threads synchronize, and join master thread (JOIN).

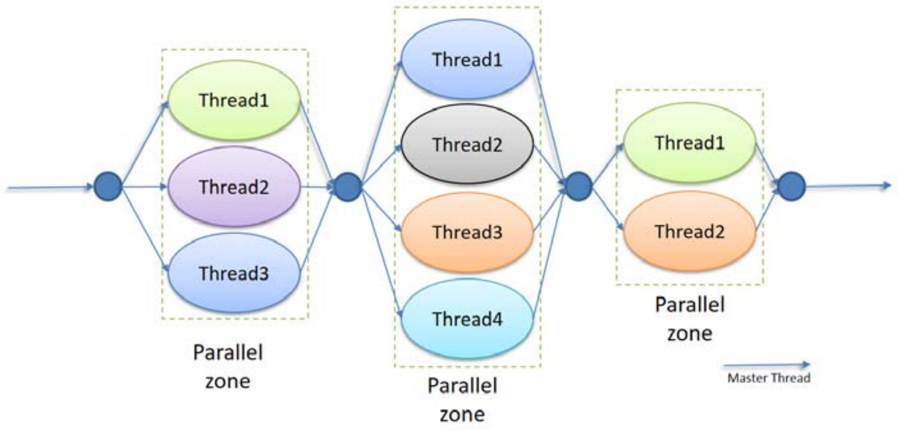


Figure 2.7: Fork-Join Model.

Shared memory architecture shares the same addressing memory among threads, where each thread can write and read in parallel and concurrently the same variables and data structures. This paradigm requires that lock mechanisms be applied in order to maintain data coherence. Generally, this kind of mechanism signifies a sequential access at variables and data structures. Therefore it can become a bottleneck in a parallel execution. This sequential access is called critical section and must be avoided otherwise the solution may be penalized with low performance. Also, shared memory architecture enables the solution to take advantage of the locality of the data using the cache hierarchy. Another interesting use of the OpenMP is to decrease the communication volume in the network because the communication occurs by memory copy.

Message passing

Message passing was designed for programming distributed memory systems. It provides mechanisms for sending messages among processes. The communication among the processes occurs by message passing. The standard for message passing is Message Passing Interface (MPI).

Distributed memory architecture uses an addressing memory for each process. In this paradigm, each process only has access to the variables that are inside of its addressing memory. No process can access directly a variable of another process in the distributed memory. Therefore, in this paradigm, there are no writing parallels and concurrently in the same variables and data structures. Distributed memory is a very appropriate solution for scalability [30] because it enables us to easily add other nodes in order to execute a solution. Communication among the processes is the main cause of the inefficiency of distributed memory solutions and it should be avoided trying to decrease the communication volume through data locality or partitioning data techniques.

2.3.3 Parallel programming paradigms

Parallel programming paradigm defines how the parallel application will be designed and how the parallel and distributed resources will be explored [26]. The most commonly used are: (1) Master/Worker; (2) Data Pipelining; (3) Divide-and-Conquer; and (4) Single Program Multiple Data (SPMD).

Master/Worker

Master/Worker, also called by other authors as Master/Slave and Task-Farming, has two entities: master and many workers. The master is responsible for decomposing the problem into small tasks and distributes these tasks among a collection of worker processes [26]. Also, master gathers the partial results in order to produce the result of the computation. The worker processes receive a task of the master, compute the task and send the result to the master, then it gets another task. This process continues until all tasks are finished. Usually, the communication takes place only between the master and the workers. This schema is presented in Figure 2.8.

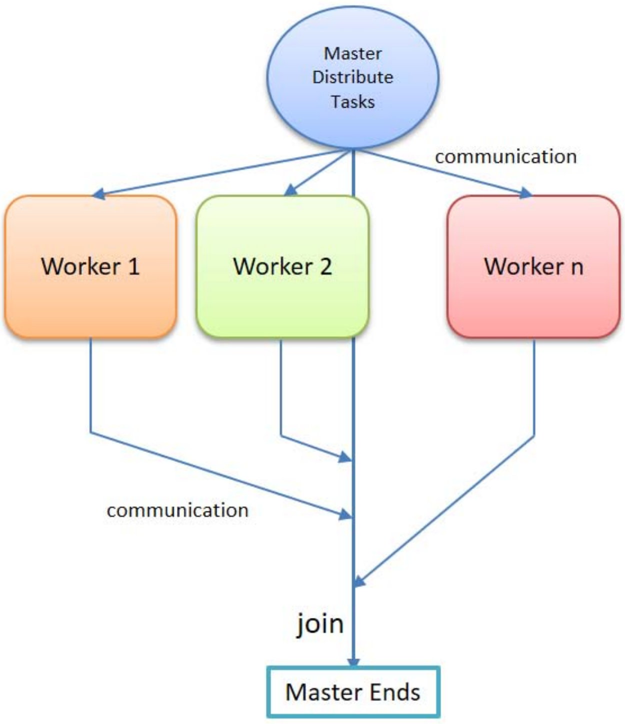


Figure 2.8: Master/Worker structure.

Data Pipelining

Data Pipelining is based on a functional decomposition. In this approach, the tasks of the algorithm are identified and each processor executes a small part of the total algorithm. Processes are organized in a pipeline and each process corresponds to a stage of the pipeline and it is responsible for a particular task [26]. The communication pattern can be very simple since the data flows between the adjacent stages of the pipeline [26]. Figure 2.9 shows Data Pipelining program structure.

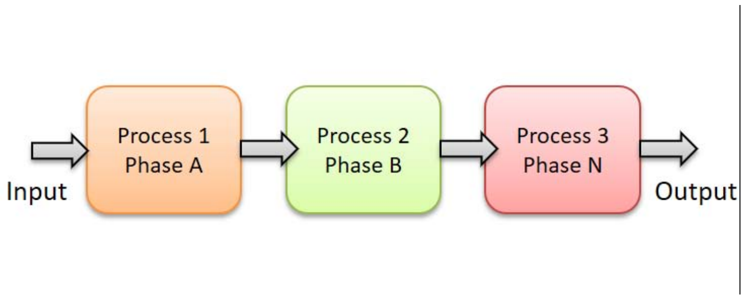


Figure 2.9: Data Pipelining structure.

Divide-and-Conquer

An Divide and Conquer problem is divided in subproblems where each subproblem is solved independently and their results are combined at final [26]. The parallelism is reached in this paradigm because the subproblems can be solved at the same time [26]. Figure 2.10 shows divide-and-conquer program structure and the main functions implemented in this paradigm: split, compute, and join.

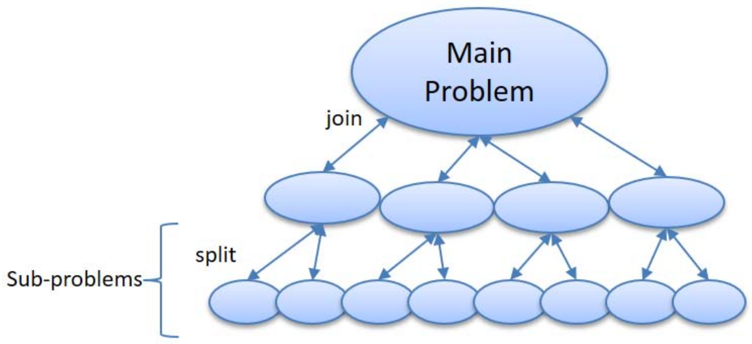


Figure 2.10: Divide-and-Conquer program structure.

Single Program Multiple Data

SPMD paradigm consists of a single executable running on each core [26]. SPMD program can give different roles to the cores through conditional branches [115]. As example 2.1:

Listing 2.1: Implementing task-parallelism in SPMD.

```
if (I am thread/process 0)
        do this;
else
        do that;
```

This example shows the implementation of task-parallelism [115] in the SPMD paradigm. Each task can be distributed among threads or processes. Example 2.2 shows that the data-parallelism also can be implemented using SPMD paradigm.

Listing 2.2: Implementing data-parallelism in SPMD.

```
if (I am thread/process 0)
        operate on the first half of the array;
else /* I am thread/process 1 */
        operate on the second half of the array;
```

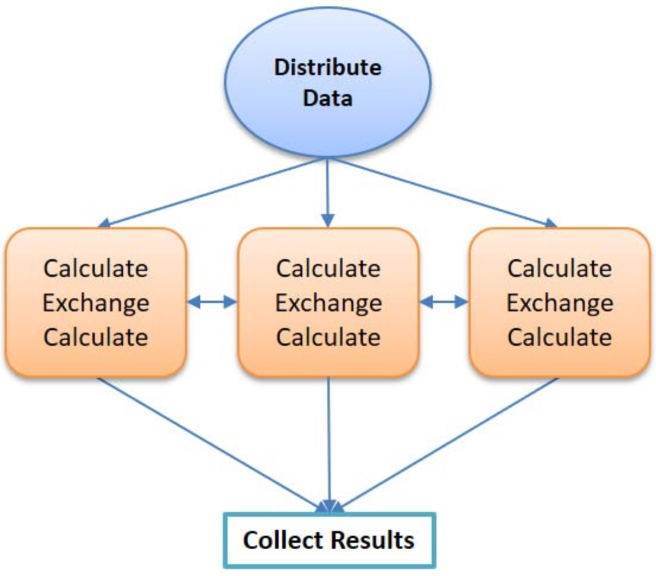

Distribute Data
Calculate Exchange Calculate
Calculate Exchange Calculate
Calculate Exchange Calculate
Collect Results

Figure 2.11: SPMD paradigm structure.

2.3.4 Performance measures

When a parallel solution is developed it is necessary discuss how fast it is. Four common measures are used: Execution time, Speedup, Efficiency and Scalability.

Execution time

Foster [57] defines execution time as the time that elapses from when the first processor starts executing on the problem to when the last processor completes execution. It is also referred to as the wall-clock time. The Execution time is composed of:

$$execution\ \ time = computation\ \ time\ \ +\ \ communication\ \ time\ \ +\ \ idle\ \ time$$

Speedup

In accordance with Pacheco [116], speedup is the ratio of the runtime of a serial solution to a problem to the parallel runtime.

$$S(n,p) = \frac{T_\sigma(n)}{T_\pi(n,p)} \tag{2.1}$$

Where $T_\sigma(n)$ represents the runtime of the serial program and $T_\pi(n,p)$ represents the runtime of the parallel program with p processes [116].

Efficiency

Efficiency is a measure of process utilization in a parallel program relative to the serial program [116]. In other words, it is the Speedup divided by the number of processes.

$$E(n,p) = \frac{S(n,p)}{p} \tag{2.2}$$

Where p represents the number of processes.

Scalability

The scalability can be categorized as either strong or weak. Strong scalability considers a fixed workload for a variable number of processors. Therefore, the amount of workload per processor decreases as the number of processors increases. The idea is to check if the execution time of the solution decreases as the number of processors increases. Weak scalability considers a constant workload per processor. Therefore, the total volume of the workload increases as the number of processors increases because the workload per unit of processor must remain constant. The aim is to observe if the execution time remains constant as the number of units of processing increases throughout execution.

2.4 High Performance Simulation

There are many definitions to simulation [155]. Simulation enables the modeling of a real problem with in order to either understand the behavior of the system or to evaluate various strategies through experiments[138].

Simulation can deal with real world problems. The challenge of solving real problems is that these problems are generally computationally complex. Therefore it is necessary to use high performance simulation in order to get these results. High performance simulation uses techniques of HPC to obtain simulations with execution time faster.

2.4.1 Modeling techniques

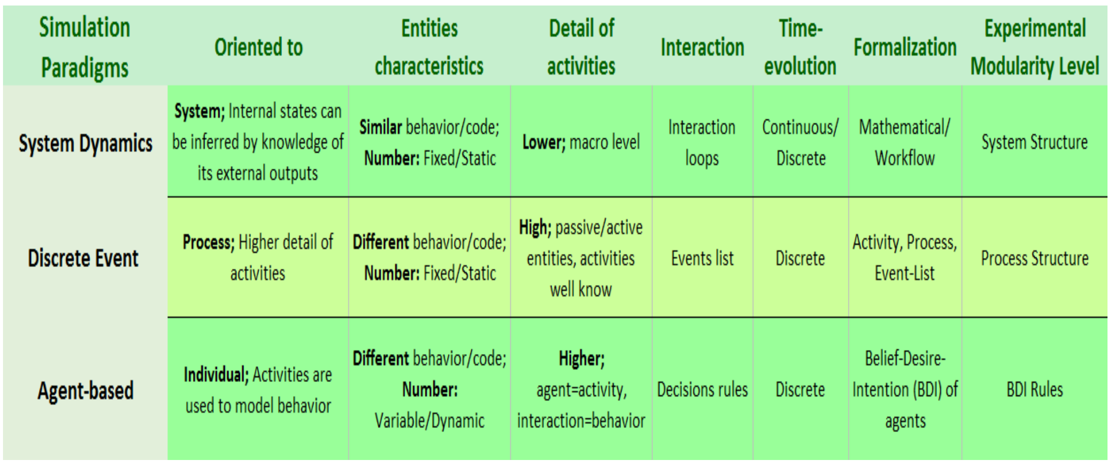
Over the years the scientific community has used modeling techniques in order to simulate its problems, such as Discrete-Event Simulation (DES), System Dynamics (SD) and Agent-Based Modeling and Simulation (ABMS).

In DES a problem is modeled as a series of discrete events where its entities change its states as time passes [95]. DES is useful for problems that consist of queuing simulations or complex network of queues, which processes follow stochastic distributions [140]. Other important DES characteristic is that it focuses on simulating events and their relationships of the underlying discrete-event dynamic system [31].

DES focus on the individual behavior of entities on the other hand SD focuses more on flows around networks [95]. Stocks are basic stores of objects and flows define the movement of items between different stocks in the system [95]. Therefore SD model is a network of stocks, flows and delays (time between the system measuring and an action) which may then depend on some general constants. A clear definition of SD is given by Brailsford and Hilton [22]: "System Dynamics models a system as a series of stocks and flows, in which the state changes are continuous. A system dynamics model views “entities” as a continuous quantity, rather like a fluid, flowing through a system of reservoirs or tanks connected by pipes. The rates of flow are controlled by valves, and so the time spent in each reservoir is modelled by fixing the rates of inflow and outflow."

Macal and North [90] define ABMS as an approach to modeling the dynamics of complex systems composed of autonomous, interacting agents. Such systems often self-organize themselves and create emergent order and behavior and are used to observe the collective effects of agent behaviors and interactions.

These three simulation paradigms have advantages and disadvantages and its uses should be analyzed in accordance with the problem we wish to model. A summary of main characteristics of three simulation paradigms [12] is presented in Figure 2.12.



Simulation Paradigms	Oriented to	Entities characteristics	Detail of activities	Interaction	Time-evolution	Formalization	Experimental Modularity Level
System Dynamics	**System;** Internal states can be inferred by knowledge of its external outputs	**Similar** behavior/code; **Number:** Fixed/Static	**Lower;** macro level	Interaction loops	Continuous/ Discrete	Mathematical/ Workflow	System Structure
Discrete Event	**Process;** Higher detail of activities	**Different** behavior/code; **Number:** Fixed/Static	**High;** passive/active entities, activities well know	Events list	Discrete	Activity, Process, Event-List	Process Structure
Agent-based	**Individual;** Activities are used to model behavior	**Different** behavior/code; **Number:** Variable/Dynamic	**Higher;** agent=activity, interaction=behavior	Decisions rules	Discrete	Belief-Desire-Intention (BDI) of agents	BDI Rules

Figure 2.12: Summary of main characteristics of three simulation paradigms.

Additional discussions and comparisons about these simulation paradigms can be found in [140, 31, 21, 22, 95].

2.4.2 Parallel and distributed simulation

Techniques of Parallel and Distributed Simulation (PDS) must be used in order to provide high performance simulation. There are five points that must be considered in this type of simulation: partitioning of data, policies of load and computing balance, synchronization of processes, migration and communication.

Partitioning data algorithm

Partitioning data algorithm distributes the simulate space over a distributed architecture. There are several partitioning data approaches and these depend on the characteristics of the problem modeled in order to get better performance results. Partitioning data approaches has a strong correlation with communication and load and compute balance.

The distribution of the data must consider the communication among processes. The approach must avoid communication processes as much as possible. Also, the distribution of the data must consider that all cores have the same volume of compute and workload. Oftentimes these constraints cannot be attended simultaneously.

Load and computing balance

Load and computing balance algorithms should assign to all cores the same workload and computing when a solution is executed by using a homogeneous architecture. It is important that all cores initiate and finalize their computing at the same time. This avoids performance degradation because a process does not have to wait for a slower process in order to continue its execution.

Synchronization

In parallel and distributed simulation, the space simulated is distributed among several processes. Each process computes its portion of the total simulated space in parallel and in an autonomous way. The simulation has to check its own state to avoid incoherence, or to exchange data among the processes.

Synchronization mechanisms broadly fall into two categories: conservative and optimistic. Banks [10] says that conservative approaches avoid the possibility of any causality error ever occurring and that the optimistic approaches use a detection and recovery approach. Causality errors are detected and a rollback mechanism is called to recover.

Migration

Migration is a process that occurs in parallel and distributed simulation due to the movement of agents from a process to another. Two of the reasons are: (1) movement of the agents in environment; and (2) triggered by other routine as partitioning, load balance, a behavior of an agent, etc. Migration can bring two types of problem: inconsistency of the simulation and performance problems.

The inconsistency of simulation happens when an agent does not consider the other agents which are located in other processes but are inside of its radius vision. Also, an inconsistency of simulation can be created when the agent does not consider the new context after the migration.

The other problem is performance degradation. The sending and receiving of agents between processes can bring performance problems specially if the processes are located in different nodes. If the processes are in the same node the operation occurs by memory copy. Otherwise, the processes occur by network communication.

Therefore, migration processes must take into account this issue when migration algorithms and partitioning of data are being proposed.

Communication

Communication among processes is the main source of performance problems in parallel and distributed solutions. Communication among processes is needed because many times a solution requires a high number of cores and these cores are distributed in different nodes. Some strategies can be used in order to decrease the impact of communication in a solution, such as:

- decrease the volume of data sent and received;
- try to use the locality in order to avoid communication of processes that are in different nodes;
- design a partitioning of data that takes into account the communication among the processes;
- design models that take into account the communication among the processes;
- improve the communication algorithms;
- develop approaches to avoid communication;

2.5 Software engineering

Software engineering provide information on how to design, implement and maintain a software using a systematic method. Four important concepts in this research are discussed as follows: object-oriented programming; interface; design pattern; and, extension of functionality and reusability. These concepts are fundamental to implement the methodology proposed in this work.

2.5.1 Object-oriented programming

Object-oriented programming (OOP) is a paradigm where all entities of a system are modeled as objects. Object has attributes and methods. The attributes contain the state of an object and the methods represent the actions that an object can execute. This paradigm is a natural fit for ABMS [140, 41]. An agent's internal state is easily represented in an object's fields while the agent's behavior is modeled using an object's methods [41]. This paradigm allows extensibility for modeling more agents, and more behaviors [140]. Also, it enables us to implement unlimited real-world entities and its relationships.

2.5.2 Interface

Interface is one of the most important concepts in this thesis. Interface is an overloaded and confusing concept in development. Interface, in object-oriented programming languages context, is a contract you should comply to in order to be considered that a "X" object is an instance of "Y" class. Interfaces do not define how to implement its methods, but it establishes which methods must be implemented. Therefore, when you say that an object implements an interface it means that you can always count on a set of public methods and properties. Thus, interface enables dynamically the interchange of behaviors, also it enables the design of solutions without previous knowledge of classes that are not still implemented.

2.5.3 Design pattern

Design patterns are cataloged solutions for common software design problems. Design patterns arise from designer experience in recurrent problems in a specific domain. Some of the advantages of using design patterns are: facility of extension and reuse of code, facility in maintainability of code, solution previously tested.

According to [60] design pattern can be divided into three categories: Creational, Structural, and Behavioral. The main design patterns used in care HPS are listed below followed by the definitions of Gamma et al.[60]:

- Factory method defines an interface for creating an object, but allows a class to define which subclass to instantiate.
- Singleton ensures that a class only has one instance, and provides a global point of access to it.
- Composite design pattern arranges objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Facade provides an unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- Strategy defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary no matter who the client is.

2.5.4 Extension of functionality and reusability

Object-oriented programming gives programmers two strong resources: extend a functionality and reusability.

To extend a functionality, the programmer can use two common features available in the main object-oriented programming languages: inheritance (class) or implement (interface). When we say that a class A inherits a class B, it means that the class A has all fields and methods of class B. When we say that a class A implements a interface B, it means that the class A has to implement all methods of interface B.

Using inheritance avoids re-writing a code and enables the new class to reuse all code developed previously. Interface is crucial for flexibility and reusability of code. The code must be implemented to receive interfaces as parameter. In almost all programming language that supports OOP, a class A can be inherited from one unique class but this class can implement several interfaces. Best practices in OOP state that programming must focus on interface instead of class. On the one hand, class means a strong coupling. On the other hand, interface enables classes of different domains to be used with the same code, if these classes implement the same interface.

Chapter 3

Fundamentals of PDS and ABMS

3.1 Introduction

This research is supported by three main areas (Figure 3.1): High Performance Computing, Parallel and distributed simulation and Agent-Based model. We will present the state of art in each of these areas focusing on the subjects that were researched and that helped us develop this work. The fundamentals of PDS and ABMS presented in this chapter will cover several issues that support our contributions presented later in Chapters 4 and 5.

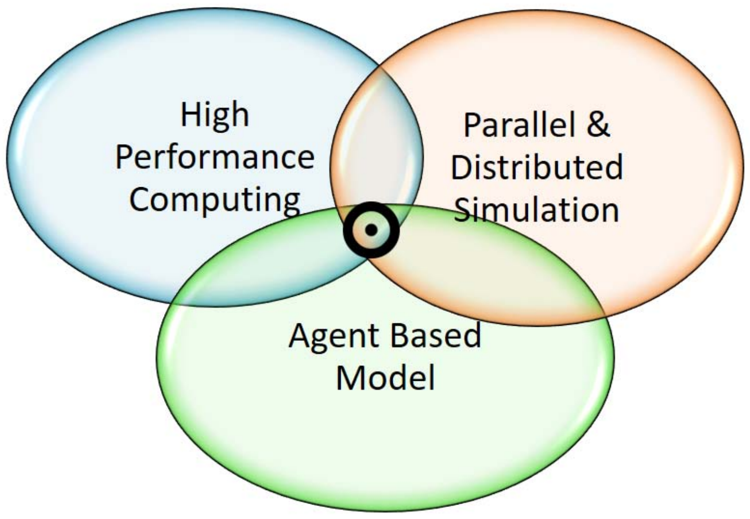


Figure 3.1: Context of the research.

3.2 Asynchronous and synchronous communication

An important issue in discrete simulation is the mechanism in which the state variables change as simulation time evolves. There are two types of time-advance approaches: Time-driven and event-driven. Time-driven approach consists of dividing the simulation time in a sequence of equal-sized steps, and the simulation clock is advanced from one time step to the next[58]. Event-driven approach consists of increasing the simulation clock when a event is processed. Events are marked with a timestamp and the simulation clock is advanced from a timestamp to the next. Furthermore, it is necessary to consider the time management protocol because this will ensure that the execution of the distributed simulation is properly synchronized.

Time management algorithms can be classified into conservative and optimistic [58]. In conservative algorithms, the process is blocked until all execution conditions are satisfied. In optimistic algorithms, the process will continue even if some execution condition is not fulfilled. Usually, optimistic algorithms include mechanisms which allow to recover from causality issues (e.g. rollbacks).

3.2.1 Asynchronous and synchronous MPI strategy

The MPI library has point-to-point communication routines that enable the programmer to use asynchronous and synchronous communication patterns. The fundamental difference between these two strategies is that asynchronous has non-blocking execution while synchronous has blocking execution. The blocking operations have a strong impact in communication time. When process A wants to send a message in synchronous mode to process B, for example, the process A has to wait until the process B is ready to receive the message. The process A will return after the message sent has been received by process B. On the other hand, non-blocking sends operations returns immediately, which allows process to continue doing computations while communication with another process is still pending. In this way, it is possible that computations and communications are overlapped.

In synchronous communication when a process sends a data the other process has to be ready to receive the data, so the Send and Receive functions should be executed in short time-slices otherwise the solution will waste time waiting. In time-driven simulation is important that all processors have the same workload in order to all processors reach the synchronization barrier together. In fact, time-driven simulations with conservative time-management algorithms have a synchronization barrier in each simulation step. This suggest that the synchronous MPI and BSP communication would be more efficient.

3.2.2 Bulk-synchronous parallel strategy

There are a lot of BSP implementation. We use BSPonMPI because it is a small communications library for bulk-synchronous parallel programming which consists of only 20 basic operations on top of MPI. The BSP computing model decouples communication and synchronization. A BSP computation consists of a sequence of parallel supersteps. Each superstep is subdivided into three ordered phases consisting of[68] [141]:

- simultaneous local computation in each process, using only values stored in the memory of its processor;
- communication actions amongst the processes, causing transfers of data between processors; and
- a synchronization barrier, which waits for all of the communication actions are completed, and then makes any data transferred visible in the local memories of the destination processes.

The BSP model allows the cost of program to be determined. A BSP program is composed by a lot of supersteps and each one is the sum of three terms: the maximum cost of the local computations on each processor, the cost of the global communication, and the cost of the synchronization barrier at the end of the superstep. So, the standard cost model can be defined by:

$$cost\_of\_superstep = w + hg + l \tag{3.1}$$

where w is a maximum over computation times, and the synchronization barrier must wait for the slowest process; g is total number of local operations performed by all processors by total number of words delivered by the communications network; h is a maximum over fan-in and fan-out of data; and and l is the synchronization overhead. Other cost models are shown in [141].

The BSP computation model has an important characteristic: it does not define how the communication and computing have to be implemented. The library that implement BSP will be responsible to define how to do the implementation of communication and computing.

MPI and BSP paradigms were designed for the development of scalable and portable parallel applications. In [68] is considered that the MPI's huge library makes efficient implementation far more problematic. On the other hand, BSP is more concise and consistent; this is why it is more easy to develop efficient parallel programs. Furthermore, BSP exposes a simple cost model (see eq. 1) for the transmission of bulked messages.

The MPI communication routines used in this work are based on pairwise sends and receives. Deadlocks in synchronous routines are possible and have to be treated because this solution does not provide a message buffer. In asynchronous communication routines it is possible to do computing while a process is receiving a message, but it is necessary to treat the race condition. In the BSP model explicit receives are not necessary because a synchronization barrier implies the end of all communication operations. However the data is available to process after the synchronization process has finalized.

3.3 Hybrid parallel programming

Hybrid parallel programming enables to explore the best that is offered by distributed and shared architecture in HPC [46]. The distributed architecture allows the scalability increasing the number of nodes. On the other hand, the shared architecture explores the efficiency, memory savings, the locality of data decreasing the communication and process migration. However, the hybrid programming model cannot be regarded as the ideal for all codes[27, 142]. Many studies were conducted to compare the hybrid parallel programming.

Cappello and Etiemble [27] consider that the performance of model depends on the level of shared memory model parallelization; the communication patterns; and the memory access patterns. They compare hybrid models with existing MPI codes by using the NAS 2.3 benchmark. In addition, they show that MPI approach is better for most of the benchmarks whereas the hybrid approach is better when the benchmark has high communication time and the level of parallelization is sufficient. Other authors [30, 79] also did the model programming comparison through NAS benchmark.

Chan and Yang [30] argue that MPI can be more favorable with the scalability of clusters. However, OpenMP can favor the speed of shared memory. In addition, the application performance can be affected by the type of problem that is being solved and its size. They show that the effect of MPI communication is the main weakness of this programming model. And finally, they conclude that OpenMP prevails over MPI especially with using a multi-core processor. Hybrid programming models can match better the architecture characteristics of an SMP cluster, and that would replace message passing communication with synchronized thread-level memory access [47].

Smith and Bull [142] discuss situations where the hybrid model may be more efficient than its corresponding MPI implementation on an SMP cluster, such as: codes which scale poorly with MPI; poorly optimised intra-node MPI; poor scaling of the MPI implementation. Chow and Hysom [36] try to understand the performance of hybrid MPI and OpenMP programs. These authors give an overall review about factors and parameters, which affect the hybrid program performance. Jin et al. [79] bring that the no well-defined interaction between MPI processes and OpenMP threads impedes a wider adoption of hybrid programming. In fact, this limitation requires a strong effort of the programmer in order to develop hybrid solutions. However, the MPI community are working on to improve the interface with threads in MPI 3 [79].

The literature presents many aspects that influence the performance of hybrid solutions. Each approach could have advantages or disadvantages depending on the: architecture [36, 65, 122], network interconnection [35], and application [27, 36]. It is possible to find many researches reporting comparisons between programming model and orientations about how to explore appropriately these techniques. For a list with additional references see Adhianto and Chapman [1]. These authors show other related works that: has shown performance improvement using the hybrid model; others that report poor hybrid performance; and some which present minor benefits to adding OpenMP to an MPI program.

3.4 Partitioning of the data

The partitioning of the data in parallel and distributed simulation is a problem that affects the efficiency of these applications. This occurs because the chosen strategy might require an increase in the barriers and communication volume among the processes in order to maintain the coherence of the simulation. The partitioning techniques try to distribute the workload in a homogeneous way among the distributed processes in order to finalize all of the tasks together and decrease the total execution time. However, this is a hard task, because the communication volume, workload, shared resources, synchronization and other aspects should be considered in the partitioning strategies. In general, the solutions stabilized a trade-off of these factors.

We have been researching partitioning strategies with the ABMS by using parallel and distributed simulation. ABM considers that all agents have simple roles, and a collective behavior emerges of agent interactions. In addition, the agents interact with the environment. Therefore, these two important issues (agents and environment) must be considered in order to partition ABMS parallel and distributed simulations, because the partitioning strategy in ABM depends on how the iteration between the agents and the environment is modeled. As a result, this iteration can represent an exchange of information among the processes and, consequently, increase the communication volume. Therefore, the relationship between the agents, the relationship between agents and the environment, and how to distribute the environment in the architecture [159] are all issues that influence the partitioning strategy. As an example, we can reference two different ABM scenarios where different partitioning approaches can be applied: fish schooling [145] and ant colonies[162]. In Solar [145], the fish simply avoids a collision with the environment. Changes do not occur due to interaction with the agent; therefore, the environment could be shared among the processes without having an impact on the communication volume. The Replicate Environment approach, defined by [159], is very well applied in this context, because the environment is static. Therefore, the environment is replicated in all processes. However, the environment in the ant colony is very dynamic. The quantity of food in the environment decreases when the ants find food. In this situation, sharing the food of the environment with all of the processes would be a bad strategy, because all of the processes would have to be notified by broadcast whenever that occurred. A possible partitioning approach for this case is to divide the environment among the processes. Each process must have a consistent view of the objects and agents that reside and interact in it[159, 98].

3.4.1 Cluster-based partitioning

Solar et al. [145] have implemented a partitioning algorithm using a cluster-based approach. This approach consists in assigning to each node a fixed set of agents. The partitioning method uses a hybrid partitioning method based on Voronoi diagrams and covering radius criterion. Voronoi diagram is a data structure in computational geometry where given some number of objects in the space, their Voronoi diagram divides the space according to the nearest-neighbor rule [7]. In this work [145], each object is represented by an agent (fish) and it is associated with the area closest to it. Covering radius criterion consists in trying to bound the area c_i by considering a sphere centered at c_i. These centroids (c_i) contain all the objects of the problem domain that lie in the area [32].

The agents are associated with a position in a three-dimensional euclidean space and together with the euclidean distance generating a metric space. The distance between objects of this metric space is defined by a set of objects X subset of the universe of valid objects U and a distance function $d : X^2 \rightarrow R$, so $\forall x, y, z \in X$, must be met the following conditions: Positiveness: $d(x,y) \geq 0, d(x,y) = 0 \Rightarrow x = y$; Symmetry: $d(x,y) = d(y,x)$; and Triangular inequality: $d(x,y) + d(y,z) \geq d(x,z)$. These conditions determine the visibility of agents, which allowing use of similarity or proximity within the distributed simulation. Therefore the partitioning method consist of two phases: the centroids selection by means of covering radius criterion which ensures it a set of centroids far away enough to the others; and the space decomposition by means of Voronoi diagrams which allow it to define similar size areas with similar number of agents.

The data structure used to store agents is called fixed-radius list of clusters[32]. As presented in the previous section, the authors are using a hybrid Voronoi diagram/covering

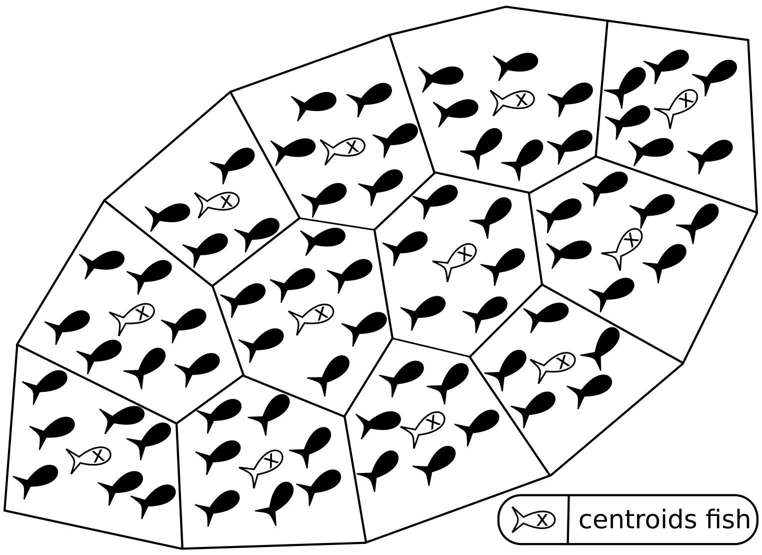


Figure 3.2: Covering radius and Voronoi diagram criterion to cluster-based partitioning

radius as build criterion for a fixed-radius list of clusters[117]. The radius is fixed in function of the maximum fish vision area. This allow them defining areas in which agents can interact only with agents belonging to adjacent areas. The data structure is formed by a linked list of clusters, Figure 3.3, implemented in C++. Each cluster object is composed of several data such as: 1) *centroid* - which is the most representative element of the cluster; 2) *pid* - the processor identifier indicates in which node each cluster is stored; 3) *bucket* - in which agents belonging to the cluster are stored; 4) *cid* - the cluster identifier indicating the cluster position in the list; and 5) information about the distances to other clusters.

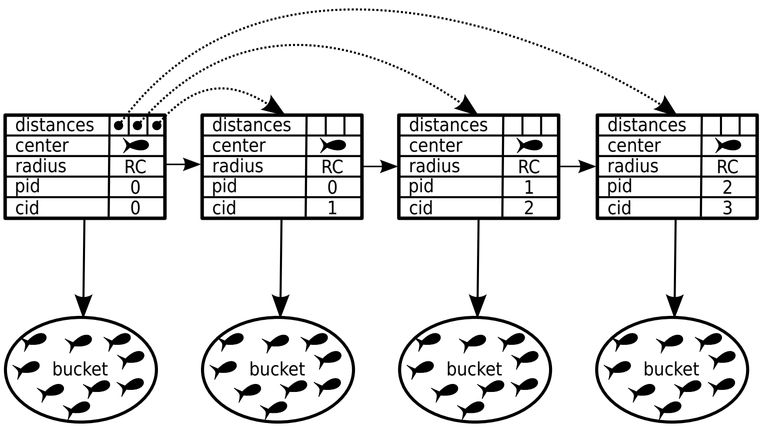


Figure 3.3: List of clusters used to distribute the agents through distributed architecture.

The list of clusters is distributed through the distributed architecture. The distribution model used is based on proximity concept. The algorithm distributes the list header together a fixed set of different clusters to each node. This fixed set of clusters is determined by the maximum number of agents that each node can have ($\frac{N_{agents}}{N_{processors}}$), and how close they are to each other. The aim is assigning contiguous groups of clusters to each node in order to decrease communication and computing involved in selecting data to transfer. The authors group clusters by proximity into sets of clusters that they have called *meta-cluster*.

3.4.2 Strip partitioning

One interesting way of dividing the environment is to create strip partitions. According to Saule [134], strip shapes implicitly minimize communications. The communication in this type of partitioning is restricted to neighboring partitions. Much research on rectilinear and strip partitioning has been done. In summary, these works try to partition workloads on processors, minimizing execution time. Nicol[108] and Saule [134] represent the workload as a matrix that is divided into rectangular fixed-size regions. These techniques are applied to solve numerical problems; however, many concepts and ideas can be applied and can be useful in an ABM context. Nicol [108] studies three different partitioning methods: iterative refinement, binary dissection and jagged rectilinear. Jagged rectilinear partitioning divides the domain into N strips, and each strip is divided into M rectangles. The strips and rectangles have approximately the same load. This allows for the calculation of the overall cost of a partition to a processor. Rao [124] reports degradation of performance when agents span two or more partitions. He presents an agent proxy approach in order to handle the spatially-explicit problem. Saule [134] agrees with Rao and mentions that spatially partitioning localized computation is the key to obtaining a low communication volume in the distributed processes. The author also suggests that each processor should be assigned a connected and compact computational work space. This work space is divided into rectangular regions. Thus, each process communicates just with its neighboring processes. Peschlow et al. [121] present a dynamic partitioning of LP for optimistic distributed simulation. The authors identify the unbalance and move the LP to another host. Bahulkar et al. [9] propose a partitioning solution taking dynamic model behavior into account. Deelman [44] execute a strip partitioning where strips migrate to other process and the strips are resized during the load balance.

Many application areas, such as biology and physics, have spatially dependent problems modeled in the Euclidean Space (ES) that require ABMS parallel and distributed solutions. The distribution of the environment represented by ES must take into account that the environment has resources (objects) that are consumed and used by all of the agents. In order to avoid communication among the processes, the partitioning must use the approach where the environment is divided among the processes and just the local environment is kept updated. This strategy avoids excessive communication among the processes, and the agent iteration occurs only with objects of its partition space.

3.5 Output analysis

One of the most important step in simulation is the output analysis. Since simulation deals with stochastic results it is a requirement to use statistical methods in order to check and make conclusions about the results obtained. Basically, two distinct phases must be analyzed in simulations: transient and steady. In the first one, the initial data must be discarded because these data produce an initialization bias. In the latter phase, when the simulation arrives into steady, the data must be produced in order to produce statistical estimator with appropriate confidence intervals. In transient phase, the problem is to figure out the number of observations to discard. In the steady state phase, the problem is find the run length which has trustworthy results.

There are three main pitfalls into output-data analysis [82]: underestimation of variances and standard deviations when the replication are not independent; failure to define warm-up period; and failure to determine the statistical precision of simulation output statistics by the use of a confidence interval.

Many output analysis methods are proposed, compared and analyzed. In [94], they compare six methods to detect the transient phase length in a non-terminating simulation. They are the following: 1) Welch's method [160] is the simplest technique to determine the warm-up however this method has a high subjectively because it is a graphical method. As the simulation has stochastic behavior, the graphical procedure requires multiple replications of the simulation in a pilot study [82], [94]; 2) SPC method requires multiple replications and at least 20 batch means must pass for normality and correlation tests; 3) The Randomization Test verifies the null hypothesis where the mean is unchanged throughout the run. It is a statistical method that no requires the normality of data; 4) The Conway Rule method truncate the initial data of output in order to reduce bias. In this method the number and length of the replications must be specified; 5) Crossing of the Means Rule method establishes the truncation point counting the number of crossings of the mean until reach a pre-specified value; 6) Marginal Standard Error Rule-5 (MSER-5) defines the truncation point considering the batch size, run length and five as the number of batches. In [120] they suggest two new algorithms for using the MSER statistic and compare them by using mean squared error. They show that MSER is asymptotically proportional to the mean squared error and therefore they argue that it is a good solution for initial transient algorithms. The removal of the effect of the initial conditions is a challenging problem. In [2] it is suggests a method for eliminating the bias by truncating the initial observations. In [161] they compare the performance of five well-known truncation heuristics. Some authors ignore the first batches in order to reduce the potential effects of initialization bias [148].

In order to analyze the steady state phase, there are two common techniques used, which are: replications [82] or batch [148], [105]. Replications are independent simulations, which must be executed many times and their results are statistically analyzed at the end. Batch technique is more applied in a non-terminating simulation. In this method, the observations of simulation are divided in batches, and each batch is considered as a replication. Methods of batch means and independent replications in the context of non-terminating simulation output analysis are compared by [3]. Batch methods improve the effects of initialization bias but produces batch means that often are correlated [3]. Consequently, the correlation between these batch must be checked. Determine the number, correlation and size of batch are a challenge in output analysis. In [135] he shows that number of batch need not be too large in order to decrease the risk of an invalid confidence interval. A vast literature about the use of batch size effect is available in [106]. Some methods in [2] and [150] assume approximately normally distributed because they consider sufficiently large run length. Other methods increase batch size until the batch means pass the Shapiro-Wilk test [148]. In addition, in [106] he observes that many batching algorithms tried to find a large, or even the largest, value of replications that also generate a valid confidence interval.

Several output analysis methods are not recommended for realistic simulations. Because they require large run length and consequently need huge computational resources. Depending on simulation parameters the total execution time of simulation to obtain the results to analysis can be not feasible. Therefore we must use statistical methods that neither demand many batch non large run length but that are statistically reliable.

3.6 Epidemiological topics

One of the aims of this work is to apply the methodology proposed to a real problem with a social impact. The ABM that will be presented later in Section 4.2.4 was developed with this purpose. It models the pupal productivity of the Aedes Aegypti mosquito, which is the vector transmitter of Dengue and Zika. Therefore, we had to understand and study some epidemiological issues.

3.6.1 *Aedes Aegypti* mosquito

Aedes aegypti is related with transmission of three arbovirus: Dengue, Chikungunya and Zika. Dengue is a febrile disease whose clinical manifestation can vary from a benign viral syndrome to a fatal hemorrhagic shock. The world incidence increased 30-fold in the last 50 years, expanding geographically to over 100 countries (Figure 3.4), with approximately 3 billion people living in countries where there is an epidemic. This results in an annual register of 50 million infections worldwide [168]. Chikungunya is a viral disease whose main symptoms are fever and severe joint pain. Zika is becoming a pandemic and it is predominantly a disease slight and asymptomatic. However, Zika is correlated with Guillain–Barré syndrome and microcephaly [1] [48].

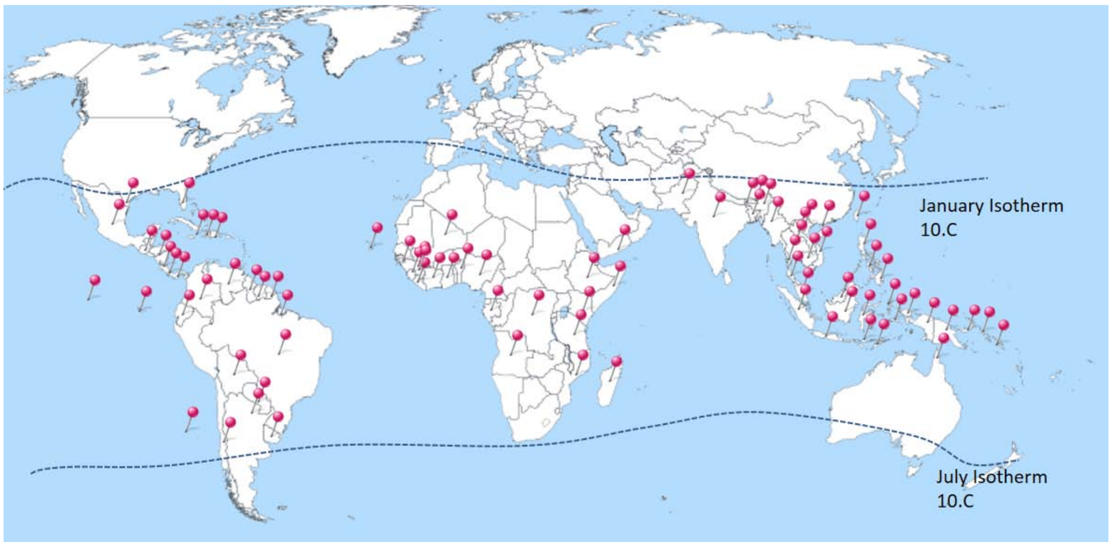


Figure 3.4: Countries or areas where dengue has been reported or that are at risk of dengue.

3.6.2 Pupal productivity

The *Aedes aegypti* control programs focus their actions on the elimination of breeding sites and the reduction of the vector population. However, one of the main problems is identifying the population density of the mosquitoes required to start or to maintain the transmission of dengue [50]. In most of the dengue control programs, as example in [23], traditional entomological indices are used, such as: House Index (HI), Container Index (CI) and Breteau

[1]On 01 February 2016, WHO declares global emergency for Zika. "The infection has been linked to cases of microcephaly, in which babies are born with underdeveloped brains... There have been around 4,000 reported cases of microcephaly in Brazil alone since October.[11]". "However, more investigation is needed to better understand the relationship between microcephaly in babies and the Zika virus. Other potential causes are also being investigated." [169]

Index (BI). These indices evaluate just the positivity, in other words, the presence or absence of the vector in containers. However, some studies point out that it is necessary to evaluate the pupal productivity of containers with the aim of identifying those that contribute most to the adult mosquito population [50].

The pupal productivity has been applied in entomologic assessments with promising results. This method identifies the most epidemiologically relevant containers, in other words, the containers that contribute most to the production of mosquitoes. Previous studies indicate that assessment of pupal productivity would be the most adequate method to assess the risk and operationalize control activities ([168], [24], [52], [53]). There are several reasons for adopting pupal productivity to assess the mosquito population. First, the larval stage is temporally far from the adult stage, and there are many biological implications in this stage interval that are not considered ([168], [153], [51]). The other reason is that, with the assessment of pupal productivity, it is possible to define the pupae per person (PPP) index. This index is the relation between the number of pupae and the number of people in a specific area. The PPP index is a good alternative to estimating the female mosquito population, because it is highly correlated with the density of adult mosquitoes. In addition, it enables us to assess the percentage contribution of each type of container ([52], [24], [53], [50]).

The major drawback of the assessment productivity method is the identification of breeding sites in the area in order to identify the containers. This activity generally signifies a great effort on the part of the health agents. Therefore, through simulation, we could use the results and calculations of the productivity of an area and then apply them to other similar areas, thus rationalizing resources ([53], [50]). The most productive container can vary from place to place, but it can be estimated by cross-sectional study. Generally, the types of containers of a specific geographical area do not change, although the pupal productivity can be very dynamic [154].

Throughout this work, the term: productivity will refer to the ability of a container to produce pupae of *Aedes Aegypti* mosquitoes. Low productivity means few pupae. High productivity means many pupae.

3.6.3 Density of the *Aedes Aegypti*

Focks says that is necessary to evaluate the pupal productivity of containers with the aim of identifying those that contribute most to the adult mosquito population [50]. In the literature we can find several approaches in order to study the density of the mosquito. A statistical approach, as Bayesian analysis, is carried out by [158]. Villela et al. [158] present a hierarchical probabilistic model for the estimation of female *Aedes aegypti* abundance from Mark–Release–Recapture (MRR) studies. Also, they use an agent-based model to generate data from MRR experiments which are used to test the proposed model. An important aspect that should be considered in mosquitoes density is the capacity of mosquito dispersion. Honório et al. [70] cite several studies on the dispersal *Aedes aegypti* that show female fly between 100 and 500 meters. Flight radius of the mosquito is a crucial parameter in the outbreak problem, because it defines the limit of the ecosystem where the mosquito might interact. Focks et al. developed a model that describe the population dynamics and the epidemiology of dengue viruses in the urban environment [54]. This model is called: container-inhabiting mosquito simulation model (CIMSiM) which is a weather-driven, dynamic life table simulation model of *Aedes aegypti* [55, 56]. CimSim is a tool that model the mosquito population using a variety of parameters. Due to the epidemiological results of this tool, its supports many studies [165] in the literature. Other tool for population estimation of the mosquito is presented by [93]. Skeeter Buster is based on the algorithms used in the CIMSiM.

3.6.4 Agent-based model for epidemiological questions

The agent-based model has consistently been used by the academic community in order to study epidemiological questions such as mosquito populations, the risk of transmission, and the outbreak of dengue. One reason is that it is a complex problem that has dynamic iterations and stochastic events. In the literature, we can find several studies that use ABMS in order to combat, prevent, and observe the vector mosquitoes transmitters. Here are some ABMS solutions for several epidemiological problems: [104] simulate the spread of sleeping sickness caused by the tsetse fly; [137] try to identify control mechanisms against tuberculosis; [128] propose a multi-agent model for vector-borne diseases; [75] focus on the simulation of the population dynamics and the population control strategies of the *Aedes aegypti* mosquito; [127] study the epidemiological and evolutionary dynamics of avian influenza viruses; [84] developed a framework for the planning of control strategies for dengue fever. According to [84], this framework uses models that have already been calibrated and validated in real case studies.

We cannot find in the literature, until the current date, any ABM that takes into account the pupal productivity of the container. However, we find an ABM for the spread of dengue in [76]. The authors model the possibility of the human agent becoming infected by hemorrhagic dengue fever and dying. The major drawback of this approach is that they consider that all pupae have an 83% chance of becoming adult mosquitoes. In other words, the same number of adult mosquitoes emerge from different containers. However the chance of a mosquito becoming an adult depends on the containers where the eggs were laid. Therefore, this is not a realistic situation, as pointed out by [24]. As an example, in her study, [24] finds that sanitation fixtures and metallic item containers had a pupal productivity of 1.8% and 32.9%, respectively. These percentages correspond at absolute values of 0.9 and 7.4 pupa per container, respectively. Thus, the estimation of adult mosquitoes can be completely different if we compare the model of [76] with the results of [24].

An interesting systematic review about modeling tools for dengue risk mapping can be found in [85]. This systematic review presents several strategies and approaches in order to study the risk of dengue. The authors consider that the prediction of spatial and spatio-temporal dengue risk is complex to model and depends on multiple and diverse factors. In addition, predictive models still lack reliability in anticipating outbreaks. In summary, many models and solutions have been proposed by the academic community. However, many questions remain open and require research. We try to understand the pupal productivity in dengue outbreaks through a model presented in Section 4.2.4.

3.7 ABMS solutions

It is important that we discuss the design of some ABMS tools as well as methodology, techniques, solution and orientation has been reported by some papers. In this section, we will focus on how far the support of some well-known ABMS tools goes to support HPC experiments. Therefore, first of all, we executed a survey in simulation tools for agent-based models. We investigate if the ABMS tool allows for the experimentation of HPC in an ABM context and how it was designed. Also, some programming practices and orientation will be presented because these can also influence the design of tool. In addition, we will discuss how some important aspects of the agent-based parallel and distributed simulation should be addressed. As a reference, we will use the following tools: D-Mason [42], FLAME [69], Pandora [130] and Repast HPC [40] because these tools enable the execution of models using HPC.

3.7.1 Simulation tools for agent-based model

Based on these surveys [4, 109, 151], first of all, we have listed nominally all of the tools and checked which tools have some features and purposes of support simulations. We have found around 80 different tools referenced in the literature. Some important aspects of these tools were noticed in these papers: 1) we have observed that some authors overlap the concepts of Multi-Agent System (MAS) and Agent-Based Model, although MAS is not the same as ABM. This misunderstanding of definitions is widely discussed in [107]. Often, Multi-Agent Systems use software agents that are autonomous, cooperate among themselves and learn with other agents and the environment. In contrast, in an agent-based model context, its agents have simple roles, and the emergent and collective behaviors are the issues in our study; 2) many other tools are applications for developing simulations rather than simulators; 3) the majority of them have the study of agent-based models as their main focus and they do not support any parallel and/or distributed agent-based simulations nor do they support high performance computing for these types of models. The tools that are the exception to this are compared with our methodology in Section 5.6; and, finally, 4) they cover a huge range of primary domains such as education, logistics, biology, chemistry, archaeology, and others.

We then selected tools that fit with the criteria: supports parallel and/or distributed simulation and/or support for agent-based simulation. The tools found are: Ascape [118], D-Mason [42], EcoLab [147], FLAME [69], MobiDyc [62], Netlogo[163], Pandora [130], Repast HPC [40], and Swarm [67, 99]. Ascape is a framework for developing general-purpose agent-based models [118]. D-Mason [42] is a parallel extension of the MASON [87] library for writing and running simulations of agent-based simulation models. EcoLab [147] was initially projected to support an abstract ecology model [4]. It supports the partitioning of agents over processors and tolerance failure support. EcoLab has a component that allows reflection to be adding to C++ language. FLAME (FLexible Agent-based Modelling Environment) is a code-generated tool that allows the user to define its agent-based model and automatically generates C code optimised for efficient parallel processing [69]. MobiDyc aims to develop agent-based modelling in the fields of ecology and biology. MobiDyc enables the user with no computer skills to develop their model through simple primitives [62]. Netlogo [163] was born as an educational project. Netlogo has a friendly GUI and an easy programming API for developing new ABMs. Netlogo does not support HPC features; however, it is a powerful ABMS tool with a huge user community. Pandora [130] is an ABMS tool of the Barcelona Supercomputer Center that has been used in several archaeology projects. Repast HPC is an agent-based modeling and simulation toolkit [40]. Repast HPC implements the main concepts of Repast Simphony with support for parallel distributed environments. Repast HPC was designed with the goal of obtaining extreme scalability for the TOP500-class supercomputer [152]. Swarm was the first ABMS software development environment created by the Santa Fe Institute. The agents are organized through a collection called "swarm" that has a scheduled event for these agents [99]. Swarm [99], which uses Objective C, does not have any parallel capabilities [38].

In addition, outside of the ABMS context, we have found the SimGrid [29]. This tool is used as a scientific instrument focused on the simulation of distributed applications in heterogeneous distributed environments, such as Grids, P2P systems and Cloud. The first version of SimGrid dates back to 1998 [28] and, today, it is a tool widely used as a scientific instrument with 159 papers published between 2000 and 2016 [74]. Basically, these publications present new approaches, strategies and solutions for the simulation of distributed applications in heterogeneous distributed environments. This expressive number of published papers shows the importance of tools like SimGrid.

As presented, in the literature [109, 4, 38, 110], we can find surveys and reviews on ABMS tools that support several issues, features, and contexts. A few of them use HPC to execute their models. However, none of these tools are designed with the aim of being a scientific instrument that facilitates the research of HPC for agent-based models that demand high performance solutions. Of course, many of these tools are open source and, consequently, other features can be added, but the time invested from programming is too high and requires high level knowledge of the tool and programming. And finally, [123] gives us interesting recommendations for the development of ABMS tools.

3.7.2 Design agent behavior

Agent behavior is the most important component of ABMS and generally only the application area researchers have enough knowledge to model agent behavior. Four strategies can be offered for application area researchers to represent an agent inside of a tool: flow diagram, primitives, Extensible Mark-up Language (XML) files and coding. These strategies can be used alone or the tool can use a combination of them.

Flow diagram is a graphic approach that enables the use of drag-drop components that represent each behavior or sub-steps of a behavior of an agent. Each component must be implemented with what the agent must do. The flow diagram can be composed of several components that define different built-in actions where the user needs to fill its predefined properties and this enables it to generate the code automatically. Another solution is to give the user the possibility of implementing a simple code. This code might be proprietary or a well-known programming language. Repast is an example of a tool that implements this solution. In these papers [112, 110], it is possible to have an idea of the advantages of using this strategy as a design for modeling the agent behavior. Other papers also present this solution for modeling [103].

Another design option is the use of primitives. The tool implements some basic primitives in order to enable the modeling of behavior. In this case, the user has to learn each proprietary primitive. These primitives can be designed to give the possibility of reconfiguration for change or to specify a behavior or action, as is done in [62]. Another strategy is to use XML files in order to define the agents and their behavior. FLAME uses this approach to define its agents, model and environment [69]. XML is capable of describing many different types of data and also provides a standard way to structure a text [69]. It is possible to define the structure and the legal elements and attributes of an XML document. Therefore, the tool has the means to guide the user in their agent's definitions. Lastly, coding is the most flexible and powerful strategy for representing an agent; however, it is also the most complicated for users with no programming skills. Through well specified interfaces, it is possible to define the rules that the user has to follow.

For a qualitative and appropriate comparison among these approaches, it would be interesting to use experimental software engineering [166]. However, using our experience as criteria, we understand that flow diagram is an easier approach for users that do not have programming skills. In terms of the possibility of the representation and extension of agent behaviors, we believe that coding is the best option. The ideal would be combine these two options into one solution. The most important thing in the design chosen for modeling the agent behavior is whether or not it can impose a limitation for representing the behavior. Therefore, we believe that a combination of these strategies is the best option.

Also, we note that, many tools offer the user various strategies for modeling the agent behaviors that try to use natural language to make the process easier. Even so, the user must have some notion of logic and structured programming.

Even, though we cited only four approaches, other strategies can be found in the literature, such as visual metaphors [83] or statecharts [114] to graphically represent the model.

3.7.3 Design of well-known tools

This section focuses on the design of the architecture of ABM tools: D-Mason, FLAME, Pandora and Repast HPC. The aim is to present how these tools design solutions for parallel and distributed ABM. We give a general overview about these designs. More details on the implementation of the main components of each of these tools are presented later in Section 5.4.

D-Mason

D-Mason is a parallel version of the Mason library for writing and running simulations of agent-based simulation models [42]. Therefore, the discussions on the D-Mason architecture go by the Mason library. Figure 3.5 shows how D-mason fits in Mason components. Mason implements the Model-View-Controller (MVC) software architectural pattern. MVC separates an application into three main components: model, view, and controller. Model is responsible for manipulating the data. View provides a visualization of the current state of the model. Controller manages all the processes of the application updating the state of the model and the view. MVC is an implementation of Separation of Concerns, and it is very useful for applications that have different frontends for the same backend, because the communication among the MVC parts is done by a well-defined interface.

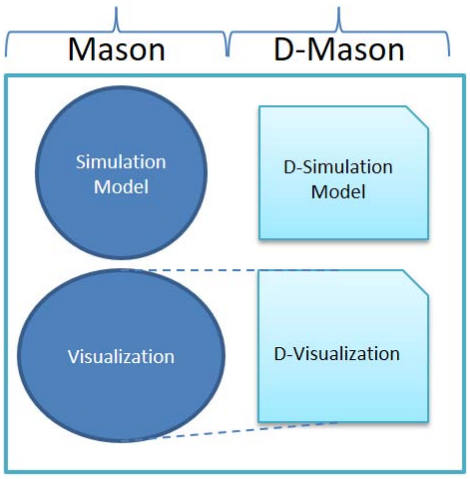


Figure 3.5: Mason and D-Mason components.

D-Mason supports the Master-Work paradigm, where the Master assigns a portion of the whole computation to each Worker. Also, D-Mason adds a new layer named D-simulation [42] that implements algorithms related to parallel and distributed simulation. The advantages in flexibility, maintenance and sustainability are clear to the D-Mason team when the architecture is structured in layers [42]. In accordance with [42], the D-Mason model is used on the upper layer of the simulation framework, with the purpose of hiding the details of the architecture as much as possible. D-Mason has well-defined Java interfaces that enable the user to implement their model in a parallel and distributed way using its built-in solutions. D-Mason was designed to have backward compatibility with existing Mason applications.

FLAME

Two key components of FLAME (Figure 3.6) are: xparser and message board library (libmboard). The Xparser component receives as input the agent behavior definitions and the model specification. Then, parallel simulation code is automatically generated for the user. The message board library component controls the communication among all the agents. Xparser creates a dependency graph based on the transition functions defined by the user. The parallelization algorithms use this graph to define the tasks to execute in parallel. Also, the dependency graph has information about the read and write operations in the message board.

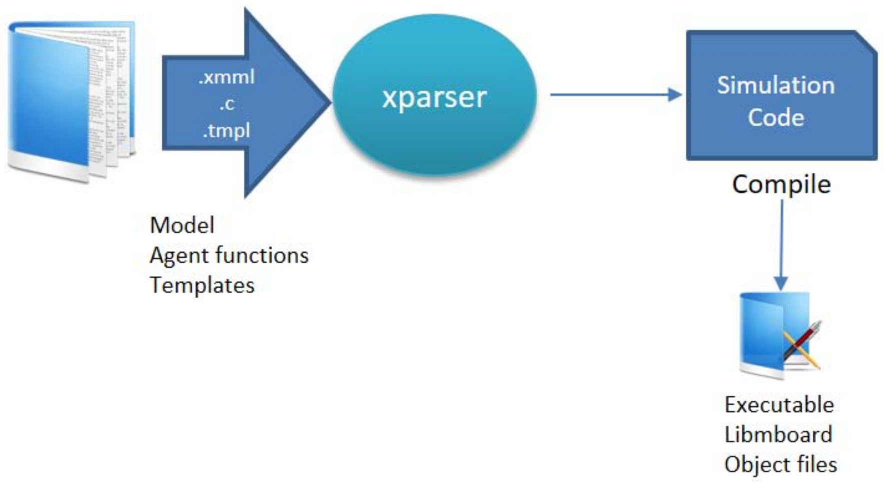


Figure 3.6: Design of FLAME environment.

All parallel and distributed algorithms are implemented in the xparser component. Changing the main algorithms requires an understanding of the kernel of FLAME. Márquez, César and Sorribes [96] propose a load balancing schema for FLAME which is based on the migration of agents. This new feature was implemented using the template engine of FLAME. The template engine uses a set of template files [33] (see Figure 3.6) to generate the simulation code through information from the model specification [96, 97]. Therefore, the authors create additional template files with the migration routines that are automatically generated, too. Small modifications in the source code of xparser component were required in order to accept new template files.

The FLAME documentation and these cited papers do not clarify how many customizations of parallel and distributed algorithms can be done, and how far they can go through the FLAME template engine. A redesign of FLAME is proposed in [34], which presents and discusses the current limitations of FLAME.

Pandora

The abstract classes World and Agent (see Figure 3.7) are the core of Pandora and they form the content of any model [130]. Many parameters of the simulation are defined using an XML. The class World represents the environment, and it is composed of several raster maps that are bi-dimensional matrices of values because Pandora gives support to Geographical Information Systems (GIS). The class Agent encapsulates any entity of the model with internal state, decision making processes and behavior [130]. This class use the Hierarchical Data Format (HDF5) protocol in order to serialize its state.

The key component of Pandora is Scheduler. Scheduler is the process that updates the set of agents and the environment that together form the model. Pandora exposes an interface for HPC experts to create their own scheduler through the combination of the bridge and factory method design patterns [130].

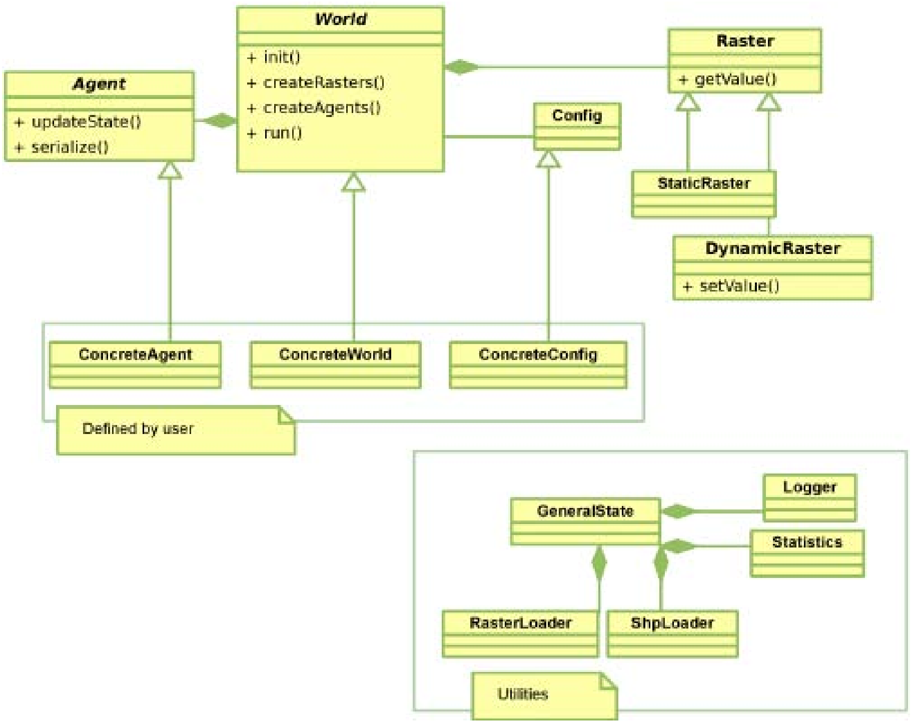


Figure 3.7: Class diagram of Pandora.

An interesting feature of the Pandora design is the flexibility of providing model development in C++ as well as Python. Pandora uses the library boost-python to link the two interfaces and enable the code binding Python calls to C++ classes and functions.

Repast HPC

Repast HPC is a parallel distributed C++ implementation of Repast Simphony for Java [41]. That is why we focus on some features of the Repast Simphony [110]. Repast Simphony has a strict separation between model specification, model execution, data storage, and visualization [110]. Repast Simphony uses a highly modular architecture, where each module is an independent 'plug-in' that can be connected or disconnected with a few lines of XML [110]. A list of several plug-ins of Repast are presented in [110].

Collier and North [41] report the lack of flexibility in previous versions of Repast Simphony. The new versions of Repast Simphony had as their aim the flexibility and reusability of its various components. Repast Simphony solved this problem through the use of contexts and projections. Consequently, the Repast HPC preserves these features and continues to encourage the flexibility and reusability of components and models. Repast HPC models agents as objects, collections of agents as contexts and the environment as projections.

Repast HPC simulations can be written in C++ or using the Repast HPC Logo-like. In accordance with [110], the idea of Repast HPC Logo is to promote ease of use and further hide the complexities of implementing a parallel simulation.

3.7.4 ABMS development recommendations

Railsback, Lytinen and Jackson [123] reviewed five software platforms for scientific ABM by implementing example models in each. More than this review, they give some development recommendations for these tools. Recommendations include: (1) completing the documentation; (2) strengthening conceptual frameworks; (3) providing better tools for statistical output and automating simulation experiments; (4) simplifying common tasks; and, (5) researching technologies for understanding how simulation results arise [123].

Documentation of the tool is fundamental for the tool to be successful in terms of its wide use by the scientific community. The documentation must be built while the code is developed. Several documentation generators are freely available. Besides Application Program Interface (API) documentation, the documentation must include several examples and tutorials on how to use it.

A conceptual framework in an ABM context is a set of standard concepts for designing and describing ABMs [123]. Lets take Swarm [99], that has well-defined conceptual frameworks, for example, swarms, collections, actions, schedules, and observers. The idea is to enable the user to think about and describe their model using these concepts. Afterwards, they can intuitively translate their model to the tool. Another tool with well-defined conceptual frameworks is Netlogo [163].

The output of the simulation must be supported by statistically trustworthy results. It would be interesting for the ABM tools to automate simulation experiments in order to define the number of simulations required, the type of statistical analysis, variable of analysis, and other statistical topics. Also, the automated execution of multiple scenarios and replicates [123] is important because this is useful for getting reliable results, what if experimentation, calibration and validation processes of the model. Netlogo [163] has a tool called BehaviorSpace, that allows for the execution of multiple scenarios and replicates.

One way to improve the trade-off between ease of use and generality of the tool is by making common tasks easier to program [123]. Two techniques for that are suggested by [123]: using high-level code that hides some functions; and better use of standardized design patterns. High-level code hidden details are not important for this type of user. As an example, how to choose the next agent to execute? For this type of user, the important thing is to know if the tool supports this random access to the agent and what the user has to do to accomplish that. Design pattern is the way to ensure that a tool can be flexible enough to future extensions and can represent several user models.

The last point is understanding how simulation results arise. The tools must allow us to check whether the results are useful or only the consequence of mistakes [123]. A discussion on the difficulty and frustration of this problem is presented in [129]. Therefore, the tool must provide probes and debuggers for observing a few agents or a small part of the software [123]. This problem becomes bigger when the simulation is parallel and distributed. Therefore, the verification process must be accompanied by a methodology.

Finally, Helbing and Balietti [66] present a deep discussion on how to implement simulations of ABM that offers important points that should be used as a check list.

3.7.5 Conclusion

After analyzing the design of these tools, we observe that all of them were designed to execute ABMS simulations using their built-in HPC approaches. They were projected to execute ABMS using HPC. The execution of models is limited to approaches available in the tools. We found that these tools were not designed to execute experimentations of HPC for ABM that require parallel and distributed solutions. The gain of new knowledge and the analysis of the hypothesis is in the application area to which the ABM belongs, such as biology, chemical, sociology, etc, not to the HPC field applied to ABMS solutions.

Also, we observed that a huge programming effort is required for add new approaches, techniques or strategies HPC in these tools, as an examples [96, 97]. In addition, these ABMS tools have support for application area researchers concerning to develop models. All these tools offer facilities for the user to develop their models, but decreasing the required HPC know-how is still a challenge.

Agent-based parallel and distributed simulation tools must support modeling and simulation research as well as HPC research. Because, we understand that the characteristics of the ABM and HPC techniques must fit together in order to extract the best solution performance. Therefore, it is important distinguish the users and provide appropriate views for them. For example, application area researchers can create ABMS and choose the best HPC features that fits its model in a transparent way; and HPC experts can implement different parallelization algorithms, techniques and solutions. Therefore, we believe that ABMS tools must be designed taking into account the two different user profiles. It can enable application area researchers to gain knowledge about the system under study using ABMs that require high performance computing solution; and enable to HPC expert users to develop techniques of high performance parallel and distributed simulation for ABM problems without high programming effort. However this is only possible if the tool is designed with these purpose and must follow good design and programming practices. The design has to deal with a trade-off: complexity for the user and performance of the solution. Decrease the complexity of user generally means use default values for specific situation where these values might do not be the best option. On the other hand, enable the user to specify these values can bring to user a high complexity. In Section 5, we present how we design our methodology that meet with support the two type of profiles mentioned before.

Chapter 4

ABM and HPC approaches

4.1 Introduction

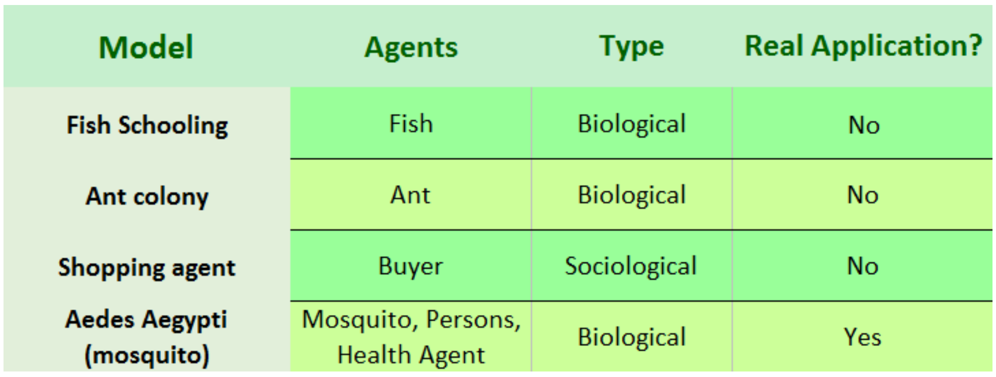
In this chapter, we present four agent-based models as case studies that were used throughout the methodology development. Also, we introduce the high performance computer approaches proposed to solve problems due to ABM execution in HPC environments. First of all, we present all these ABM in detail. Then, we present the HPC approaches focused on how to solve the problems mentioned.

4.2 ABMs case study

The agent-based models used as case studies are: Fish Schooling, Ant Colony, Shopping Agent, and Assessment of *Aedes Aegypti* pupal productivity models. Figure 4.1 presents some information about the models used as case studies. The "Agents" column indicates the agents modeled. The type of model is specified in the "Type" column. The "Real application" column indicates if the model is based on real problem. Fish Schooling, Ant Colony and Shopping Agent are synthetics models. On the other hand, the model Assessment of *Aedes Aegypti* pupal productivity has a social impact. Therefore, we will give more details about this model than the others due its importance and relevance.

4.2.1 Fish Schooling

Fish Schooling is a behavior in which a group of fish are swimming in harmonious patterns. It is composed of many fish of the same species and can be observed in almost 80 percent of the more than 20,000 known fish species in some phase of their life cycle. Fish schooling is one of the most common social groups in nature. This is an example of a self-organized



Model	Agents	Type	Real Application?
Fish Schooling	Fish	Biological	No
Ant colony	Ant	Biological	No
Shopping agent	Buyer	Sociological	No
Aedes Aegypti (mosquito)	Mosquito, Persons, Health Agent	Biological	Yes

Figure 4.1: Models used as case study.

system where we find a leaderless, non-hierarchical and decentralized structure. This social aggregation shows complex emergent properties such as strong cohesion and a high level of synchronization. A collective behavior emerges as a result of local interactions between fish that are within a limited vision range.

We have implemented the fish schooling biological model described in [71] and [72]. This model takes into account that a new fish's position and orientation depends on the position and orientation of a fixed number of nearest-neighbors. The model identifies three vision areas: attraction, repulsion and parallel orientation (see Figure 4.2).

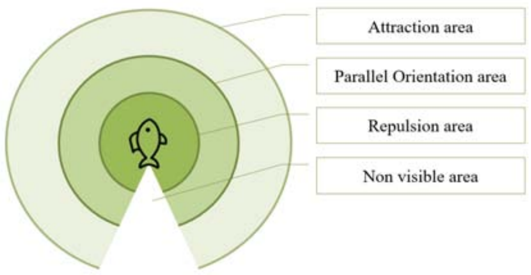


Figure 4.2: Fish's vision areas.

Depending on the position of its neighbors, the fish chooses one of these behaviors. The repulsion behavior avoids a collision between the fish. In the parallel orientation behavior, the group moves in the same direction. Attraction behavior maintains the cohesion of the group. More information and details on this model can be found in these references: [17, 16, 143, 146, 145, 144].

4.2.2 Ant colony

The ant colony model is based on [162]. In this model, the ants' main rules are: finding food and bringing the food to the ant nest. In the search for food, the ant drops a chemical when it finds some and moves back to the nest carrying the food. When other ants sense the chemical, they follow it to the food. The chemical trail intensifies as more ants follow this trail, allowing other ants to find the piece of food (see Figure 4.3).

We verified that our parallel computational model follows the functional features described by [162]: it explores the food source in order, beginning with the amount of food closest to the nest and ending with the amount of food farthest away from the nest. This behavior emerges in ant colonies because it is easier to create a stable trail to the nearest food than to food that is farther away. The chemical only lasts a short time before it evaporates; consequently, the trail must be strengthened by other ants. Therefore, more distant food requires a larger number of ants to form a stable pheromone trail.

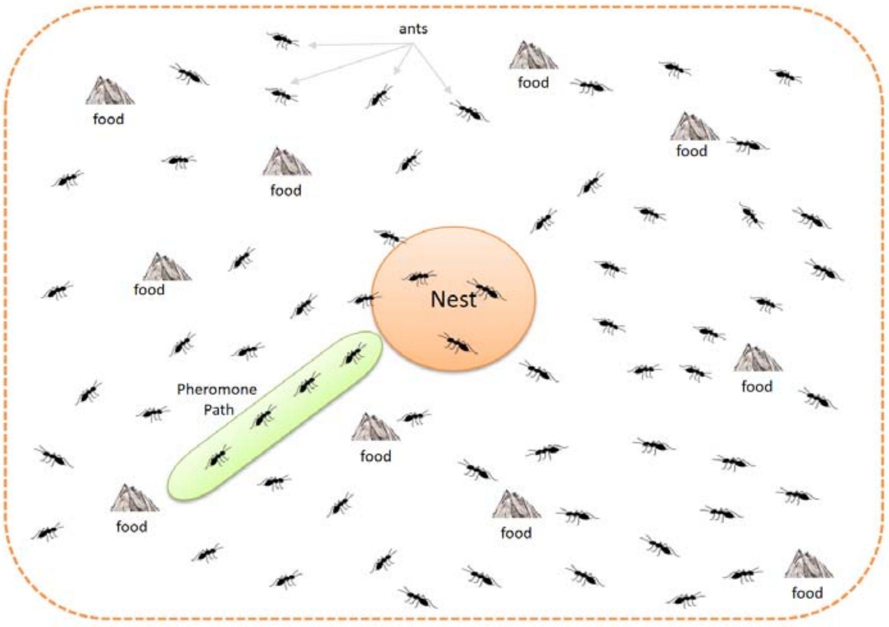


Figure 4.3: Ant colony.

4.2.3 Shopping agent

We are using and adapting the Shopping Agent model described by [61]. In this model, each agent has a shopping list. The products must be bought in the stores which are distributed throughout an environment. Each shop sells just one type of product. The goal of the agent is to purchase all of the products on its list.

Gilbert and Troitzsch [61] have developed three versions of the model: 1); the first version is the simplest version (see Figure 4.4). Agents randomly walk around the environment in order to find a shop which sells some product from their list; 2) in the second version, Figure 4.5, the authors add another behavior: the agents can see neighboring shops that are inside their covered radius; and 3) in the third version, Figure 4.6, we find the smartest agents. In this version, beyond the behaviors of Versions 1 and 2, the agents exchange information about the known shop positions.

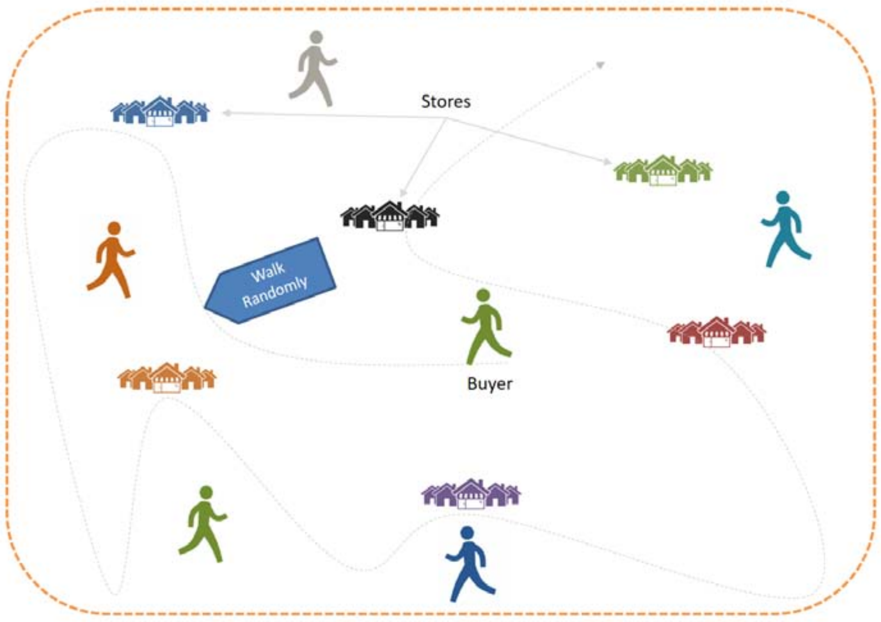


Figure 4.4: Shopping Agent model version 1.

The Gilbert and Troitzsch [61] model was implemented in Netlogo [163]. The Netlogo implementation is supported by cellular automata theory. Therefore, some functional features such as *seeing neighboring shops* are implemented using the Moore Neighborhood concept. This concept was replaced by covering radius where the agents interact with objects of the environment or agents that are inside this radius.

4.2.4 Assessment of *Aedes Aegypti* pupal productivity

The ABM for assessing the *Aedes Aegypti* Pupal Productivity takes into account the capacity of pupae production for each container where the mosquitoes lay eggs. The idea is to offer health agents a model that allows for the simulation of different spatially distributed container profiles and consequently help in the fight against and the surveillance of *Aedes aegypti*. As a result, health agents could simulate the reduction of the vector population density by

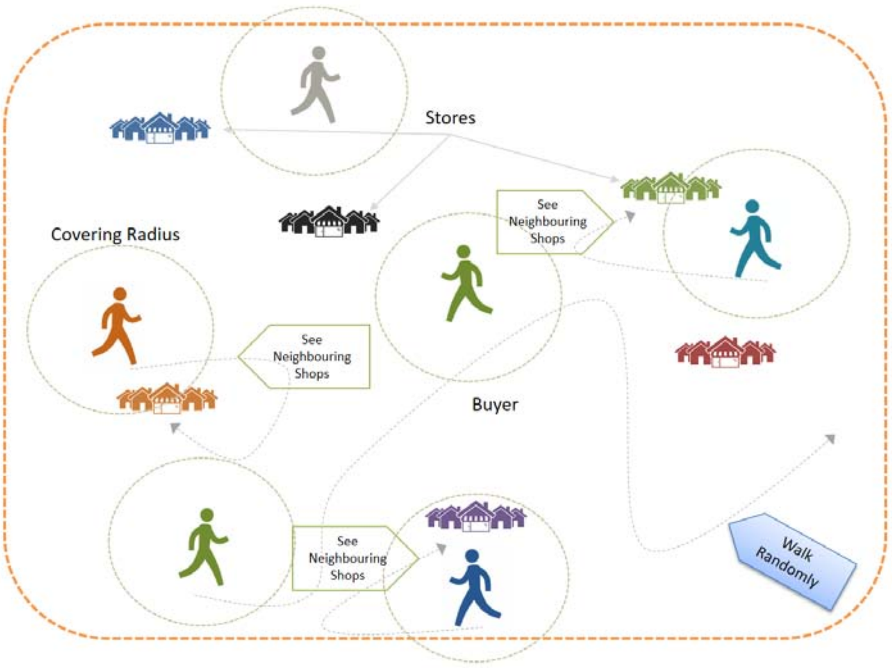


Figure 4.5: Shopping Agent model version 2.

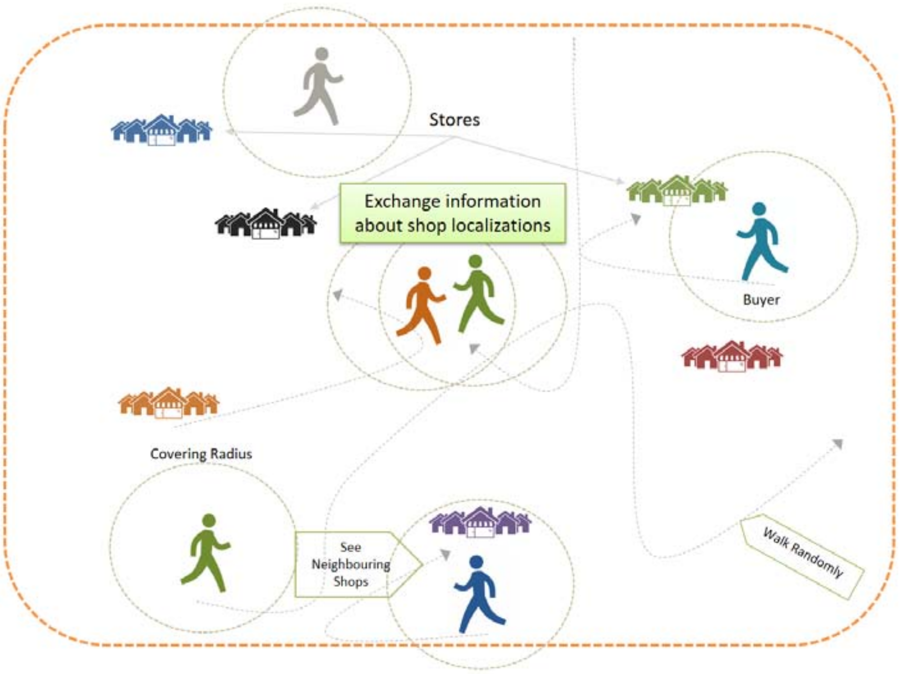


Figure 4.6: Shopping Agent model version 3.

eliminating or modifying containers that work as mosquito breeding sites following the [167] orientations. In addition, the proposed model could be used by experts in order to support the decisions made and make health action recommendations. As application examples, we can cite that the health agents can guide the control activities where containers with higher productivity were found. Currently, the *Aedes aegypti* programs of surveillance and prevention give the same importance to all type of breeders [23]. Moreover, the health agents could develop orientation of educational messages to people in order to eliminate specific types of containers, consequently, reducing the total costs and improving the surveillance and the control of the mosquito.

Several control actions of *Aedes Aegypti* have been proposed by [167]. One of them is to identify the mosquito population through sampling methods. The traditional method divides the area, considering similar socio-environmental characteristics, in order to have data homogeneity. These homogeneous areas are called Strata. Each Stratum is composed of houses, and each house might have a breeding site of *Aedes Aegypti*. Figure 4.7 shows the iterations among the environment, mosquitoes and the people that produce the transmission of disease (Dengue, Chikungunya or Zika). The houses can have different quantities of containers, and each container can have a different productivity level (see Figure 4.7, Steps 1 and 2). Thus, the estimation of the mosquito population can be completely wrong if the containers are considered equals. In this ecosystem, the mosquito is the intermediate host, and the human is the definitive host. The mosquito is infected only when it bites a person that is infected (Step 3). Therefore, the mosquito has to be infected to transmit the illness to the person (Step 4). Otherwise, the mosquito continues without transmitting the arbovirus (Step 5). The relationship between the mosquito and the person is modelled and detailed in the flowchart presented in Figure 4.8. In nature, it is not common for the male to bite people. Therefore, only the female mosquitoes bite, in order to lay eggs, and the female always lays eggs in different containers (Step 6). This behavior increases the chances of survival for the eggs and has a strong influence on the dissemination of these arbovirus. The preventive actions of the public health agent can be more efficient in reducing the mosquito population if they know which containers of a specific area are more epidemiologically relevant (Step 7). Additional details about the agents and the environment are explained in the next subsections.

The Agents

The proposed model has three agents: Mosquito, Person, and Public Health Agent. Each agent has attributes and behaviors defined in the subsections as follows. For the sake of simplicity, we will not explain all of the behaviors in detail, but will instead focus on the most important ones.

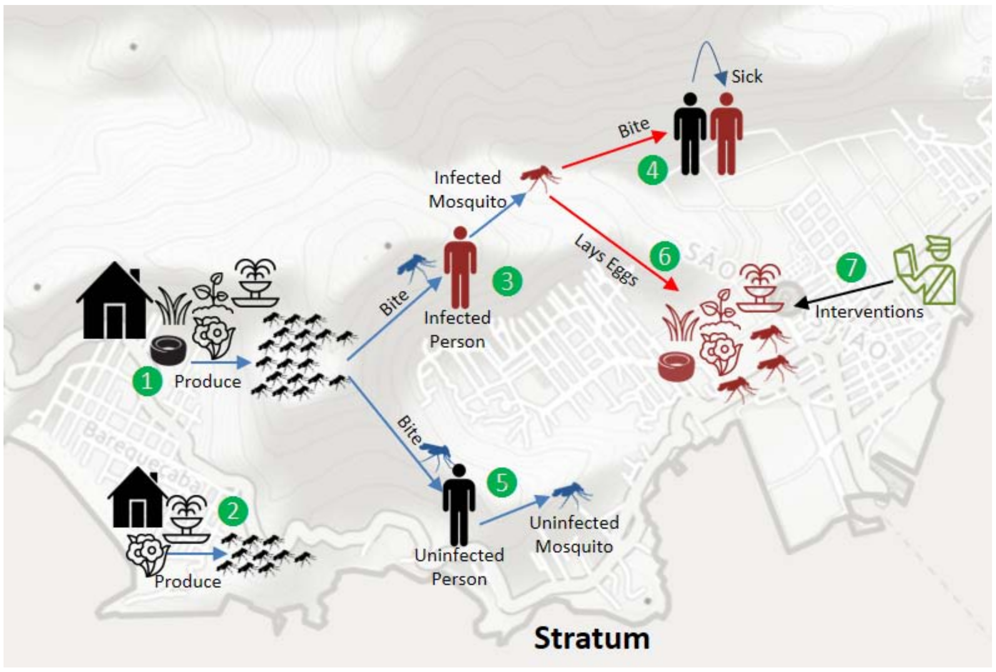


Figure 4.7: Ecosystem among the environment, mosquitoes and the people. Its iterations produce the transmission of the arbovirus (Dengue, Chikungunya or Zika).

Mosquito

Attributes:

- Lifespan: indicates the age of the mosquito (6-8 weeks).
- Infected: indicates if the mosquito is infected with the arbovirus.
- Extrinsic incubation: the period (in days) that the arbovirus takes to complete its development in the *Aedes Aegypti*. During this period, the mosquito is not able to infect the people. Even if the mosquito has the virus, it cannot transmit it to people during this period.
- Transmit: indicates if the mosquito can transmit the arbovirus to a person. If the mosquito has the virus and the extrinsic incubation period has finished, then the mosquito is able to transmit the virus a person.

Behaviors:

- The mosquito looks for people in order to obtain blood for egg production. The transmission of the arbovirus occurs because the mosquitoes require blood to grow their eggs. Therefore, the transmission of the virus might occur when a mosquito bites a person.

- The mosquito bites a person.
- The mosquito lays eggs in the containers. Each female lays 87 per batch, on average. The females can produce up to five batches of eggs in a lifetime. In addition, the female tries to lay eggs in at least three different containers. The initial number of eggs laid per batch is equal for all containers. However, container productivity will vary, depending on where the eggs were laid. This behavior is detailed in the Figure 4.9.
- Mosquitoes fly within 100 meters of where they emerge [167]. This radius of flight of the mosquito is important, because it defines its area of actuation. This means that the mosquito bites people and reproduces within this area.

Person

Attributes:

- Infected: indicates if the agent is infected with the arbovirus.
- Intrinsic incubation: the period (in days) that the arbovirus takes to complete its development in the person. In this period, the person is not able to infect a mosquito. Even if the person has the virus, it cannot transmit it to a mosquito during this period.
- Transmit: indicates if the agent can transmit the arbovirus to mosquito. If the person has the virus and the intrinsic incubation period has finished, then the person is able to transmit the virus to a mosquito.
- Knowing: indicates how much a person knows about the prevention of these diseases. This attribute can be used in order to simulate the effects of education on people.

Behaviors:

- People walk randomly and live in the Stratum.
- People learn about *Aedes Aegypti*.

Public Health Agent

The public health agent carries out preventive actions against *Aedes Aegypti*. Their actions may be oriented according to container profiles. The agent can give an orientation of educational messages to people.

Behaviors:

- The public health agent walks in the Stratum.
- The public health agent makes interventions in containers.
- The public health agent eliminates containers.
- The public health agent gives an orientation of prevention for people.

Environment

In ABM, the environment is the place where the interactions between the agent-agent and agent-environment occur. Basically, there are two objects which have to be modelled: the Stratum and the container.

Stratum

Attributes:

- Has a collection of agents: mosquito, person and public health agent.
- Has a collection of containers.
- House (premise) index (HI). This index defines the percentage of houses with *Aedes aegypti* breeding sites.
- Breteau index (BI). This index defines the number of buildings for each 100 buildings researched where positive breeding sites were found. Positive breeding sites have containers with larvae of *Aedes aegypti*.
- Index by type of container (ITC). This index defines the relationship between the number of types of positive containers and the total number of positive containers researched.
- Has a collection of buildings.

Container

Attributes:

- Percentage of productivity indicates how many adult mosquitoes this site can produce.
- Positive indicates if the containers have larvae of *Aedes aegypti*.
- Epidemiological relevance indicates how epidemiologically relevant the container is.

Flowchart of the main behaviors

The main behaviors of this model are presented in the following flowcharts. Figure 4.8 shows the interaction between mosquitoes and people.

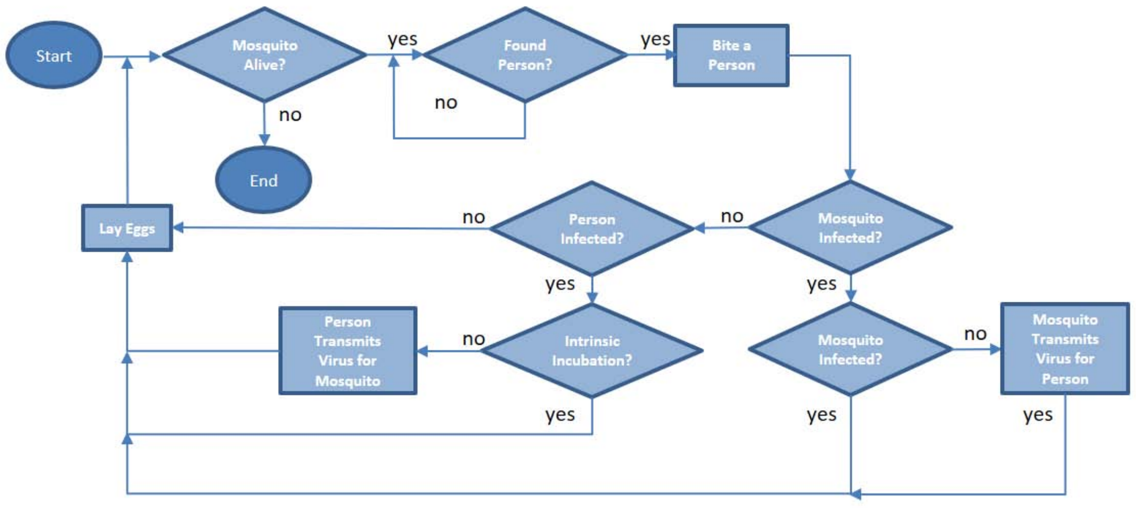


Figure 4.8: This flowchart represents the interaction between mosquitoes and people.

Figure 4.9 shows the laying eggs behavior. The mosquito has to lay its eggs when it bites a person. Generally, each female has five batches in a lifetime. The female has to find a container, then lay a specific portion of the eggs of that batch. The model maintains life only of a determined number of eggs, according to the productivity of the container where the mosquito laid the eggs.

This model has all the main variables parameterized; therefore, it is able to represent any of the three arbovirus with a simple adjustment of these parameters, such as extrinsic and intrinsic incubation period.

4.3 HPC approaches

Realistic ABMS are generally composed of thousands of agents and require reliable results supported by statistical studies. To have results in a viable time, the solution should be executed in HPC environments. But we observe that the HPC approaches must fit specific characteristics of the model in order to obtain better performance solutions. In this section, we present the HPC approaches proposed in our methodology that can deal with some problems associated with the ABMs presented previously.

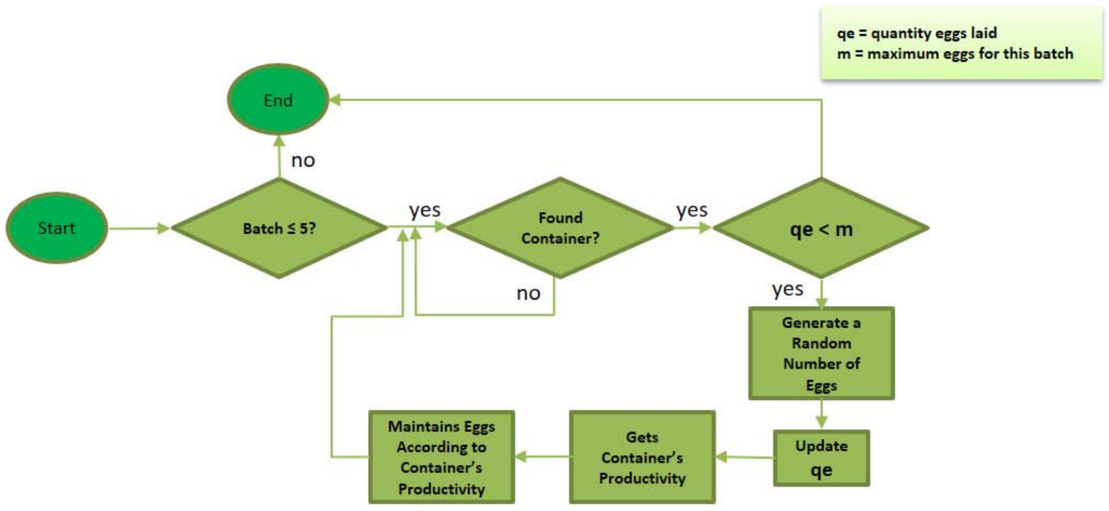

qe = quantity eggs laid
m = maximum eggs for this batch
Start
Batch ≤ 5?
no
End
yes
Found Container?
no
yes
qe < m
yes
Generate a Random Number of Eggs
Update qe
Gets Container's Productivity
Maintains Eggs According to Container's Productivity

Figure 4.9: This flowchart represents the mosquito agent behavior: laying eggs.

4.3.1 Communication patterns

Figure 4.10 shows the communication pattern of the Fish Schooling model throughout the execution. The green area represents the time spent in computing. The red area represents the time spent in communication.

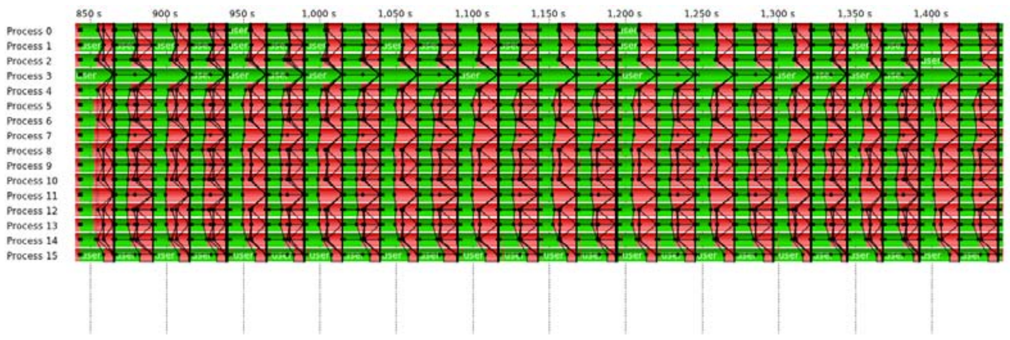


Figure 4.10: Graphic of processes communication shows the behavior of Fish schooling model throughout the execution.

Communication is a problem in Fish Schooling model because it is a communication-consumer parallel application. Therefore, we have to decrease the communication time between processors to get more efficient results. In this way we should understand and analyze communication in time-driven simulations. We use a time-driven mechanism because this agent-based model describes the motion of a agents in discrete time steps. Furthermore, we implement a conservative time-management protocol because each processes requires exchanging information from adjacent processes before starting the next simulation step. Time-driven simulations tend to have synchronous behavior and as can be see in Figure 4.10. Then, which is the best communication strategy in time-driven simulation? Asynchronous, synchronous or both? These are the questions that lead this approach.

In order to answer these questions, we propose three communication strategies: asynchronous and synchronous message passing and bulk-synchronous parallel. Previous works [144–146] that uses Fish Schooling model uses an asynchronous communication strategy. We propose to add the synchronous communication strategy and bulk-synchronous parallel. We decided to propose these two strategies because we have noticed that time-driven simulations tend to have synchronous behavior. By reviewing the literature [141] [68] we have verified that the BSP computing model can be an alternative to improve performance of parallel applications. Fish Schooling model is a communication-consumer application, so we can analyze how the communication using asynchronous, synchronous and BSP can bring some benefit and how this communication strategies influences on ABMs performance (e.g. scalability, efficiency, etc).

4.3.2 Hybrid cluster-based partitioning

The four ABM case studies were developed for distributed memory. There are several broadcast communications in the Fish Schooling model because the processes must exchange information in order to keep the coherence of simulation. Consequently, there is a lack of efficiency when large number of cores are used and these cores are distributed in many nodes. This problem can also be found in other cluster-based models; therefore, this is the main motivation to propose a hybrid version of cluster-based partitioning (see 3.4.1). Also, the following two key features support the hybrid version proposed:

- Each logical process needs to know all the positions of the centroids for each simulation step. Agents are constantly in motion, therefore the centroid changes according to their position. In addition, the agents do the cluster migration process during all simulation. The migration process occurs when the agent change their location from one MPI process to other. In order to solve these problems, the pure MPI cluster-based partitioning was designed and developed by using MPI broadcast communication. Currently in the MPI version, each core receives one MPI process. Thus, the quantity of MPI broadcast messages increases when the number of cores increases.

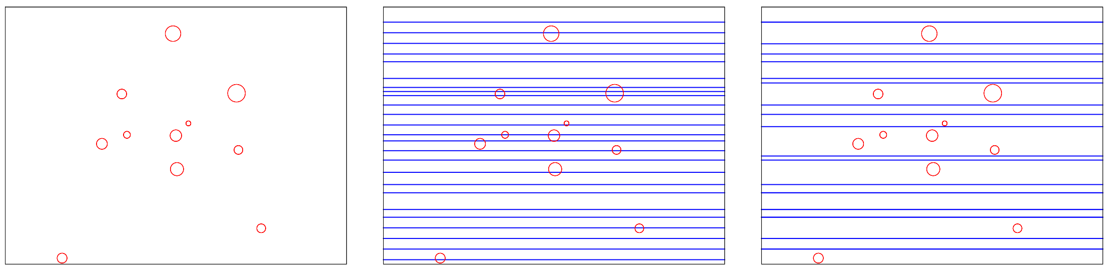
- The partitioning algorithm uses the meta-cluster to store consecutively the clusters in the same process. As agent behavior is influenced by their neighbors the operations of computation and communication occur between the agents that are in same meta-cluster. This way, it is possible to explore the fine-grain parallelism through OpenMP threads.

Agents are inserted, deleted and updated on the clusters throughout simulation. Therefore the same cluster and agent may be computed at same moment. Keeping the coherence and the code free of memory access violation resulted in an undesirable slowdown by using critical section. After preliminary experiments, we observed that changes in the list of clusters data structure were necessary. Therefore we propose a new data structure to deal with the simultaneous inserting, deleting and updating problems.

4.3.3 Strip partitioning

Strip partitioning proposed aims to decrease the communication volume that divides the environment among the processes, avoiding shared objects in different processes. This situation is common in models where the specific resources are consumed by agents. One example is in the Ant Colony model where the agents eat the foods. Shared objects mean that messages must be sent and received by processes in order to keep the objects' state updated. This problem grows when the environment is dynamic, and communication volume increases as a consequence. Therefore, we have developed a strip partitioning algorithm where each object of the environment is restrained in just one partition (Figure 4.11(c)). The algorithm prevents objects from being located in two partitions, as exemplified in Figure 4.11(b). The Euclidean Space is first divided into N strips of approximately the same size, where N is the number of cores. Following this, each strip is divided again until the partition limit restriction heuristic is reached. We define this heuristic as the size that the partitions can take on. It has three proposals: a) to avoid shared objects between partitions. An object cannot be shared between strips. This restriction avoids an increase in communication between strips. These communications would be necessary in order to keep the object state updated; b) to avoid excessive communication between neighboring strips. If the strip is too small, then the agents will exceed the limits of the strip with greater constancy; and c) to avoid a migration process to non-consecutive partitions when the partition are too small. The migration process is the movement of agents of one partition to another. In this way, the communication only occurs between neighboring strips.

Figure 4.12 represents the partitioning scheme proposed. The strips may have two directions: horizontal (H) or vertical (V). Each partition can be created in a fixed (FI) or a variable (VA) way. FI partitioning creates N partitions where N is equal to the number of cores. For the fixed partitioning, just one mapping approach can be applied. One partition is



(a) Ant colony environment (b) Environment partitioned with sharing objects (c) Environment partitioned with no sharing objects

Figure 4.11: Object distributions in the environment.

allocated to one core (M0). We can apply two heuristics for a variable partitioning creation, because the number of partitions is higher than the number of cores: in the first one, the partitions are created until the creation of other partitions is no longer possible. We call that heuristic partition last restrictions (PLR). These restrictions are mentioned above. In the second heuristic, the partitioning algorithm takes into account the cost of computing (PMC). Firstly, this heuristic executes the PLR heuristic, then it calculates the computing cost for each strip. The partitioning algorithm finishes when it finds the smallest standard deviation between the strips. The heuristics PLR and PMC might be used with mapping M1 and M2. M1 mapping divides the total number of strips among the cores. M2 considers the cost of each strip.

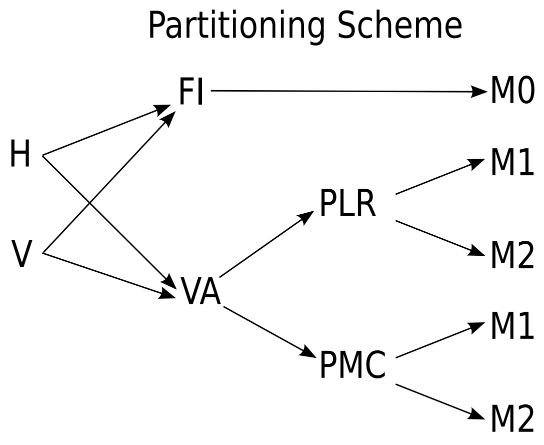


Figure 4.12: Partitioning scheme

We define the partitioning problem as follows. Our problem is partitioning a Euclidean Space (ES) in a set of cores. ES has a fixed size limit $[x,-x]$ and $[y,-y]$. An ES may be divided into many partitions. A partition is defined as a set of strips: $s = (x_1, x_2, y_1, y_2)$ where $(x_1, x_2) \subset [x,-x]$ and $(y_1, y_2) \subset [y,-y]$. The ES is composed of objects which cannot be shared between strips. Each object o has a position (x,y) and size, and $\forall o \in s_x$, if $o_x \in s_x$ then $o_x \ni (S - s_x)$, where S is the total number of strips.

The execution load on a core (c) is the sum of all partitions allocated in this core. The total computing cost of the partition strategy is the sum of the computing cost of each of the strips. We consider that, if the area of strip s_1 is bigger than the area of strip s_2, then the migration cost of strip s_1 is less than strip s_2. All objects have a computing cost. The quantity of objects in the strip has an impact on computing and communication between strips. Therefore, our objective function tries to find the computing cost difference between the strips. This function uses as terms: A) total area of the strip; B) computing cost of the strip objects; C) computing cost objects at neighboring strips; and D) quantity of neighboring strips.

$$F(S) = \sigma(cost\_strip_{s=1}^{S}(A_s + B_s + C_s + D_s)) \tag{4.1}$$

This function (4.1) calculates the cost of each strip (s), then returns the standard deviation(σ), considering all strips (S). The lower the σ, the better the result, because it means that the difference in the cost of computing in the strips is lower. Term A specifies the size cost of each strip. It is defined by Equation 4.2.

$$A_s = strip\_size\_cost_s = \frac{strip\_area_s}{total\_area} \tag{4.2}$$

The quantity of agents is proportional to the strip area; therefore, the computing cost of the agents increases as the size of the area also increases, because the chances of the strip receiving more agents are higher.

$$B_s = \sum_{j:o_j \in s_i}^{n} comp\_cost\_obj_o / \sum_{o=1}^{n} comp\_cost\_obj_o \tag{4.3}$$

In the ant algorithm, the agents try to find the objects; therefore, the strips which have more objects tend to receive more agents and have more migratory processes. A strip can have zero or many objects, and each object has a computing cost. We calculate the proportional cost of objects for each strip with Equation 4.3, where o is an object and n is the total number of objects in the environment.

$$C_s = \frac{\sum_{j:o_j \in (s_{i-1}, s_{i+1})}^{n} comp\_cost\_obj_o}{total\_comp\_cost\_obj} * strip\_size\_cost_i * number\_obj(s_i - 1, s_i + 1) \quad (4.4)$$

The strip is subject to more migratory processes if its neighbors contain objects, because the algorithm converges in this direction. Therefore, we consider the size of the strip ($strip\_size\_cost_i$) and the number of objects in the neighboring strips ($number\_obj(s_i - 1, s_i + 1)$). This implicates an agent's migration to another strip. The term defined by 4.6 represents the influence of the strip on the total volume of migration.

$$number\_neighbors\_strip = \begin{cases} 1 & \text{if } (s_i = 0 \text{ or } s_i = S) \\ 2 & \text{if } (s_i <> 0 \text{ or } s_i <> S) \end{cases} \quad (4.5)$$

$$D_s = number\_neighboring\_strips/(S-2)*2+2 \quad (4.6)$$

Strip partitioning returns a naive partitioning, where the load balancing focuses on area and not on the load[134]. Therefore, the load balancing must be covered in the mapping process. We use three mapping approaches: a) one strip per core; b) many strips per core with distribution by division; and c) many strips per core with distribution by computing evaluation. In the first approach, the number of strips is the same as the number of cores. Therefore, each strip is allocated to one core. When the partitioning algorithm generates many partitions, we can distribute the strips by doing a simple division: the number of strips by the number of cores. Neither of these mapping approaches (a or b) are effective, because they do not take into account the computing requirement of the strips. Therefore, some type of load imbalance can be generated. However, the last approach (c) distributes the strips considering the workload of each core. The partitioning algorithm saves the load computing of each partition when the objective function is calculated. Then, we can use the computing load of partitions in the mapping process. This approach tries to set the equivalent computing load for each core.

4.3.4 Hybrid Strip partitioning

The partitioning of data is the distribution of space simulated throughout the distributed architecture. The space simulated in this context is composed of agents and an environment. These techniques try to reduce the communication time among the distributed processes and evenly distributed the agents throughout the cores. The strip partitioning avoids the objects of the environment being shared between processes. Consequently, it avoids extra communication in order to update the state of the objects that are changed because of the interactions with the agents. But, we observed, using Ant Colony model, a concentration of ants near nests and food. This is a natural and correct behavior of the ants in this model. The problem is that some cores become idle when this behavior occurs. Therefore, we propose a hybrid strip partitioning with the aim of decreasing the idleness of cores. We decrease the idleness of these cores through the creation of OpenMP threads which are used to compute the extra agents that are in other cores.

The hybrid strip partitioning is implemented with OpenMPI and OpenMP. This partitioning checks the proportion of the quantity of agents inside a strip and dynamically creates a number of threads. First, we calculate an initial number of threads using Formula 4.7, where the *number_agent* is the total number of agents inside a partition; the $total\_number\_agents$ is the total number of agents simulated; and *number_cores* is the total number of MPI processes divided by 4. We use four because it fits better with the CPU architecture; however, this value is parameterized. Finally, the number of threads is defined when we apply the rules in accordance with Equation 4.8.

$$threads = number\_agent/total\_number\_agents * number\_cores \tag{4.7}$$

$$threads = \begin{cases} 2 & \text{if } threads = 1 \\ threads & \text{if } threads >= 2 \text{ and } threads <= 8 \\ 8 & \text{if } threads > 8 \end{cases} \tag{4.8}$$

We calculated the number of threads required using Equations 4.7 and 4.8. If the number of threads is higher than one, it indicates that there is a concentration of agents in the strip. The higher is the number of threads calculated, the higher the concentration of agents. We never create more than eight, threads, because doing so can cause a saturation, resource contention and excessive locks.

4.3.5 Optimal run length for simulations

Simulation require reliable results supported by statistical studies, therefore we applied and adapted this statistical approach [37] in order to define the optimal simulation length. We avoid the simulation runs steps of simulations beyond what is needed. This method is divided in three parts: 1) the first one is to identify when the steady state starts; 2) the second one is to identify the run length; 3) and the last one is the replication analysis.

Method to Identify the Steady State

This step is composed of two phases: transient state and steady state. The best way to analyze these states is plotting an output variable of interest. Identifying this phase is important because these steps (warm-up) does not should be considered in output analysis. In accordance with Chung [37], we should use the linear regression approach in order to identify the end of transient state. This approach uses the least-squares method to determine if the linear regression slope coefficient is equal to zero for a specific range of observations. The range of observations must be advances if the slope is not zero or insignificant. The transient state finish when the slope is equal to zero. Therefore the null hypothesis is that the slope of the data is zero. The null hypothesis is rejected if the *P-value* is less than 0.05 because this α level is statistically significant.

Method to Identify the Run Length

The next step is to identify the autocorrelation in order to estimate the variance. It avoids that statistical study be affected negatively. We will use the batch method to finding the autocorrelation. The batch method consists of the following three steps [37]: 1) identify the non significant correlation lag size; 2) make a batch ten times the size of the lag; 3) and make the steady state replication run length ten batches long.

The non significant correlation lag size represents the interval between observations that have little correlation to each other. In order to define the lag size, a number of initial observation sizes must be specified.

The next step is to figure out the correlation, which is obtained by batch ten times the size of the lag. First of all, we have to calculate the batch size by using the equation 4.9. Next, we have to obtain the time for a single observation, shown in equation 4.10. Afterwards, we calculate the transient length time by equation 4.11. And finally we have to calculate the execution time of batch in equation 4.12.

$$batch\_size = 10 * nonsignificant\_lag\_size \tag{4.9}$$

$$single\_observation\_time = \frac{total\_time\_execution}{number\_of\_observations} \tag{4.10}$$

$$transient\_length\_time = single\_observation\_time * transient\_phase \tag{4.11}$$

$$batch\_time = single\_observation\_time * batch\_size \tag{4.12}$$

In order to finish the correlation calculation, we have to calculate the run length by using the equation 4.13.

$$length\_of\_run = transient\_length\_time + (10 * batch\_time) \tag{4.13}$$

After these calculations, the run length is found . We have to discard the transient state and the rest must be divided by the number of batch size. Then, these batch are the individual replications that should be used to output analyses. But before, we have to verify if these replications are statistically significant. Therefore, we will analyze and compare the means by using Analysis of Variance (ANOVA) and Duncan multiple-range test.

Replication Analysis

We have to ensure that the number of replication is enough. Then, we have to know the Standard error (equation 4.14) by using relative precision.

$$standard\_error = t_{1-\alpha/2,n-1} * s/\sqrt{n} \tag{4.14}$$

Where t is the t-distribution for $1-\alpha/2$ and $n-1$ degrees of freedom; s represent the standard deviation of the replication means; and n is number of replications, in our case 10. Relative precision give us a extensible model because we can use relative mean value. We will use 0.10 as relative precision (equation 4.15). Therefore our standard error can be just 10% of mean for our data.

$$relative\_precision = \frac{standard\_error}{\bar{\bar{x}}} \tag{4.15}$$

Where $\bar{\bar{x}}$ is the mean of the replication. The equation 4.16 finds out the required number of replications to specific level of relative precision.

$$number\_replication = \left[\frac{standard\_error}{vrp * \bar{\bar{x}}}\right]^{1/2} \tag{4.16}$$

Where vrp is the value of relative precision (0.09). If the Relative Precision is less than 0.10 than it means that the number of replications found is sufficient. If the replication number is enough, then we have to determine if the mean are statistically significantly different from the others at a given α level. And therefore we will use ANOVA in order to verify which means are different.

ANOVA requires normal distribution but our replications have binomial distribution. Based on Central Limit Theorem, some author propose generate replications sufficiently large. But it is costly for realistic distributed simulation. To solve this problem we propose to approximate the binomial distribution to normal distribution, using equation 4.17.

$$\frac{variable\_of\_interest - (np)}{\sqrt{np(1-p)}} \tag{4.17}$$

Where n and p come from the binomial distribution. For each $variable\_of\_interest$ of each replications we approximate to normal distribution after we apply the ANOVA. If the P-value is less than 0.05, thus we can conclude that the means are different. We can reject the hypothesis that all the replications have identical means. However, this does not mean that every average differs with every average. Therefore we have to use the Duncan test to identify where the differences are. In the Duncan test, means with the same letter are not significantly different. Consequently the replications with the same letter cannot be used together to do output analysis. Because those replications must be significantly different for representing an overall population.

The adaptation proposed in this contribution can be generalized for models which use any variable of interest and whose probability distribution is binomial. However, other probability distribution approximations to normal distribution can be found in the literature.

Chapter 5

Care High Performance Simulation

5.1 Introduction

In this chapter, we detail Care HPS. First of all, we will present Care HPS history. It is important because it shows Care HPS antecedents and how we arrived to Care HPS. Afterwards, we will explain its architecture. Care HPS was designed thinking about reuse and the possible extension of its functionality. Then, we will present a comparison with other tools. Here, we will focus on how each tool addresses specific issues of ABM and parallel and distributed simulation. Finally, we give some examples of how to use Care HPS as a scientific instrument.

5.2 History

The initial ideas about Care HPS emerged in 2013. However, we cannot disregard all of the know-how developed and acquired by group research before the conception of Care HPS. Almost all of this knowledge has given support to and has been implemented in Care HPS. Therefore, the history of Care HPS is divided into two periods: the first between 2004 and 2013, when many contributions in HPC for ABM were proposed; and the second after 2013 when the idea of the current conception of Care HPS started to emerge and its implementation started.

The contributions of our group research in HPC to ABM from 2004 until 2013 can be summarized in the following way. The first version of the simulator dates from 2004. Firstly, we implemented the first fish schooling ABM version with MPI. The distribution of data throughout the architecture was implemented with a partial partitioning. The synchronization of the distributed processes was controlled using conservative protocol. After that, the fish schooling model was improved and other HPC features were included. Next, in 2005/2006 [102, 101, 100], the simulator started to support a higher number of agents after some optimizations. Also, a halo exchange routine for communication between agents near partition borders was implemented. In 2008, some optimizations were implemented to improve scalability and communication [43]. Next, in 2009, we implemented a version of the simulator using a Fuzzy Logic to represent the behaviour of fish agents [63]. In 2010, obstacles were added in order to improve the environment representation, and different species were simulated in the same simulation space[144]. In 2011, the group research figured out that the partial partitioning could be replaced with a clustering algorithm partitioning [145]. Then, in 2012, a load balancing routine for agents was implemented [146]. And lastly, in 2013, two new MPI communication approaches were developed (BSP and asynchronous) [143].

From this point on, all new implementations in the new simulator (named Care HPS) have considered two basic oriented-object programming concepts: extension of functionality and reusability. The biggest change in the original version of the simulator was uncoupling the agent layer. After that, all important HPC features were redesigned. Then, all previous HPC features implemented in the simulator were also decoupled and included in Care HPS. This process was always cyclic and progressive. In addition, new HPC features were added to Care HPS, as described as follows. In 2014, the clustering algorithm [145] was improved to support the memory shared paradigm [16]. In the same year, we adapted a statistical method defined by [37] to identify the best run length of the simulation. In 2015, we have implemented a new ABM (Shopping agent 4.2.3) and strip partitioning algorithm[18] using the existing agent layer and features to extend partitioning algorithms. In addition, other features were added: serialization, environment layer and model layer. And finally, in 2016, we have implemented a hybrid strip partitioning algorithm and other support features.

5.3 Methodology

Care HPS was designed to be a tool that can be used for researchers, for both the application area user and for the HPC expert user, in order to execute experiments with agent-based models that require high performance parallel and distribution simulations as the main objective.

These two types of users must follow the methodology described in this section which is implemented in Care HPS. First of all, we will focus on the application area user. This user has to follow the stages presented in Figure 5.1.

- Define ABM - in this stage, the application area user must define the agent to be studied. Here, the behaviors of the agent are implemented in order to represent the roles that this agent will follow. Next, the user has to define the environment where the agents will live and interact with other agents. Also, the user has to add the objects to the environment. The interactions between agent and objects are defined in this stage.

- Configure simulation - in this stage, the user has to override a set of Factory Methods [60]. It permits all of the complexity of the HPC to remain hidden for the application area researchers. Therefore, the user has to: (1) define the model class that represents its problem. Here, the user has to specify the agent and environment of the model; and, the HPC features that the model will use such as partitioning, load balance, distribution of environment, etc. In this step, the user can choose which built-in HPC features to simulate its model. (2) initialize the agents; and (3) finally, set up the simulation and prepare the system to begin the simulation. Actions such as: define output file, distributed environment, distributed agent and create synchronization are examples of methods that must be called in order to execute the simulation.

- Output analysis - several files with simulation output are generated for the user. The user has to apply a statistical approach to these data in order to check if they are statistically reliable for a correct output analysis. If this is true, then the ABM scientists can analyze the data and reach conclusions and gain knowledge about the system under study. Otherwise, the user has to execute the simulation again with a greater number of executions. Chung [37] presents methods to identify the best run length of a simulation. An example in ABM can be found in this paper [17]. Finally, it is important to note that the output files can be redefined by the user in order to generate specific information on the model.

Figure 5.2 takes into account the point of view of the HPC experts. We add a new stage and the others are similar to the stages of the application area user with a few changes, which we describe as follows.

- Define or Re-use ABM - the main aim of an HPC expert is to study new HPC approaches for ABMs. Therefore, this type of user can create their own model or use a previously developed model in order to analyze new HPC approaches.

- Configure simulation - in this stage, there is a change in the focus, as compared with the same stage from the point of view of the application area researchers. The HPC expert has to override the Factory Method that will create the approach that is being

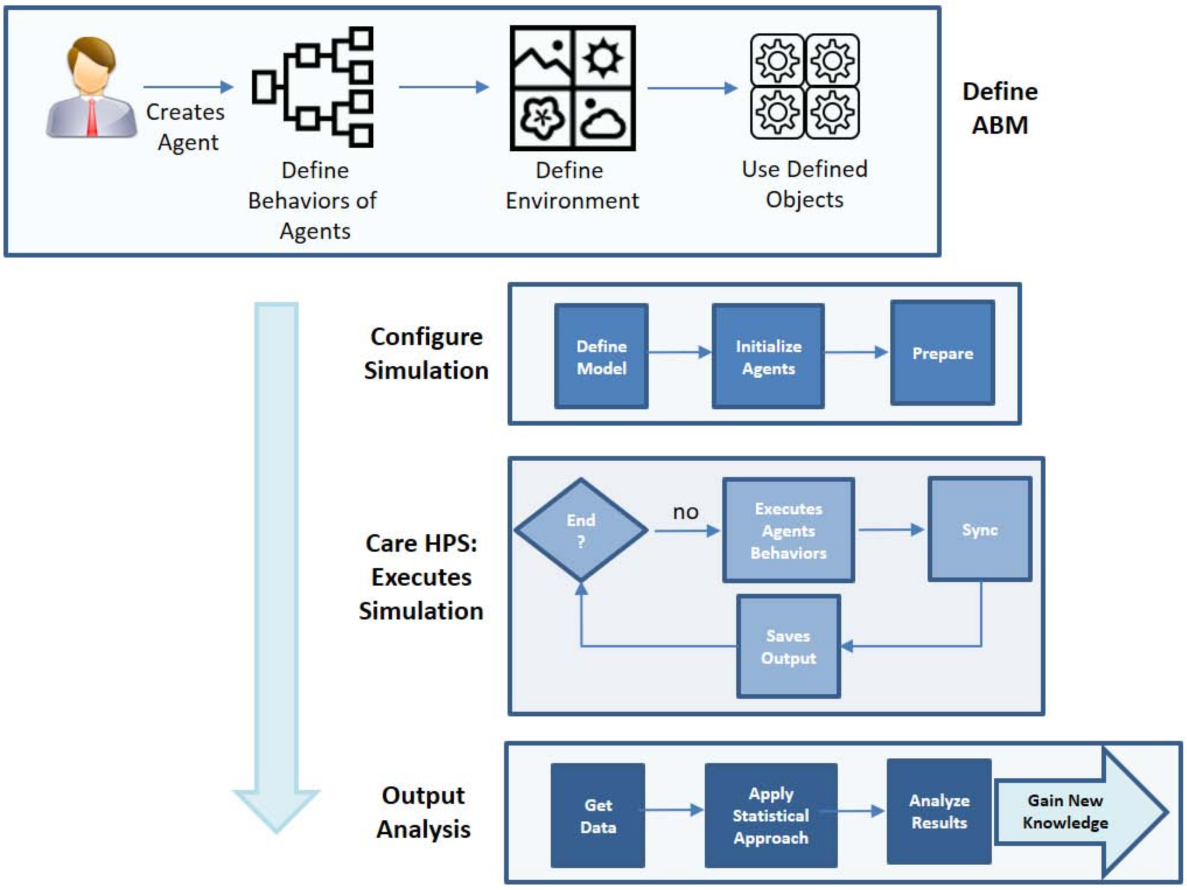


Figure 5.1: Methodology to application area user.

developed or tested. If the HPC expert is creating a new partitioning, then they have to override the Factory method that creates the partitioning. If the HPC expert is creating a new load balancing, then they have to override the Factory method that creates the load balancing, and so forth.

- Output analysis - in this stage, the HPC expert has to check if the data are statistically reliable for a correct output analysis. This user does not need to analyze the results to reach conclusions and gain knowledge about the ABM used because that is not their goal. Their objective is not to study the emergent behavior of a specific model. Their goal is to analyze the behavior of the new HPC approach that is being developed.

- Performance analysis - the user can calculate the performance measures (speedup, efficiency, scalability) using the output files of the new proposed HPC approach.

With this information, the HPC expert can analyze the performance results and reach conclusions about the new proposed technique.

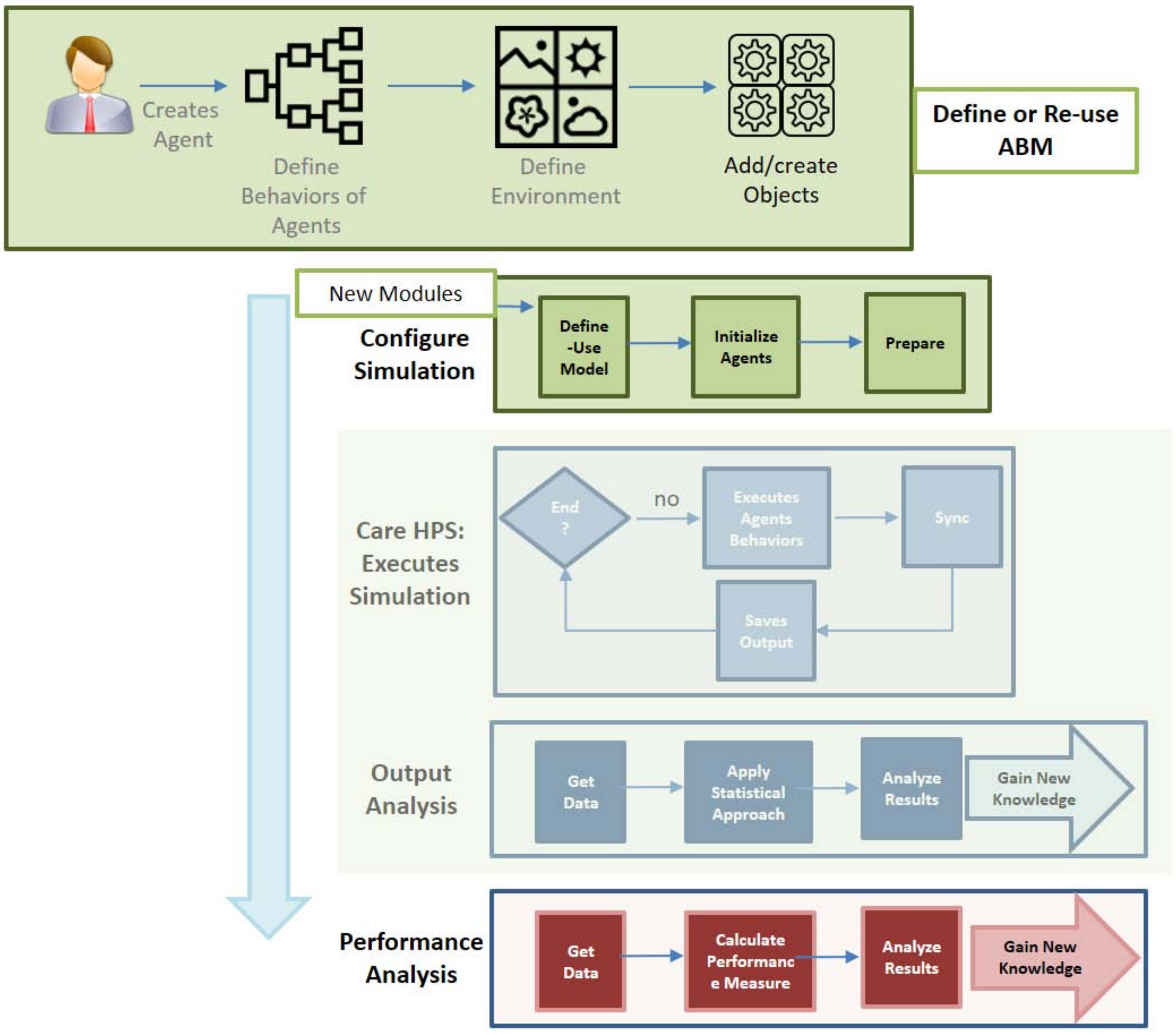


Figure 5.2: Methodology for HPC expert.

Factory methods are the key to hiding the complexity of HPC from the application area researchers, also to enabling the HPC expert to develop new HPC approaches with few code lines. Another important aspect of the methodology is that it follows a "rigorous protocol" and a "contract" defined by interfaces. A new HPC approach, a new model, a new agent, and so on have to respect the interface defined. In this way, it enables both application area researchers and HPC experts to propose new solutions, and Care HPS can handle these new solutions without previous knowledge of them. The next section presents the concretization of this methodology.

5.4 Architecture

Several techniques of software engineering were used to develop Care HPS. These techniques are the basis for Care HPS as a scientific tool. Therefore, Care HPS has been designed with a focus on good oriented-programming practices. The design of Care HPS enables HPC experts to extend functionalities and to reuse previously written code. Care HPS has several layers and components that easily enable the addition of new features and resources. This is possible because Care HPS uses several design patterns [60]: Composite, Facade, Factory method, Strategy, Singleton. These design patterns allow the HPC users to improve their HPC solutions. In addition, Care HPS permits all of the complexity of the HPC to remain hidden for the application area researchers. For more details, definitions and examples of these design patterns see [60]. In addition, other works [111] mention the importance of using a design pattern in ABM architecture. Also, some other solutions [42], [130], [40] have used design patterns to extend and implement their features.

Figure 5.3 presents the current architecture of Care HPS. In addition, the diagram of classes is presented in Appendix A. Care HPS supports the SPMD paradigm. This paradigm enables us to use the task-parallelism and data-parallelism in our behalf to improve the performance of Care HPS. As examples: task-parallelism is used to avoid communication between processes that are not neighbors in the partitioning algorithm; and data-parallelism is used to execute agent behaviors that are located in the same core. Many other features of Care HPS take advantage of these two parallel programming model. The most important aspects of this architecture will be explained in the following subsections.

5.4.1 Agent

The Agent component has a built-in *agent* class that has the main fields required to represent ABM agents in a Euclidean space 2D and 3D. The fields are:

1. *position* - indicates the position of the agent in space;
2. *velocity* - indicates the direction of the agent;
3. *covering radius* - indicates the radius of the iteration of the agent with the environment and other agents;

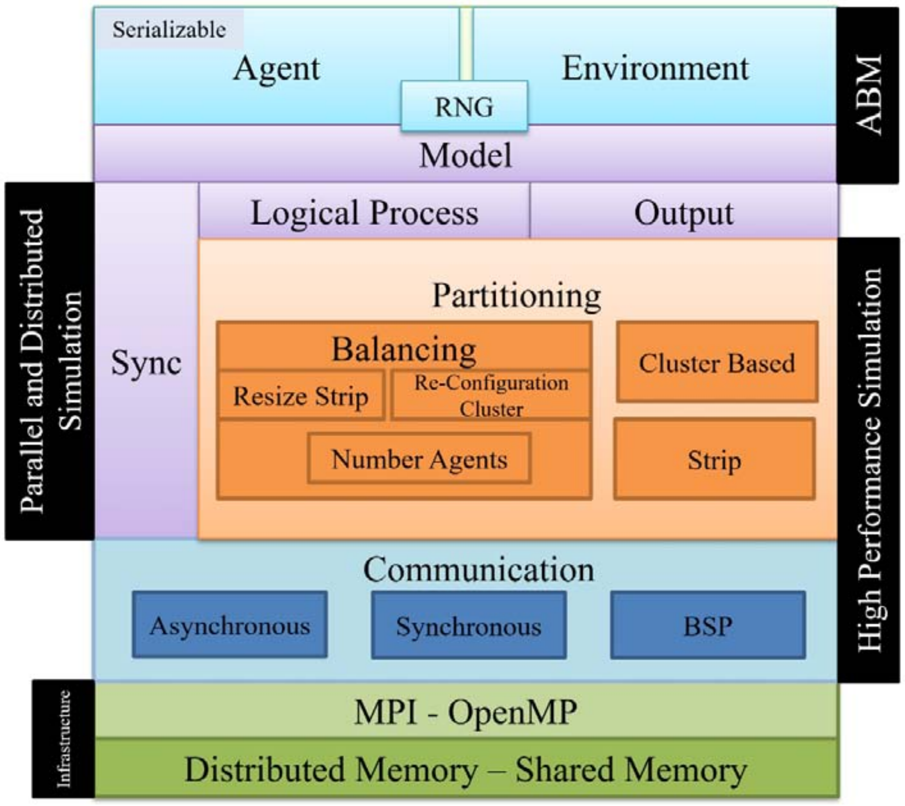


Figure 5.3: System architecture proposed.

4. *member* - used to save and load the state of the agent (serializable component);
5. *ID* - identifies the agent uniquely;
6. *random_move* - indicates if the agent moves randomly; and,
7. *alive* - indicates if the agent is alive. Sometimes the agent is no longer useful along the simulation, and, therefore, it can be excluded from the simulation space.

Care HPS is prepared to receive a class of type *agent*. This is the most important class that has the responsibility of implementing the behavior of agents. The problem modeled sometimes requires additional fields that can be included in the user's agent classes (concrete class of agent). Methods of the concrete agents, which implement local roles and behavior, also depend on the problem modeled. Therefore, the user has to implement these methods and call them in an appropriate order and using logical and/or conditional statements inside of the basic method called *update_agent*. This method is executed many times by Care HPS from the beginning to the end of simulation. Care HPS does not implement a schedule of events. Rather, all agents are added in a bucket. At each simulation step, the system calls the *update_agent* method of each agent inserted in the bucket. The group behavior emerges as a consequence of the agents' interactions occurring in this loop execution.

The agent can interact with other agents or environment inside of its covering radius. This value can be changed in the execution time in order to implement different behaviors of agents. This same concept, called an area of interest (AOI), is used in other solutions such as [42].

Another important feature of the agent class is the ability to save and load the state of the agent. This feature will be explained with more details as follows (see Subsection 5.4.5).

5.4.2 Environment

The environment in ABM simulation is the space where the agent has "life". The environment holds the agents that interact with the environment and with other agents. Currently, Care HPS supports Euclidean Space for 2D and 3D. To create an environment in a model, it is necessary to extend the classes: *environment* and *object*. These two classes compose the environment component. The *environment* class will hold agents, and it will be the container of possible objects created inside of it. The objects of the environment might be created or extended from the *object* classes.

With the aim of implementing this feature, we are using a math approach to model and represent the environment. This approach enables us to dynamically create a representation of any type of obstacle or resource. The environment class must be specialized in order to represent the desired environment. Each environment should be composed of objects which might represent the objects of the real world, such as buildings, food, paths, etc. Care HPS enables the user to define objects with the most appropriate representation for the problem. Therefore, objects must be specialized to achieve this. For example, we can use the linear_plan class to simulate a coast, valley and rocks, as an example, at the sea.

We chose to use a math approach instead of using agents to represent an environment, as we can find in other ABM approaches, because the math approach supports a wide range of representations of space. Several math expressions can be added in the same environment class as objects, and many combinations can create the required environment. Also, this approach easily supports the interaction between agents and objects of the environment with simple math operations of intersection between an agent's vector and the equation that represents the object.

Care HPS enables us to replicate the environment in all processes. This means that all processes have a full copy of the environment. This avoids communication with other processes in order to get this information. This option should be used when the resources of the environment are not consumed during the simulation.

5.4.3 Random number generation

Random number generation is a useful feature that is also available in other ABM tools mentioned in Chapter 3. Care HPS provides a component for random number generation. The user can create: normal, exponential, gamma and uniform distributions and get random numbers within a range. The user can use this feature to create the initial input of a simulation as well as to define random values for agent behavior such as velocity or position.

5.4.4 Synchronization

The simulation advances in steps and a sync is therefore necessary before the next step. The synchronization can be used for many purposes: maintaining the simulation coherence, executing load balance, executing compute balance, or re-partitioning. After collecting other information, the system can decide if some of these routines must be executed. As the synchronization uses a value in order to make decisions, Care HPS enables the user to choose which model method to call. For this, the user must implement the *wrapp_to_synch* method, which is a callback method that will be called by *simulate* method of a concrete class of *pds* interface. Care HPS user can choose which reduced operation will be executed.

The design of Care HPS allows other synchronization approaches to be implemented and proposed. Care HPS implements a conservative [77] protocol in synchronization. In this implementation, the processes are blocked until all distributed buckets have finished the execution of the last simulation step. Each process has its own bucket with agents. For all agents, their roles are executed when the *update_agent* is called by the system (see Subsection 5.4.1). This conservative protocol implementation may be used in order to make decisions, such as finalizing the simulation, executing a load balancing, or other control actions. Synchronization creates a barrier in order to wait for all processes to finish a specific step and then initialize the communication processes among the processes. Therefore, in this type of protocol, it is extremely important that all processes are balanced in order to ensure that all processes will arrive at the synchronization point at almost the same time. Otherwise, a performance degradation can occur.

In contrast with other ABM tools, such as Repast HPC, we do not implement a scheduler to iterate the simulation forward. The execution approach of Care HPS is quite simple. We consider the agents to be autonomous and therefore know what they must to do and when to do it.

5.4.5 Serializable

Serializable is a required feature in parallel and distributed simulation because the agents might migrate from one process to another. This operation is common in an ABM distributed solution, and it can occur as a consequence of a local role, as a motion in space, or because of a load balance or partitioning routines. For basic fields of agents the state is saved automatically by Care HPS, additional fields must be added by the user in the field *member* of the agent. As we said before in Subsection 5.4.1, all agents must be inherited from the class *agent*. This class has the main properties of the agent. The user's model may create concrete classes in order to specify the agent's behavior that represents the model. Therefore, new agent properties can be added by the user. The serializable feature enables Care HPS to know what the new properties are and, consequently, it can use this information in the communication process. The main purpose of the serializable feature is observed when the migration of agents among processes occurs and the serializable information is used in the communication process.

In the migration processes, the agents are deleted of logical processes and created in other processes. Therefore, the agents' states must be preserved in this operation. The *serializable_member* class has all of the members which must save their state in the communication processes. This mechanism is required because Care HPS has a generic agent implementation. Therefore, Care HPS needs to know which properties created by the user must be preserved when the agent moves to another partition.

The user's applications do not need to implement any communication routine. This is provided by the communication layer (see 5.4.7). But the user must specify which properties must preserve their state. All states are saved and loaded in the *member* data structure. This data structure supports the main primitive type data of C++ language. The user has the responsibility of saving and loading the field in the same order. Only the members created at concrete agent classes need to be specified by the user. The base fields of agent class have their states saved automatically by Care HPS. We delegate to the user to inform which fields are new and which ones must be saved. This is compensated by the fact that the user does not have to worry about manipulating the communication layer. The serializable mechanism is the key to automating the communication process for the user. The communication process works with the agent superclass; therefore, it does not know about the specialized fields.

A similar approach is found in the literature [40]. However, Care HPS solution is more simple. In the literature, the user has to specify each field of the agent class, provide the serialization code, execute the pack, and execute the unpack. In Care HPS, the user needs only to manipulate the serializable component, inform which fields need to be saved, and call the save and load methods at the appropriate moment.

5.4.6 Partitioning

Data partitioning is the most important aspect of parallel and distributed solutions because the distribution of data over a distributed architecture has a strong correlation with communication, load and compute balancing. Care HPS provides four partitioning strategies: cluster-based partitioning [145], Hybrid cluster-based partitioning [16], strip partitioning with a pure MPI solution [18] and the hybrid strip partitioning. Moreover, Care HPS enables HPC experts to extend new partitioning algorithms.

The cluster-based partitioning approach can be used for ABM, where its agents' interactions can be grouped into clusters [78] after an unsupervised classification of patterns. The criteria used in the cluster-based method is based on Voronoi diagrams and covering radius [145].

The hybrid cluster-based partitioning approach uses OpenMP parallelization technique called fine-grain. This approach consists in the parallelization of the loop nests in the computation part of the MPI code [27]. We have focus on the main functions of the partitioning algorithm. The main challenge of our hybrid parallelization is to use the current list of clusters data structure (Figure 3.3) with OpenMP threads concurrently. As an example, see Figure 5.4, where the list of cluster is represented by variable *lc*. This is a vector of list of cluster. Each MPI rank, specified by the pid variable, manipulates just the Agents that are in it. The insert method includes a new Agent in the bucket. This has the same behavior when we need to include new clusters inside of the list. The problem in this solution is that many cores execute this operation concurrently through the shared variable *lc*. Therefore, the critical section is required. Critical section enables just one thread to have access to this code region at the same time. Thus only one active thread can access the data referenced by the code.

The data structure proposed and used in hybrid cluster-based partitioning deals with the simultaneous inserting, deleting and updating problems. The new data structure is presented in Figure 5.5. In this new data structure, we create dynamically a vector of meta list of clusters. The size of this vector depends on the number of processes created by the simulation. The Meta list of clusters is composed of two fields: 1) ID, which has the MPI rank number; and a vector which has a *n*-list of clusters, where *n* is equal to thread number created by MPI process. This data structure enable us to reduce, almost eliminate, the critical sections, because, the clusters are distributed among the threads and each thread executes its operations in its own list of clusters. MPI process continues working with the list of clusters (Figure 3.3) but the computing is executed in the meta list of clusters (Figure 5.5) when the OpenMP section begins. After data are computed through the OpenMP threads these data are consolidated in the list of clusters of MPI process. The consolidation operation does a sequential copy of agents from the OpenMP data structure to the MPI data structure.

Figure 5.4: Portion of code with critical section.

```
for (long pid=0;pid<lc→length();pid++) do
    if (lc→at(c)→get_pid()==pid) then    ▷ The variable "c" is a cluster. Check if the cluster is in this process
        #ifdef OPENMP
        #pragma omp critical ...    ▷ OpenMP pragma critical section
        {
        #endif

        lc[pid]→insert(a); ▷ The variable "pid" is MPI Rank process and the variable "a" is an agent that will be inserted at a cluster

        #ifdef OPENMP
        }
        #endif
    end if
end for
```

Therefore this effort is valid just when the time spent in this operation is less than the communication time of the MPI processes. In the pure MPI version, one MPI process is created per core. On the other hand, the number of MPI processes is reduced by using the hybrid solution. This decreases considerably the communication time and all the inherent overhead of the management process and its context. In the hybrid solution, there are more clusters doing computing inside the same MPI process. Consequently, some communications and computing among the cluster are eliminated. Figure 5.6 shows a code example using Data Structure 5.5.

The strip partitioning approach fits better with ABMs which have a spatially dependent problem. The strips are created in accordance with the number of MPI processes. This partitioning algorithm chooses the best partition configuration considering the number of processes and the distribution of objects in the environment. This strategy avoids objects being shared by processes [18].

The user chooses the partitioning approach through the factory method of *model* class. The partitioning has two main methods public to the user: *distribute* and *execute*. The first method distributes the agents over the distributed architecture; and the second one executes the agents' behaviour which calls the *update_agent* method (see Subsection 5.4.1). The distribution of the agents and their behaviour's execution are completely transparent for the user. Also, other functionalities such as the migration of agents, point-point communication, and loading the state of agents are executed by partitioning algorithms.

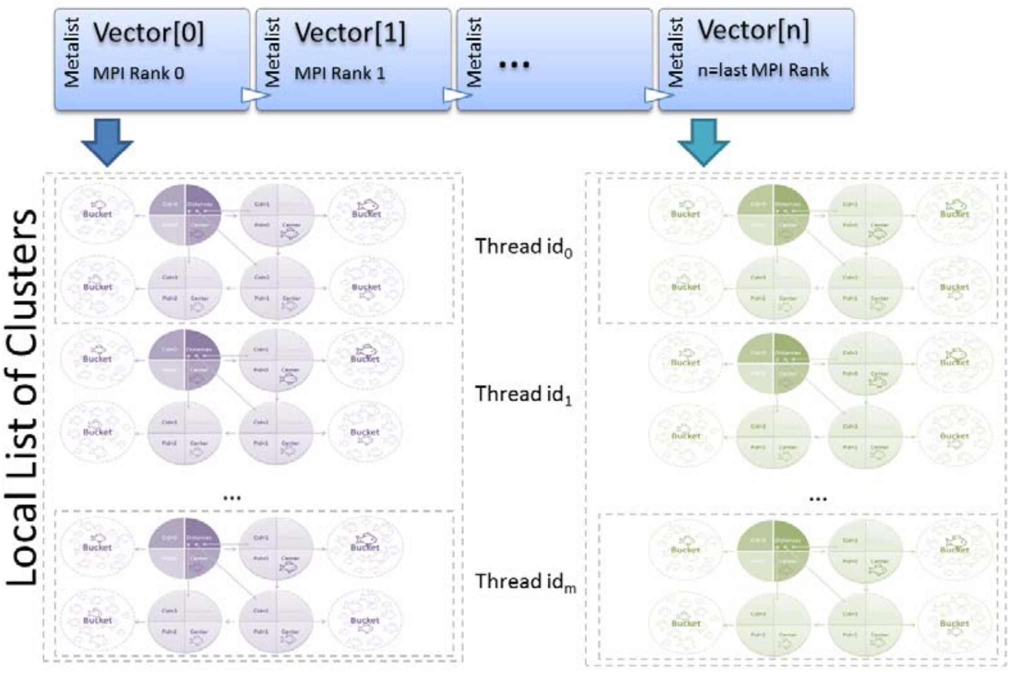


Figure 5.5: Meta list of clusters proposed in order to avoid critical section in the OpenMP code.

5.4.7 Communication

Care HPS communication processes occur basically when: 1) there is a migration of agents among processes; 2) the processes exchange information for the synchronization process; 3) initialization of the simulation; and 4) when the load balancing, compute balancing and partitioning are executed. There is no direct communication between the agents through MPI routines. However, there is local communication between agents through memory copy or shared memory.

The communication layer is hidden from the user. They do not need to know anything about it. Only the partitioning algorithms interface with the communication layer. This layer uses the pack and unpack pattern in order to send and receive data. Care HPS has implemented three communication patterns: asynchronous, synchronous and BSP [143]. Figure 5.7 presents a schema of the communication strategies implemented. Asynchronous and synchronous message passing strategies were implemented using OpenMPI. For asynchronous strategy, we used MPI_Isend and MPI_Irecv. For synchronous strategy, we used MPI_Send and MPI_Secv. And for bulk-synchronous parallel strategy, we used the BSPonMPI[149] API. BSPonMPI is a widely used implementation of the bulk synchronous parallel (BSP) computing model.

Figure 5.6: Example of cluster insertion with proposed data structure and without critical section.

```
vector<list_of_clusters* > vector_lc;              ▷ create a vector of list of cluster
...
for (long pid=0l;pid<this→nprocs;pid++) do                          ▷ for each process...
    for (unsigned int i=0l;i<this→num_thread;i++) do                ▷ for each thread...
        vector_lc.push_back(new list_of_clusters());
    end for
    metalist_lc.push_back(make_pair(pid,vector_lc));
end for
...
metalist_lc[pid]→lc_thread[i]→insert(c);    ▷ The variable "pid" is MPI Rank process,
                                  ▷ "i" identifies the thread that has to manipulate these data,
                                                ▷ , and the variable "c" is a new cluster

```

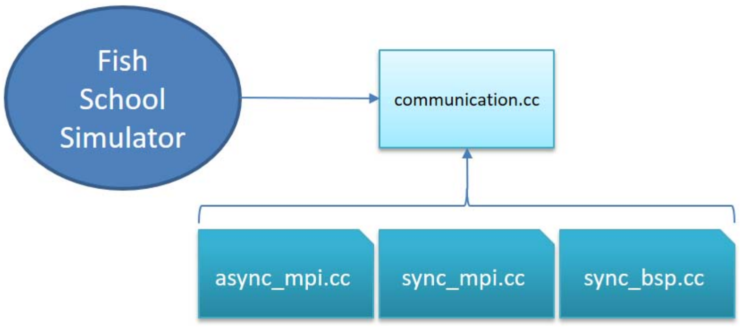


Figure 5.7: Communication strategy implemented.

The MPI communication routines used in Care HPS depend on the partitioning used. For example, the cluster-based partitioning needs a broadcast communication, because sometimes the agent migrates to an unknown process. In the strip partitioning approach, the communication occurs just among the neighboring processes, because the agents can only migrate to partition neighbors. Therefore, only MPI_Send and MPI_Recv routines are required.

5.4.8 Load and computing balancing

The partitioning algorithm distributes the data throughout the architecture. In an ABM parallel and distributed solution, the algorithm of partitioning must distribute the agents equally among the processes. A well-done partitioning algorithm could ensure a balance among the processes, if all agents have similar computing and communication throughout the whole simulation and there is no agent migration. However, this situation is not common. Generally, the agents are executing different roles at the same moment and the migration between processes is a common procedure in distributed simulations, especially in Euclidean space simulation, where the agents might move constantly.

Therefore, during the simulation, two situations can occur: 1) some processes can contain more agents than others; and/or 2) some processes can spend more time computing than others. This first situation is due to the migration of agents between processes; and the second one is because the distribution of agents does not take into account the computing differences between agents. These situations have a direct impact on the total execution time in simulations that use conservative protocol, because the faster processes have to wait for the slower processes to finish before moving forward to the next step in the simulation.

Load and computing balancing is an issue too complex to get effective results, because many aspects have to be considered in order to implement these routines. In Care HPS, the algorithms of load and computing balancing are designed as a composition of the partitioning of data. The distribution of data has particular properties that should be considered in order to develop a feasible load balance routine. Therefore, the algorithms of load and computing balancing should act on partitioning of data algorithms. Currently three balancing policies are implemented: migration of cluster [146], resize of the strip and number of agents. In addition, new strategies can be implemented and extended.

5.4.9 Model

A model in Care HPS can be built using the following interfaces: *agent*, *environment*, *pds* and *model*. The behaviours of the agents are defined in a concrete class of *agent*. The environment is represented by the *environment* and its objects.

The concrete class *model* requires some methods to be implemented by the user in order to define: 1) the size of the agent - this information is used at the communication layer to execute the pack and unpack; 2) the type of agent that the simulation will create; 3) the type of object that the environment will contain; 4) the type of environment; and, 5) the partitioning of data approach.

This interface *pds* is the Abstract Factory class in the Abstract Factory design pattern [60]. With a concrete class *pds*, it is possible to create products related with the simulation, such as model, partitioning, load balancing, environment, agent, etc. This class is composed of many factory methods that must be overridden by the user in order to define which object will be created in execution time. As an example of the most important factory methods, we can cite: *getModel* - create or get the model; *initialize_agent* - initialize the agents of simulation; *prepare* - setup and define behaviour and parameters of the simulation; *simulate* -

execute the simulation steps; wrapp_to_synch - defines which method will return the value for the synchronization.

The concrete class *pds* represents the parallel and distributed simulation of the specific *model*. *pds* uses *model* to create the *agent* and *environment*. Also, the concrete class *pds* configures and defines the characteristics of simulation. In Section 5.5, we present some example using these classes.

5.5 How to use Care HPS as a scientific instrument

In this section, we will give some examples of how to use Care HPS from two points of view. In the first, we will focus on the application area researcher' point of view with the following examples: 1) how to implement an environment with an obstacle for a fish; 2) how to create and extend an agent using a buyer agent as an example; and 3) an implementation of Buyer Model Versions One and Two. We will not explain all of the implementation detail, but will instead focus on the most important aspects. In the second point of view, we present the implementation of a hybrid strip partitioning based on [18]. We will show how to use Care HPS if the HPC expert user wants to propose a new HPC solution.

5.5.1 Application areas researchers' point of view

Application area researchers are interested in how to model their problems in ABMS. Therefore, in this section, we demonstrate how to represent the Agent, Environment and Model using Care HPS.

Creating an environment

ABM requires an environment where interactions occur between an environment's objects and the agents. The agents must execute local rules when these interactions happen. As an example of this kind of interaction, we can cite the repulsion behaviour in the Fish Schooling model. The new fish's orientation depends on its position and orientation. If there is an obstacle ahead, the fish must change its direction in order to avoid a collision with the environment's objects. Consequently, this fish behaviour influences its neighbors after the local interactions. Listings 5.1 and 5.2 show implementations of this fish's behaviour. We create one linear plan defined by Equation 5.1 that represents a rock:

$$2*x-13*y+5 \tag{5.1}$$

We check the intersection between the plan and the vector equation of the line representing the fish's position and direction. The vector equation of the line is defined by the equation:

$$P = A + tv \tag{5.2}$$

Where P indicates the next position of the fish. A represents the actual position, v the direction, and t is a constant which represents the maximum vision range of the fish. We need a vector equation of the line in order to verify if there is a point (P) that is the root of the equation that represents the object of the environment, in this example, represented by the object described by Equation 5.1. Therefore, the code sums the current fish position (A) with the next fish position (tv), where t is defined as a characteristic of the agent fish (MAXIMUM_VISION_RANGE). The code checks if this new point (next fish position) is a root of environment equation. If so, the repulsion behaviour should be executed in order to avoid the collision.

Listing 5.1: Creating an environment for the fish

```
pds_fish* PDS_FISH=NULL;
ABM_fish* ABM=NULL;
environment* ENV=NULL;

PDS_FISH = new pds_fish();
ABM = PDS_FISH->createABM();


/** Creates an environment with a
plan(ax + by + cz + d = 0) defined by
equation: 2*x - 13*y + 5 */

ENV = PDS_FISH->createEnvironment(
                    new linear_plan(2,-13,0,5));

/** The user can create how many objects inside of
the environment that are required. */
/** Code that creates other objects inside of the environment
defined by the equation: 2*y-3 */

// ENV->createObject(new linear_plan(0,2,0,-3)); //
```

Code 5.2 shows Equation 5.2 in the parametric form.

Listing 5.2: check_collision implementation

```
bool linear_plan::check_collision(vector3d Ap,
                 vector3d v, float t){

/** The point P of Vector equation of a
line is represented by x, y and z variables. */
float x, y, z;

float distance;

x = Ap.get_x()+ (t*v.get_x());
y = Ap.get_y()+ (t*v.get_y());
z = Ap.get_z()+ (t*v.get_z());

distance = abs(
this->a*x + this->b*y + this->c*z + this->d)/
sqrt((this->a*this->a)+(this->b*this->b)+
            (this->c*this->c)
            );

return (distance<=t);
}
```

The agent must call the *check_environment* method 5.3 in order to interact with the environment. This method implements the interactions between the agent and the environment. The code checks if the collision happens for each object that has been created in the environment. If there is a collision, then the repulsion behaviour of the agent is executed. Any other behaviour of the agent can be used.

Listing 5.3: check_environment implementation

```
void fish::check_environment(void){

vector<object*> obj_env;
obj_env = ENV->getObjects();
for (vector<object*>::iterator ob=obj_env.begin();
                                    ob!=obj_env.end();
                                    ob++)

 if ((*ob)->check_collision(this->get_position(),
                                this->get_velocity(),
                                MAXIMUM_VISION_RANGE))
                 this->repulsion(*this);
}
```

Until now, the objects in Care HPS have been represented by linear plan and circle. Care HPS enables the users to extend the object class by adding other types of objects such as surfaces, objects with special characteristics, squares or other types of geometric shapes.

Creating and extending an agent

In this section, we present how to create a new agent and how to extend it.

1. Creating a new agent

   The user has to create a class that inherited from *agent* class 5.4. After that, the user has to declare and define some methods. In this class, the methods that represent the behaviour of the agent must be defined and these methods must be called inside the update_agent method.

   Listing 5.4: Creating a new agent: buyer

```
#include "agent.h"

class buyer: public agent{}{

        //The user must declare and redefine some methods
        // This is a pseudocode of the update_agent method
        update_agent(void){
        if (something)
                behaviour1();
        else{
                behaviour2();
                behaviour3();
        }
}
        // Pseudo methods that represents the buyer's
        // behaviours
     behaviour1(){
                do_this;
       }
     behaviour2(){
                do_that;
       }
     behaviour3(){
                do_nothing;
       }
 }
```

2. Create a new Buyer Version 2 extending the buyer agent

   To create a new version of the buyer, the user just needs to extend Buyer Version 1 and override the methods required.

   Listing 5.5: Creating a new Buyer class

```
#include "buyer.h"

class buyer_v2: public buyer{};
        //Override the methods required

#endif
```

3. Create a new model for Buyer Version 2 that extends the previous model buyer class. This model class is required because it will represent the model and manipulate the Buyer Version 2 created in the previous step.

   Listing 5.6: Creating a new model Buyer Version 2

```
#include "model_buyer.h"

class model_buyer_v2: public model_buyer{

};
#endif
```

   Another option could be to create a subclass directly from the *model* class. Extending the model buyer class is a better option because we can reuse some methods.

4. Overriding the factory method that creates the agent in the class: *model_buyer_v2*. These methods must call the constructor of Buyer Version 2. Through these factory methods, Care HPS can dynamically create any type of agent. This is a key point of Care HPS.

   Listing 5.7: Overriding the factory methods of model_buyer_v2 class

```
class model_buyer_v2: public model_buyer{

// Other overloads of the factory methods

shared_ptr<agent> model_buyer_v2::factory_agent_sptr(agent &i,
                                                      bool serializable){
```

```
_tmp_a = dynamic_cast<buyer_v2*>(&i);

if(_tmp_a)
        return shared_ptr<buyer_v2>(new buyer_v2(*_tmp_a,
                                        serializable));
else
        throw std::runtime_error(The agent could not be
        converted for a buyer_v2! Check which type of agent
        you have inserted.");

}

}
```

As we can see in Listing 5.7, the user just has to change the agent's constructor to create the buyer smarter. The code invokes the factory method that has the Buyer Version 2 constructor instead of the buyer constructor.

5. Create a pds_buyer_v2 class extending the previous pds_buyer class

   The *pds* class defines which model will be used in the simulation. The user must redefine the constructor, destructor and override the *getModel* method in order to specify the model that will be created at runtime.

Listing 5.8: Creating a new pds class (pds_buyer_v2.h)

```
class pds_buyer_v2: public pds_buyer{
public:
        pds_buyer_v2(void):pds_buyer(){};
        virtual ~pds_buyer_v2(void){};

        model_buyer_v2* getModel(void){
                if (this->_model==NULL)
                        return new model_buyer_v2();
                return (model_buyer_v2*) (this->_model);
                }
};
```

This section presented the few classes that must be implemented by the user in order to represent an environment, model and agent using Care HPS.

5.5.2 Expert HPC user

One of the main purposes of Care HPS is to enable the expert HPC user to be able to develop and propose new techniques at HPC for ABM. In this section, we give an example of how to develop a new partitioning approach using Care HPS. We will implement the hybrid strip partitioning presented in 4.3.4. The hybrid strip partitioning is implemented with OpenMPI and OpenMP.

Developing a new partitioning approach

To implement a new partitioning strategy, the expert HPC user needs to follow some simple steps. Note that other extensions can follow the same sequence because all other features have a similar design:

1. Creating the partitioning class

   The first step is to create a class that inherits *partitioning_strip* class 5.9. The new partitioning class could be inherited from a partitioning interface, but we chose inheriting directly from the *partitioning_strip* because we reuse all methods, except the method *execute*.

   Listing 5.9: Creating a class that inherits from the partitioning strip.

```
class partitioning_strip_hybrid : public partitioning_strip{

    public :
                    // Contructor methods

                    // Override the execute method
                    void execute(int);
};
```

2. Implementation of partitioning strategy

   Listing 5.10 presents the implementation of the *execute* method. In this code, we do a loop parallelization, dynamically creating the number of threads, using Equations 4.7 and 4.8, that will execute the behavior of a set of agents.

   Listing 5.10: Implementation of partitioning strategy.

```

void partitionig_strip_hybrid::execute(){

        //here goes the other partitioning codes
long thread = number_agent/total_number_agents*number_cores;
if (abs(num_thread)>=1){
        if (abs(thread)==1)
          omp_set_num_threads(2);
        else if ((abs(thread)>=2) and (abs(thread)<=8))
          omp_set_num_threads(abs(thread));
        else if (abs(thread)>8)
          omp_set_num_threads(8);

        #pragma omp parallel default(none) firstprivate(i)
                        shared(_bucket)
        {
         unsigned long j;
         #pragma omp for private(j) schedule(dynamic) nowait
         // For each agent execute its roles.
         for(j=0;j<_bucket->size();j++)
                _bucket->at(j)->update_agent();
        }
}
else{
  // Execute the loop without thread just with pure MPI
}
};
```

3. Changing the concrete model class

   After finishing the partitioning strategy implementation, we have to change the concrete class of model that will use the new partitioning. The only change in this class is the implementation of the factory method *factory_partitioning()*. This factory method defines which partitioning algorithm will be created.

   Listing 5.11: Changing the code of concrete model class: model_ant.cc

```
//here goes other the factory methods of the model class.

partitioning* model_ant::factory_partitioning(){
        return new partitioning_strip_hybrid();
}
```

If the expert HPC user wants to test other already implemented strategies of partitioning, for example, the user just has to change the factory method and no other changes are required.

Therefore, the expert HPC user can invest time in testing, improving and proposing the HPC technique instead of doing model implementations or other needed implementations.

5.6 Comparison with other tools

With Care HPS features in mind, we will present the main differences among similar tools found. We are concerned with ABM tools that have support for parallel and/or distributed executions of the model and with similar features to Care HPS. In the bibliographical survey done in State of art 3.7.1, we found the following tools: FLAME [69], Netlogo [163], D-Mason [42], Pandora [130], and Repast HPC [40]. For the main features of Care HPS, see Table 5.1, we will explain how each ABM tool address these issues. This comparison will focus on how each tool addresses a feature rather than strengths and weakness, because, in spite of supporting the parallel and distributed execution, each tool has a different domain and was designed for a different purpose than Care HPS.

5.6.1 Agent

Care HPS has a built-in agent class that represents ABM agents in a Euclidean space 2D and 3D. The user can extend classes from this base class in order to model their problem. In FLAME, the agents are represented by X-machine [39], and its behaviour is defined by a directed acyclic graph. X-machines are similar to finite state machines; however, the main difference is that X-machines have a memory that is used at transitions between states [38]. D-Manson implements the agent extending an abstract class called *RemoteAgent*, which enables the agent to be serializable. The agent actions are defined at the method *step*, where all behavioral models of the agents are implemented. In this method, each agent computes its new state and stores it in a buffer until synchronization. Pandora uses the *Agent* class that encapsulates any entity of the model. The user must define the *updateState* method to specify the state, decision-making processes and behavior of the agent. This method is executed at every simulation step [130]. Repast HPC represents its agents by C++ class. The user has to implements their agent class and behavior by methods in this class [40]. Also, all agents must implement the *Agent* interface [40]. All these tools allow for the extension of a base class (or proprietary unit: X-machine) where the agent's behaviour can be implemented.

5.6.2 Environment

The interactions among agents occur inside of the environment. Depending on the problem, the environment can be static, where there are no changes in its objects, or dynamic, where the interactions among agents or the interactions of agents with the environment can change it. Care HPS represents the environment through two classes: *environment* and *object*. The *environment* class is a container for the *object*. The interaction between agents and the environment is implemented using a math approach. FLAME considers that the environment must be represented by an agent if the environment is changeable and an agent can interact with it [136]. Therefore, FLAME uses an X-machine in order to represent an environment. D-Manson has built-in classes that represent a space at 2D and 3D, and the environment can be represented as Grid or Continuous space [86, 88]. Pandora extends classes to represent an environment. The *World* class uses an XML file with parameters for the initialization and creation of the environment. The *Config* class provides access to the XML file for *World* class.[130]. Both of these classes are abstract classes and must be implemented by the user. Repast HPC implements the environment with Context and Projection concepts [40]. A context contains a set of agents and has Projections associated with it [40]. A projection is a way of structuring agent relationships [6]. The Repast HPC has a spatial projection that takes into account the proximity of the agents and the network projection where the agents are connected explicitly with each other [6]. Some of the tools listed here use their own agent classes in order to represent the environment. The advantage of this approach is that the user can handle the whole simulation composed of objects. So, the communication between an agent and the environment is through the messages of the objects. As presented in Section 5.4.2, we chose a math approach to represent an environment because: 1) we believe that representing the environment and its objects as agents hurts the semantic of the agent definition in an ABM context; 2) the math approach allows for easier implementation and interactions with objects in a Euclidean space; and 3) this approach enables the representation of a huge number of environments and objects.

5.6.3 Synchronization

The synchronization operation in distributed processes is generally a critical operation because it is a block operation and, therefore, the faster processes have to wait for the slower processes. Overlap routines can be used as a possible technique to minimize this problem, but this type of code cannot always be used because of the data dependence. We think that the synchronization can be adapted in accordance with the problem. Therefore, it is important to have a public interface for extension by users. Care HPS enables the user to define which method will be used for synchronization. A callback function calls this method and uses its returned value to make a decision. FLAME controls the synchronization through the message boards to ensure that all agents can see the same set of messages [34]. D-Mason implements a local step-by-step synchronization of each region. Each region is synchronized with its neighborhood before each simulation phase [42]. Pandora defines the synchronization process through a scheduling system. The user can define its own scheduler solution [130]. Repast HPC handles any necessary synchronization using a discrete event scheduler [40].

5.6.4 Serializable

Serializable is a required feature in parallel and distributed simulation because the agents might migrate from one process to another. For basic fields of agents the state is saved automatically by Care HPS, and additional fields must be added by the user in the field *member* of the agent. The serializable feature can be implemented by memory in the X-machine in FLAME. D-Mason uses the publish-subscribe design pattern to propagate agent state information [42]. In this way, the agent is able to update and maintain its state at each simulation step. Pandora saves the states of the agent in a file format. This file format supports high performance computing applications [130]. Repast HPC supports the serializable feature through serialization. The user has to provide a serialization code to extract the agent state, package it for transfer and then unpack the transferred package [40].

5.6.5 Partitioning

Data partitioning is the most important aspect of parallel and distributed solutions. The distribution of data over a distributed architecture has a strong correlation with communication, load and compute balance. Care HPS provides three partitioning strategies: cluster-based partitioning [145], pure MPI strip partitioning [18] and hybrid strip partitioning []. Beyond that, Care HPS enables HPC experts to extend new partitioning algorithms such as the hybrid strip partitioning presented in this paper. The data partitioning of FLAME occurs at the agent level [34]. Two static partitioning approaches are available, which can be chosen by the user: Separator Partitioning and Round Robin Partitioning [33]. Separator Partitioning is a kind of geometric partitioning where FLAME takes into account the position of the agents in order to define the partition that will be allocated [33]. In Round Robin Partitioning, FLAME distributes the agent at a time to each partition in turn. Agents of a specific type would be allocated to one or more set(s) of partitions [33]. D-Mason implements a space partitioning. The space simulated is partitioned into regions by the master. Each region has a set of agents and each region is assigned to a worker [42]. Pandora implements a spatial partitioning. Each node receives a part of the simulated environment and the agents located within its boundaries. Each part of the environment is divided into four different sections, and the sections are executed sequentially [130]. Repast HPC implements the partitioning through a projection concept. The user can choose from among three types of projection: grid, continuous space and a network [40].

5.6.6 Computing and loading Balancing

Parallel and distribution simulations can become inefficient if they are not accomplished with a loading and computing balancing. When the ABM is a spatial dependence problem, the movement of an agent from one partitioning to another as a consequence of a loading or computing balance action is quite complex, because these solutions have generally already predetermined which partitioning will execute that part of the problem. All of the simulation space is divided among the partitions. Therefore, if the agent is moved from one partition to another, then it will act in a different part of the simulation space. Each core is responsible for the process of one or more partition(s). Redefining which core will execute the partition is not a problem, but all environment, objects and agents allocated at the partition will also be executed. Therefore, increasing or decreasing the granularity of the simulation space might be a good strategy for spatial dependence problems. On the other hand, when the problem is not spatially dependent, then the movement of an agent to another partition as a consequence of a loading or computing routine is more simple. But the movement of the agent can depend on the ABM problem, therefore, an extension of the available solution of the loading and computing balancing can be required. The loading and computing balancing in Care HPS is dependent on the partitioning strategy. Care HPS enables one or more balancing policies to be implemented for each partitioning strategy. Currently, three load balance policies are implemented and new balancing algorithms can be proposed and extended. FLAME considers that the communication among the agents has a more significant impact on the total execution time than the light-weight computational nature of many agent types [33]. FLAME does not implement a load balance routine directly. It should be reached using a partitioning strategy, as in the example presented in [38]. According to [96], FLAME has no routines to allow the migration of agents between the processes. D-Mason deals with load balancing through the granularity of the simulated space's decomposition [42]. The granularity of the partition must be defined by the user and it defines the number of regions to be assigned to each worker. The problem with this approach is that agents can migrate between regions, and, consequently, one region can hold more agents than another. Therefore, the load balancing is not guaranteed and needs to be addressed by the application [42]. In Repast HPC, the load balance can be reached through projection parameters. Some examples can be found in [6].

5.6.7 Communication

The communication process is the key for performance in parallel and distributed solutions. Care HPS communication occurs when: 1) there is a migration of agents between processes; 2) the processes exchange information for the synchronization process; 3) initialization of simulation; and 4) when the load balancing, compute balancing and partitioning are executed. In FLAME, the agent communicates with the environment and other agents through a Message board [34]. The communication is all-to-all through message boards, and there is no direct agent-to-agent communication [34]. Agents must filter their messages on the message board using an API and manage their communications. The message board and its access by agents are key operations within FLAME [33]. D-Mason uses the publish-subscribe design pattern in order to propagate agent state information. Each worker receives relevant messages through a multicast channel [42]. Therefore, each worker must subscribe to the channels associated with the regions assigned to it. Pandora supports OpenMP and MPI APIs in its communication routines. Local communication uses OpenMP inside a given interaction range [130]. On the other hand, neighbour nodes receive border information by MPI every time a step is executed [130]. Pandora's communication solution does not deal with concurrent data changes in the border. Repast HPC automates much of its communication processes. However, the user must provide a serialization code to pack and unpack the agent information [40].

5.6.8 Model

High performance solutions require parameterizations that will be somehow translated for code. For an application area research user, the meaning of the parameters can be complicated. Therefore, this increases the complexity for the user. On the other hand, decreasing this complexity for the user means hiding the technical details that consequently will assume default values that probably will not bring the best performance results, for example: number of threads for memory shared solution, the affinity of processes, size of messages, and many others. Therefore, the bigger trade-off of ABM tools resides in modeling the facilities for users that are not experts in computing. Care HPS user must define its model by implementing and overriding some classes using the C++ language specification. Care HPS was designed with the purpose of hiding all the possible complexity of HPC techniques from the application research user. The user has to manipulate few classes without requiring the know-how of HPC. The FLAME model specification is defined in XMML, which is FLAME's agent specification language based on the XML standard [33]. The XMML must represent the model, and the user must provide information such as: the agent properties; a list of functions with current state and end state; user defined data types; possible inputs; and/or conditional statements. Each function must be implemented by the user in a C program source. A model in D-Mason is defined by using the subclass of Mason's model class called *SimState* [87]. It contains a discrete-event schedule, a random number generator, and fields. *SimState* is completely accessible by Mason's agent. Pandora has two abstract classes (*World* and *Agent*) to represent the content of any model [130]. Repast HPC does not provide any interface or extend any class to represent a model [40]. The user has to create and initialize the Repast HPC components to model their problem.

Table 5.1: Comparative table of Care HPS with other ABM tools

ABM Tool	How does it addresses ...							
	agent	environment	sync	serialization	partitioning	balancing	communication	modeling of a problem
Care HPS	Extends a class	Extends a class	Callback method	Using built-in class	Built-in or Extended	Built-in or Extended	Built-in or Extended	Extends a class
FLAME	X-machine	X-machine	Message board	X-machine	Built-in	Feature implemented indirectly through of other feature.	Message board	XMML
D-Mason	Extends a class	Built-in	Built-in	Feature implemented indirectly through of other feature.	Built-in	User must provide the solution	Publish /Subscribe	Extends a class
Pandora	Extends a class	Extends a class	Built-in or Extended	Serialization	Built-in	User must provide the solution	Built-in	Extends a class
Repast HPC	C++ class	Built-in	Built-in	Serialization	Built-in	Feature implemented indirectly through of other feature.	Built-in	C++ class

The literature has many ABM tools with different approaches, solutions, implementations and domains, and each achieves varying levels of performance and facilities for different types of users. After this comparison, we can summarize that Care HPS differs from other ABM tools in the following point:

- Care HPS was designed to be a tool to do science, for both the application area user and for the HPC expert user, with agent-based models that require high performance simulations as the main objective.

Chapter 6

Experimental results

Care HPS was designed with layers and components that enable the extension and reuse of its functionalities and code. These layers are composed of HPC, simulation and modeling techniques which were proposed and developed throughout the last few years. In the next sections, we will explain these contributions and we will present some experiments and discussions.

6.1 Assessment of *Aedes Aegypti* pupal productivity validation

In this section, we will analyze the completeness and functionality of Assessment of *Aedes Aegypti* pupal productivity model, proposed in Section 4.2.4, through its validation. Also, we will show that it is possible to use Care HPS in order to model a real problem.

6.1.1 Model validation

This model was implemented using Netlogo [163]. We chose Netlogo because it is a well-know Agent-Based programming language that enables us to analyze and study our proposed model in order to develop a first proof of concept. Moreover, Netlogo enables its users, in our case, epidemiologists, to execute its simulations easily by just adjusting its parameters. In all tests, we take the dengue virus into account and we use the same initial parameters: number of mosquitoes, percentage of infected mosquitoes, number of people, percentage of infected people, thirteen containers with pupal productivity percentages, average and standard deviation of eggs per batch and the minimum number of containers needed for the mosquitoes to lay eggs. This implementation, see Figure 6.1, offers the user important information such as: a) number of mosquitoes in each life stage; b) numbers of infected and uninfected mosquitoes; c) numbers of infected and uninfected people; d) number of pupae per person; and e) number of pupae per container. The mosquitoes are represented by yellow points (uninfected) and green points (infected). The people are represented by a blue face (uninfected) and a red face (infected). The containers are represented by colorful squares.

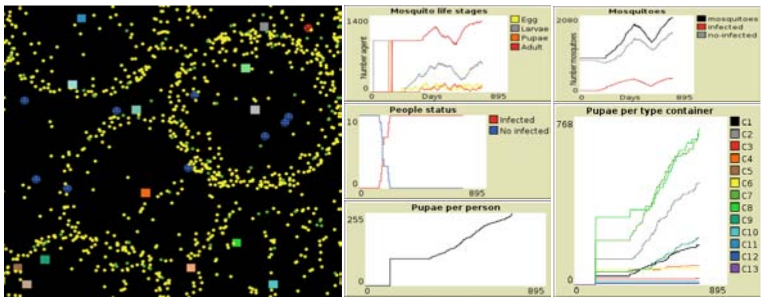


Figure 6.1: Netlogo implementation of the proposed model. This figure shows the output interface of the simulation and some important reports.

In order to verify the proposed model, we executed the simulation 1500 times, where each replication simulated 100 days and calculated the average pupae produced by each container, Figure 6.2. Then, we compared the average of the percentage of pupae per container that we obtained from the simulation with the percentage of the container productivity defined in [24]. We will use this real container productivity as a reference. In accordance with [24], the containers were inspected monthly for the occurrence of mosquito immature stages during two consecutive vector-breeding seasons in 2002-2004. The biggest difference that we found was 5.41% in the Container 8, and, in the other containers, the difference was lower than 1.32%. Therefore, we consider the results obtained to be satisfactory. We follow the output analysis defined by [37] in order to guarantee that the output results are statistically trustworthy.

In the next experimentation, we will show the number of pupae per container, see Figure 6.3. The data are plotted using the *ln* function in order to make the comparison and visualization easier. The red time curve presents the simulation without taking into account the productivity of the container, and the straight red line represents the average of pupae per container. Here, the mosquitoes lay the eggs, and the environment considers that the probability of an egg becoming a mosquito is the same for all containers. This means that 100% of eggs become adult mosquitoes. Some consequences of not taking the level of productivity and epidemiological relevance of the container into account can be seen in the graph presented in Figure 6.3. First of all, the standard deviation of the pupae per container (PPC) in this scenario is too high, as it is almost 94% of the value of the PPC average. In addition, if we compare the three curves (with and without productivity), we can see that the difference among the PPC indices is also high.

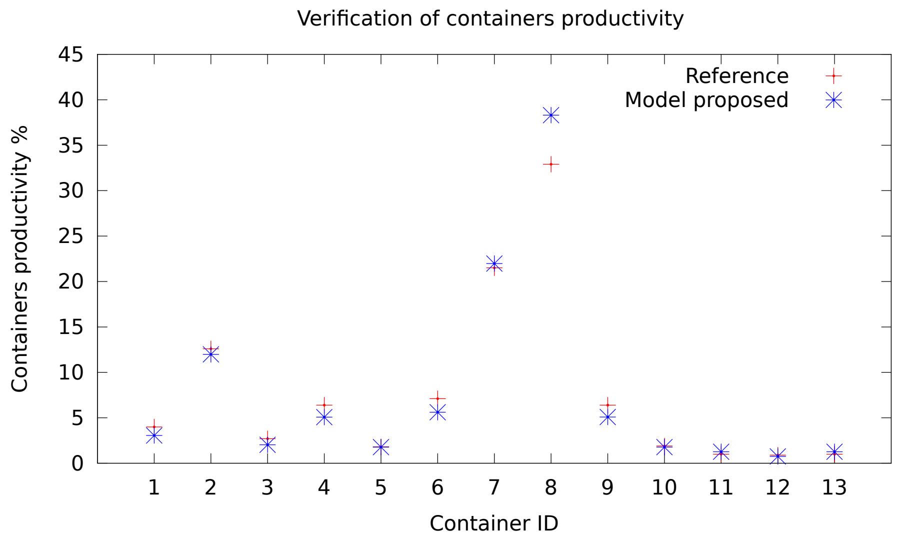


Figure 6.2: Comparative between the container pupal productivity with reference to the average of the percentage of pupae per container obtained from the proposed model.

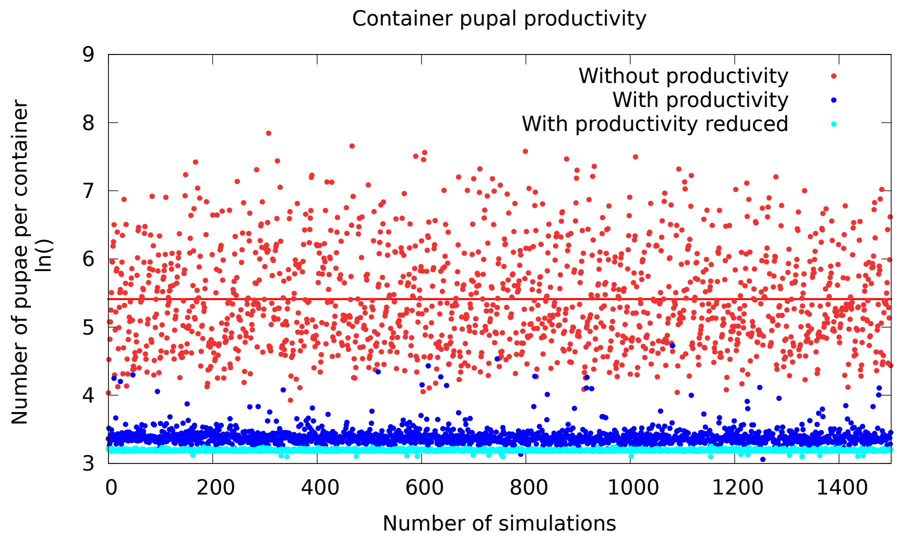


Figure 6.3: Pupal productivity in the containers.

In the other two curves, the productivity of the container is considered. The only differences are the values of probability for each container. In the dark blue curve, we use the probability of container defined by [24], and, in the light blue curve, we decrease all of the percentages of all of the containers to the value 7.69. The behavior of these curves demonstrates that the model is sensitive to the productivity's parameterization. This also demonstrates that the type of container has a strong impact on the dengue epidemic and that it collaborates with other findings that suggest that the prevention and control of the dengue epidemic should use other indices such as pupae productivity.

The model enables us to simulate different scenarios in order to support the decision made. Figure 6.4 demonstrates a hypothetical situation where a health agent wants to analyze the effects of removing all removable containers before making a decision. The red curve represents the number of infected people, considering all of the containers (in this case, thirteen) that are in the same area and the productivity of each one. The blue curve represents the simulation data, considering that the environment only has the fixed containers. As we can see, the elimination of removable containers represents a considerable decrease in the number of infected people.

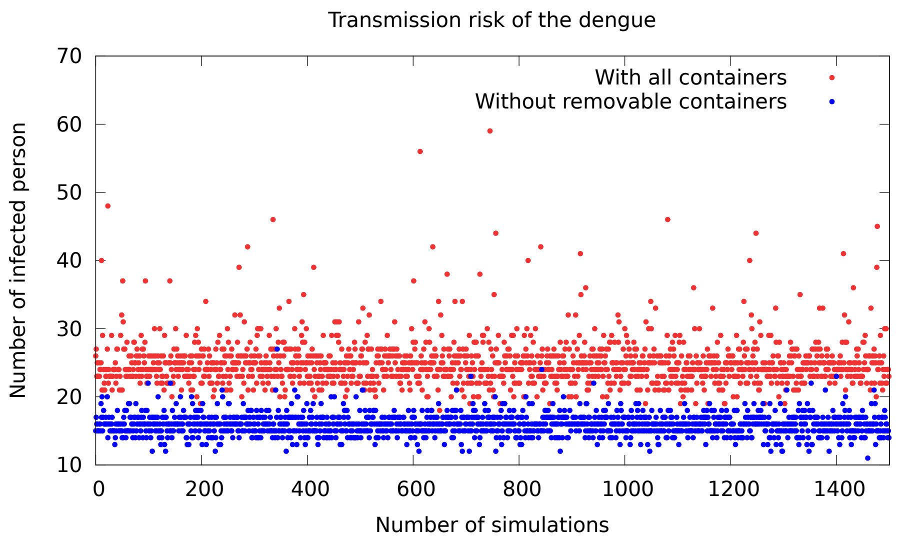


Figure 6.4: Hypothetical situation where a health agent changes the model parameters in order to simulate actions for decisions made.

The simulation also shows that emergent behavior of the mosquitoes can be observed when they fly near the containers where they were born. They create a very well-defined area of actuation. An important implication of this behavior is that the spatial distribution of the containers can influence the transmission risk of dengue. Basically, this occurs for two reasons: 1) mosquitoes have a radius of flight. All of the life cycle of a mosquito occurs inside of its flight area: laying eggs, finding blood, reproduction. Thus, its micro-level behaviour is restricted to this situation area. Therefore, the more mosquitoes, containers and people there are inside of this area, the higher the chances will be of a mosquito getting and/or transmitting dengue. And, 2) another micro-level behavior that we believe has a strong impact on dengue is the ability of mosquitoes to lay eggs in different containers. This is a natural behavior that increases the odds of perpetuating the species. Suppose we have scenario A, where a mosquito finds only a container with 5% of productivity inside of its actuation area. This means that only 5% of total eggs laid (87 eggs per batch in this simulation) will become adult mosquitoes. In scenario B, however, if another container with 20% productivity is inside the actuation area of the mosquito, then the quantity of pupal per container inside this actuation area will be greater than in scenario A, because the mosquito will lay one part of its 87 eggs per batch in one container and the other part in the other container. The chances of more adult mosquitoes inside this actuation area will be higher, as well as the chance of an incidence of dengue. Therefore, a container might have its productivity potentialized, which means provide a greater absolute number of mosquitoes for the Stratum, if it is near other containers, affecting in this way the dengue outbreak. Figure 6.5 has two areas of actuation that are indicated by two circles with green and pink contours: as we can see, the mosquitoes that were born in these containers, marked by two red arrows, might lay eggs in any container within the circled area.

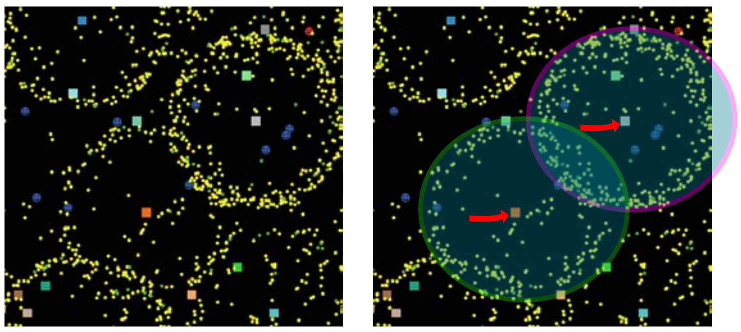


Figure 6.5: Area of the mosquito actuation.

The lack of productivity container control has a huge standard deviation. It supports some studies that cannot find a strong correlation between the BI and the transmission risk of dengue. These experimental results enable us to conclude that the proposed model can simulate adult mosquito productivity, taking the productivity of containers into account. Nevertheless, it is important to note that the productivity of the container depends on many factors and that the definition of this information is fundamental to supporting an accurate prediction. On the other hand, we believe that our model could be used in order to define several scenarios of productivity of some containers in order to give epidemiologist researchers enough information to be able to make decisions and carry out interventions.

One important observation is that, a realistic Stratum can measure many squares hectares. As an example, the study conducted by [24] covers a total area of 400.4 km^2. In our experiments, we simulate a Stratum with approximately 30600 m^2, which is too small, if we consider real dimensions. Even so, the experimentations took three hours to finish 1500 independent simulations. This type of problem is a computationally demanding one that might require a parallel and distributed solution, as already presented in other works [125], [124]. Therefore, as our objective is a more realistic simulation with statistically reliable data, a high performance computing (HPC) simulation to simulate a dengue outbreak will be used.

6.1.2 Translate the model to Care HPS

In this section, we present how to represent a model developed in other tools in Care HPS. The aims of this experiment are: (1) show Care HPS is able to represent models developed in other programming languages; and, (2) show that Care HPS can represent a real problem. Each important part of the Netlogo model will be shown with its correspondent code part in Care HPS. We will not explain all of the implementation detail, but will instead focus on the most important aspects. The full code of the Netlogo model can be find in Appendix B.

In ABM, the system is formalized using agent and environment. This model has as agents: mosquito and person. These agents interact with the environment, which is composed of the Stratum and the container.

Agents in Netlogo

The agents can be represented as Turtles. It is possible to create a "breed" of turtles and define specific behaviors for this breed [164]. Listing 6.1 shows the creation of mosquito and person agents. Also, this listing shows how to define specific properties of these agents through functions <NAME_OF_BREED>-own.

Listing 6.1: Defining agents in Netlogo

```
breed [people person]
breed [mosquitoes mosquito]

; defines properties which are common for all agents
turtles-own [
  infected? ; it indicates if the agent is infected.
  age ;it represents the lifespan of the mosquito and the person.
]

mosquitoes-own[
        ; defines properties of the mosquito
  extrinsic_incubation?
  has-to-deposit-eggs?
  batch
```

```
   .
   .
   .
  my_lifespan
]

people-own[
        ; defines properties of the person
  intrinsic_incubation?
]
```

Agents in Care HPS

In order to represent the mosquito and person, it is necessary to define the two C++ classes that inherit from the Agent superclass (see Listings 6.2 and 6.3).

Listing 6.2: Defining the mosquito agent in Care HPS

```
#include "agent.h"
// other includes

class mosquito: public agent{
protected:
        long lifespan; /** Indicates the total age of the mosquito must
            alive (6-8 weeks). */
        long incubation; /** Indicates the extrinsic incubation of
            mosquito (8-12 days). */
        bool infected; /** Indicates if the mosquito is infected with
            the dengue virus. */
        long age; /** Indicates the age of the mosquito. */
        bool is_incubation; /** Indicates if the mosquito is in
            extrinsic incubation state. */
    .
    .
        .
    // Other attributes and methods declaration
}
```

Listing 6.3: Defining the person agent in Care HPS

```
#include "agent.h"
// other includes
```

```
class person: public agent{
protected:
        long lifespan; /** Indicates how much time of the person after
            be bitten per a infected mosquito. */
        long incubation; /** Indicates the intrinsic incubation of
            mosquito (3-15 days). */
        bool infected; /** Indicates if the person is infected with the
            dengue virus. */
        long age; /** Indicates the age of the mosquito. */
        bool is_incubation; /** Indicates if the person is in intrinsic
            incubation state. */
    .
    .
        .
    // Other attributes and methods declaration
```

Simulation process in Netlogo

The simulation process is implemented at *Go* method (see Listing 6.4), with the interaction among the agents and environment taking place in this method. As we can see, the methods of the agents which define their behaviors are called in this method. In this case, the mosquito behaviors are defined in other methods: update-mosquito, move_mosquito, bite-person, deposit-eggs, as well as the person behavior: move_person.

Listing 6.4: Defining the simulation process in Netlogo

```
to go
      if (ticks = simulation-steps ) [stop]
      tick
      ask mosquitoes
        [ update-mosquito ; Update the state of mosquito
          move_mosquito
          if ( age > time_pupae_stage ) ; If the mosquito is in the
              adult stage
           [bite-person
            deposit-eggs ; After bite a person the mosquito deposit eggs
            ]
          ]
      ask people
        [move_person
          inc-age
          ]
```

```
    if print-output-file?
    [print-result]

end
```

Simulation process in Care HPS

The interaction among the agents and environment in Care HPS must be implemented in the *update_agent* method. Agent behaviors are implemented as methods of the mosquito and person classes. As we can note, in Listings 6.5 and 6.6, the algorithm implemented is the same as the one implemented in Netlogo. The differences are pertinent to language primitives.

Listing 6.5: Defining the simulation process in Care HPS. Specifying the behaviors of the mosquito agent.

```
void mosquito::update_agent(void){

        this->update_mosquito();
        this->move_mosquito();

        if (this->age > MODEL->get_time_pupae_stage()){
                bite_person();
                deposit_eggs();
        }

        this->set_alive(false);

}
```

Listing 6.6: Defining the simulation process in Care HPS. Specifying the behaviors of the person agent.

```
void person::update_agent(void){
        this->move();
        this->inc_age();
}
```

As mentioned in the previous section, all of the agents are added in a bucket and the kernel of simulation calls the *update_agent* method of each agent inserted in the bucket.

Environment in Netlogo

Stratum and container are the elements that represent the environment in this model. Stratum is represented as a World. The World is two-dimensional and is divided up into a grid of patches [164]. The containers are represented as a patch. Each patch is a square of the grid over which the turtles can move [164]. Netlogo also considers patches as agents.

Listing 6.7 presents how to define the properties of the container. Model creates one container for each container type in a random position of World.

Listing 6.7: Defining properties of container in Netlogo

```
patches-own [
  type_container ; Bromeliads=1, Ships=2, Water tanks=3, Drains=4, ...
  positive? ;it indicates if the container has eggs.
  num-egg
  num-larvae
  num_pupae
  num_adult
]
```

Environment in Care HPS

Stratum is represented as a Euclidean Space; therefore, we extend the environment class in order to represent the Stratum (see Listing 6.8). The container is represented by a new class named container (see Listing 6.9). This class inherits from the object class. It was created in order to represent specific properties of the container.

Listing 6.8: Defining Stratum in Care HPS

```
#include "environment.h"

class environment_stratum: public environment{
public:
        environment_stratum(void);
   // Other attributes and methods declaration
}
```

Listing 6.9: Defining container in Care HPS

```
#include "mosquito.h"
class container: public object{
protected:
        float a,b; /** Coordinates of circumference center point.*/
        float r; /** Radius of the circumference.*/
        bool positive; /** Indicates if the container has pupal.*/
        bool productivity_container; /** Indicates the if productivity's
            container will be considere in model.*/
        float per_productivity_container; /** Indicates the productivity
            's container.*/
        long num_pupae;
public :
        container();
   // Others methods declaration
}
```

Interaction among agents and environment in Netlogo

As we commented before, Netlogo is composed of patches over which the agents can move. In this model, the interactions among agents and environment occur when two agents are in the same patch. An example of this occurs when a mosquito bites a person. It is implemented by method *bite-person*, see Listing 6.4.

Listing 6.10: Method of mosquito: bite-person

```

to bite-person   ;; Mosquito bite a person

  if female? and not has-to-deposit-eggs? ; Just the female mosquito
      that bites the person
  [
    let p one-of people-here ;; Person is in same patch of mosquito so
        the mosquito bite the person?
    if p != nobody
    [
       ;; Execute this code if there is a person in same patch of
           mosquito
      ]
   ]
end
```

Interaction among agents and environment in Care HPS

The interactions of an agent with other agents or the environment occurs just by checking if the other agent or object of the environment is inside of its covering radius. The method *people_here* asks the environment class if there is an intersection between the two agents. This intersection code comes from environment superclass; therefore, the user does not need to implement it.

Listing 6.11: Method of mosquito: bite-person

```
void mosquito::bite_person(void){
shared_ptr<agent>* p;
        if (this->female and !this->has_to_deposit_eggs){ /** Only
           female mosquito bites the person and the mosquito bits in
           order to deposit the eggs.*/
                p = this->people_here(this); /** Check if a person is in
                    same patch of mosquito so the mosquito bite the
                    person. */
        if (p!=NULL){
            //Execute this code if there is a person inside of covering
                radius of the mosquito.
                }
        }
}
```

HPC approaches for this model

The hybrid strip partitioning is an appropriate partitioning technique for this model. There are two reasons for that: 1) the containers have arbitrary positions. Thus, the heuristics of this partitioning can find the best strip partitioning configuration. 2) there is a constant movement of agents in the environment. Thus, it is possible to use shared memory techniques to optimize the computing of the agents. In order to decrease the concentration of agents, we can use strip re-size as a load balancing algorithm. The communication of processes can be a synchronous communication. Currently, we are implementing this model in Care HPS. After preliminary results, it will be possible to analyze if these techniques fit with this model or if this model requires improvements or new approaches.

6.2 HPC techniques

One of the highlights of Care HPS is it allows the extension of new HPC techniques through the use of simple interfaces. Basically, there are the following HPC contributions:

- Communication patterns - we analyze and compare three communication strategies: synchronous and asynchronous message passing (via MPI) and bulk-synchronous parallel (BSP) using the Fish Schooling model. These three communication strategies were implemented in the communication layer of Care HPS.

- Hybrid cluster partitioning - exploring the multi-core architecture is an important issue to obtaining high performance in parallel and distributed simulations. However, the simulation features must fit on parallel programming model in order to increase the performance. We developed a hybrid MPI+OpenMP version of cluster-based partitioning. In the hybrid approach developed, we fit our simulation features in the following manner: the communication between the processes happens via message passing whereas the computing of the agents by OpenMP threads. In addition, we propose a new data structure for partitioning the fish clusters which avoid the critical section in OpenMP code. As a result, the hybrid version significantly improves the total execution time for huge quantity of agents, because it decreases both the communication and management of processes overhead, whereas it increases the utilization of cores with sharing of resources.

- Optimal run length for simulation - in scientific simulations the results generated usually come from a stochastic process. New solutions with the aim of improving these simulations have been proposed, but the problem is how to compare these solutions since the results are not deterministic. Consequently how to guarantee that the output results are statistically trusted. We apply a statistical approach in order to define the transient and steady state in distributed simulation. We used linear regression and batch method to find the optimal simulation size. We have applied and adapted the simple statistical approach in order to define the optimal simulation length. Also, we propose the approximate approach to normal distribution instead of generate replications sufficiently large. This method can be used in other kind of non-terminating science simulations where the data either have a normal distribution or can be approximated by a normal distribution.

- Strip partitioning - data partitioning is one of the main problems in parallel and distributed simulation. Distribution of data over the architecture directly influences the efficiency of the simulation. The partitioning strategy becomes a complex problem because it depends on several factors. In an ABM, for example, the partitioning is related to interactions between the agent and the environment. Therefore, parallel and distributed simulation should dynamically enable the interchange of the partitioning strategy in order to choose the most appropriate partitioning strategy for a specific context. We propose a strip partitioning strategy to a spatially dependent problem in ABM applications. This strategy avoids sharing resources, and, as a result, it decreases communication volume among the processes. In addition, we develop an objective function that calculates the best partitioning for a specific configuration and gives the

Table 6.1: Execution environment

Machine name	Configuration
Cluster Dell	512 cores. AMD Opteron 6200 1.6 GHz, L2 (2MB), L3 (6MB), 8 nodes with 64 cores each distributed in 4 sockets with three cache levels, 64 GB RAM pernode, Interconnection Gigabit Ethernet.
Cluster IBM	128 cores. 32 nodes 2 x dual-core Intel(R) Xeon(R) CPU 5160@ 3.00GHz, 12GB fully buffered DIMM, Integrated dual Gigabit Ethernet.

computing cost of each partition, allowing for a computing balance through a mapping policy. The results obtained are supported by statistical analysis and experimentation with an Ant Colony application. We developed a solution where the partitioning strategy can be chosen dynamically and always returns the lowest total execution time.

- Hybrid strip partitioning - Into a spatially dependent problem can occur a concentration of agents in some parts of the environment. When it occurs few cores can be manipulating many agents while other are idle. In order to minimize this problem, we developed a hybrid version of the Strip partitioning. The algorithm detects when there are a high number of agent into a strip and create OpenMP threads in order to execute these agents.

All of the experimentations presented in this section were carried out using one of the environments described in Table 6.1.

6.2.1 Communication patterns

This section shows the experiments that analyze communication of three communication strategies implemented and presented in Section 4.3.1: asynchronous and synchronous message passing and bulk-synchronous parallel.

The execution environment used to testing this contribution was the Cluster IBM (see Table 6.1). Experimental results were obtained from the average of 250 simulation steps. The code was compiled using C++ (gcc 4.3.2), STL (c++ standard template library), MPI namespace (openmpi 1.4.1) and BSPonMPI (v0.2). The simulation experiments were carried out by using 1, 2, 4, 8, 16, 32 and 64 cores. The data input were formed by three files containing 131.072, 262.144 and 524.288 agents. These input data were synthetically generated using a uniform distribution and filtered by a function that gives them a spatial distribution of a fish school. This adaptation is necessary to adapt the input and obtain data group cohesion in all experiments. In relation to the output data and validation between the three algorithms, we have used a replication method to generate output data and relative values of the magnitudes under study.

Dynamic load balancing strategy is invoked depending on the following load imbalance thresholds:10%, 20%, 30%, 40% and 50%. Experimental results help us to choose which is the best configuration (load imbalance threshold) that we will use for analyzing communication strategies. The dynamic load balancing algorithm used in this work was implemented in [146]. This algorithm defines T as the load imbalance threshold. The number of agents per core is bounded by:

$$MIN = MEAN * (1 - T) \tag{6.1}$$

and

$$MAX = MEAN * (1 + T) \tag{6.2}$$

, where

$$MEAN = Ni/Nc(numberofagents/numberofcores) \tag{6.3}$$

If the number of agents stored in a specific core is out of bounds, then the load balance algorithm is invoked. Therefore, when we say load imbalance threshold is equal to 30% it means $T = 30\%$. We have implemented three different communication layers: asynchronous and synchronous message passing (via MPI) and bulk-synchronous parallel.

We can see in Figures 6.6 and 6.7, there are variations in terms of communication times for the same communication strategy by means of using different load imbalance thresholds. In the following analysis we have considered a 10% as load imbalance threshold. Figures 6.6 and 6.7 show that the asynchronous communication strategy has the best performance in terms of communication times in comparison with other.

In Figure 6.8, we can not observe variation in term of speedup for all communication strategies by using from 2 to 8 cores. On the other hand, there are more significant speedup variations in terms of communication times for all communication strategies by using 32 and 64 cores. These variations are a result of the amount of communication between processes. Furthermore, the efficiency and behavior of each communication strategies are more evident. In the asynchronous communication strategy almost all processors have higher speedup. Thus, interesting behaviors can be seen in the BSP communication strategy. We can verify that the BSP communication strategy tends to increase the speedup by using 32 and 64 cores with workloads of 262.144 and 524.288 agents. Consequently, we believe that if we increase the number of agents by using 32 and 64 cores the BSP communication strategy will be able

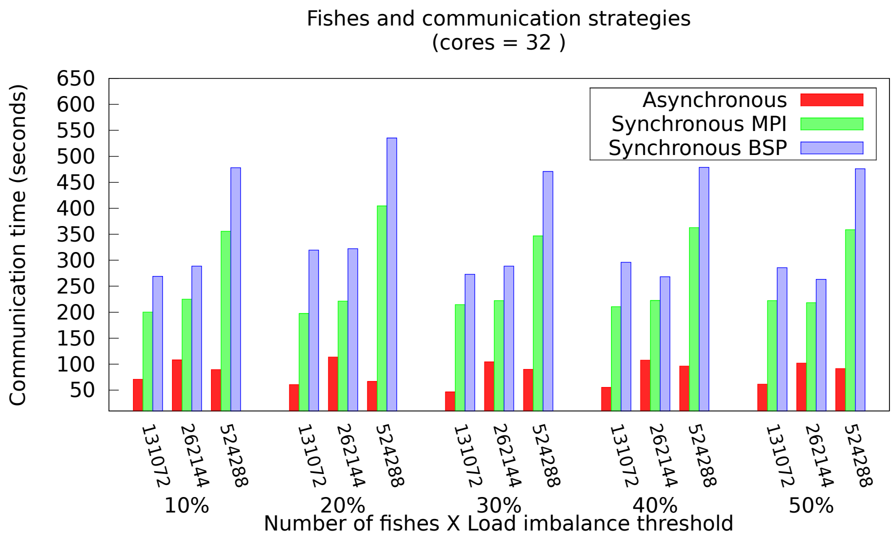


Figure 6.6: Communication time: load balance x number of fishes x communication strategies (32 cores)

to give us an acceptable total execution times. Additionally, when the simulation is performed with 524.288 agents, the BSP gain in terms of speedup is similar to the asynchronous speedup by using 32 cores. Moreover, Figure 6.8 confirms that our distributed solution is scalable because the speedups of the communication strategies: asynchronous, synchronous BSP, and synchronous MPI were, respectively, 23.54, 21.60 and 20.79 by means of using 64 cores and 523.288 agents. Besides that we can see the scalability into BSP strategy by using 64 cores with workloads of 131.072, 262.144 and 523.288 agents, its speedups were, respectively 12.39, 16.30 and 21.60.

The Figures 6.9, 6.10 and 6.11 show the computing time and communication time of each communication strategies by using 16, 32 and 64 cores with workloads of 131.072, 262.144 and 524.288 agents and load imbalance factor of 10%. It is easy to understand the gain in terms of speedup when we analyze the relation between computing and communication, see Figures 6.9, 6.10 and 6.11. We have developed a distributed solution in order to solve large-scale models. It has no sense analyzing communication strategies when we are using a reduced number of cores because it does not give us relevant information. Therefore, we only will show experimental results by using 16, 32 and 64 cores. Experimental results show that the computing time obtained by using a asynchronous communication strategy is higher than the other communication strategies. If this it occurs, then the synchronous message passing and BSP strategies would be a viable solution if we could decrease communication time of these strategies. In asynchronous communication strategy we can observe that computing times tend to increase whereas communication times tend to decrease. On the other hand,

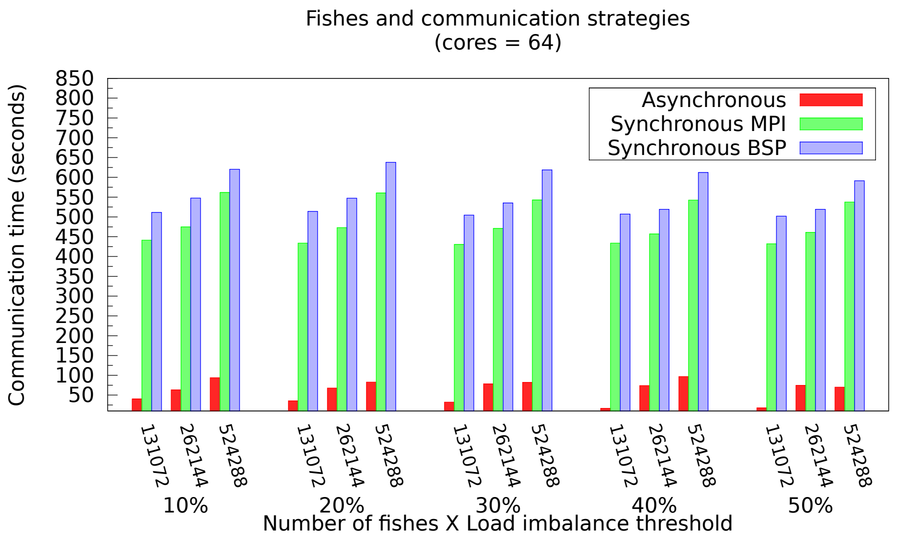


Figure 6.7: Communication time: load balance x number of fishes x communication strategies (64 cores)

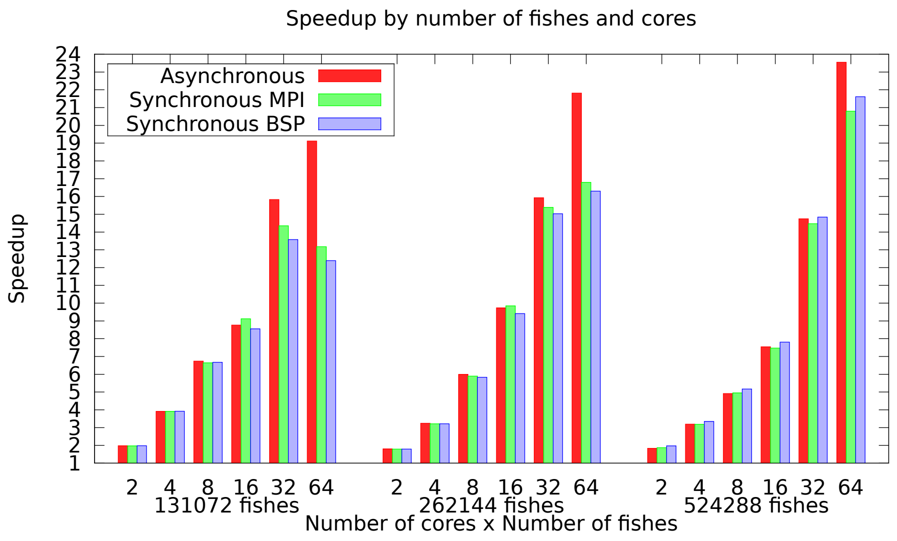


Figure 6.8: Speedup of each communication strategy in different contexts. Each data set that has different number of fishes was executed in varied number of cores.

the computing times of the synchronous message passing and the BSP strategies tends to stabilize whereas communication times tend to increase as the number of fish and cores are incremented.

Clearly, in Figures 6.9, 6.10 and 6.11 we can see that large-scale simulations require higher communication usage. Both figures show that as the number of agents is increased, communication times are more expensive. As example, the communication time of the

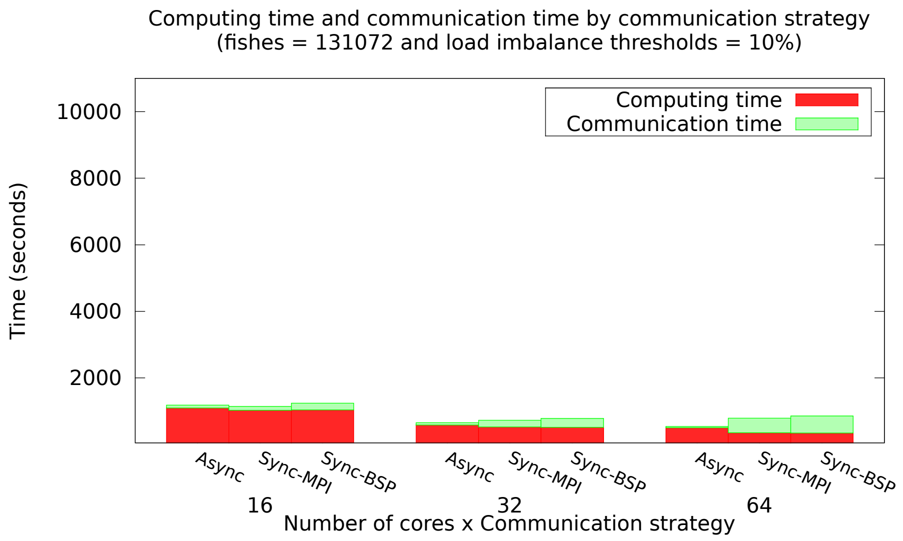


Figure 6.9: Computing time and communication time x communication strategies (131.072 fishes and load balance factor 10%).

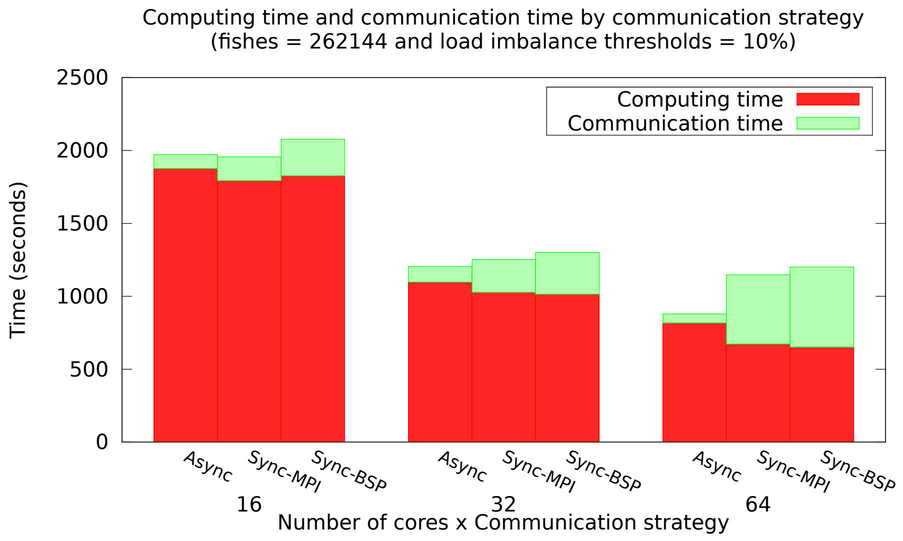


Figure 6.10: Computing time and communication time x communication strategies (262.144 fishes and load balance factor 10%).

synchronous message passing strategy by using 64 cores with workloads of 131.072, 262.144 and 523.288 agents were 441.38, 474.76 and 561.56 seconds respectively.

Figures 6.9, 6.10 and 6.11 show how the communication strategy impacts on the global simulation performance. The synchronous communication strategies decrease the computing time. We can think that synchronous communication strategies are the solution to decrease the total simulation time. But, in view of complexity inside logical process what it is gained

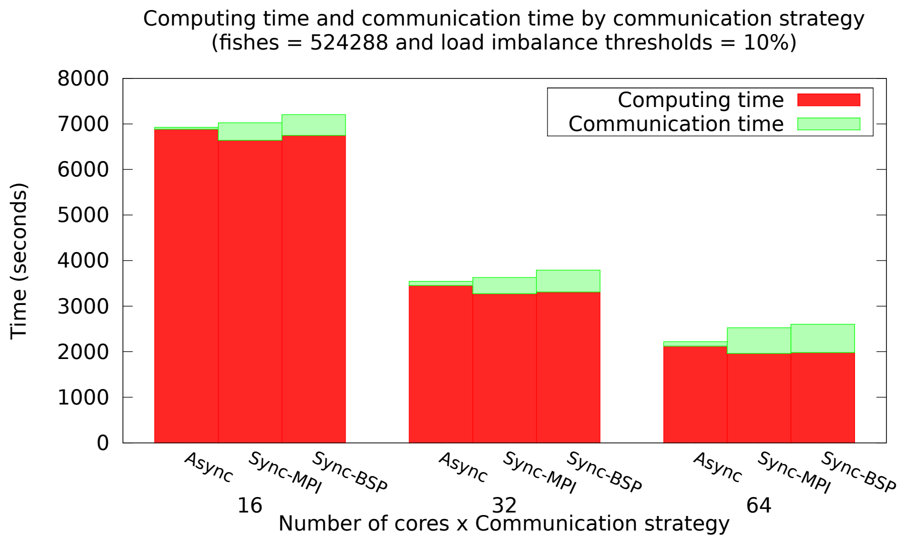


Figure 6.11: Computing time and communication time x communication strategies (524.288 fishes and load balance factor 10%).

in computing is lost in communication. The computing time in asynchronous communication strategy is the higher in comparison with other communication strategies. On the other hand, the cost in terms of communication time is the lowest. In synchronous communication strategies we have a set of communication barriers. It could suggests that synchronous communication strategies show a good performance and the asynchronous communication strategy could be discarded. Indeed the synchronous communication strategies reduce the computing time in comparison to asynchronous strategy, however what it gains in computing time is what it loses in communication time because the simulation is very heterogeneous and complex. Our simulation has a high complexity, so it prejudice the efficiency of synchronous communication strategies and the gained performance. Despite that the time-driven simulation has a synchronous behavior the synchronous communication strategies shown results are not the most appropriate.

In other words, the synchronous communication strategies have a better performance in terms of computing times, but higher communication times in comparison to asynchronous communication strategies. As a consequence of the previously described, the asynchronous communication strategy has a better performance in terms of total execution times in comparison to synchronous message passing and BSP. The improvements to the asynchronous method, after in-depth analysis of the code, come from the waiting-time reduction that occurs in those processes that are highly balanced in workload and belong to the same meta-cluster.

In this experimentation, we have analyzed and compared three communication strategies: asynchronous and synchronous message passing (via MPI) and bulk-synchronous parallel. The main conclusions that can be extracted are:

- We showed that time-driven simulations do not always increase the performance by using synchronous communication strategies. Many researchers believe that the synchronous communication strategies are more appropriate for time-driven simulations because it has a synchronous behavior. The results show that synchronous communication has worse performance in terms of execution times in comparison to asynchronous communication strategy. Despite that the time-driven simulation has a synchronous behavior we have shown that synchronous communication strategy is not the most appropriate. Considering that, time-driven simulations by using conservative time-management algorithms have a barrier in each simulation step it could be suggested that synchronous MPI and BSP would be more efficient. If all of the cores did the same quantity of computing the synchronous communication strategies would be more efficient. Because the time-slice between send and receive would be lower.
- In addition we have observed that asynchronous communication strategy increases the computing time because of the management communication protocol. On the other hand, we can observe that the communication time decrease. However the total execution time in asynchronous communication strategy is lower than the total execution time with the synchronous communication strategies. Therefore we can observe the efficiency of the asynchronous communication strategy in this type of simulations.
- We could verify that bulk-synchronous parallel strategy tends to gain speedup when resources, such as: number of cores and number of fishes are increased. This strategy have shown to be scalable.

6.2.2 Hybrid cluster-based partitioning

This section presents the results of hybrid cluster-based partitioning proposed in Section 4.3.2. We used the Cluster Dell (see Table 6.1) as execution environment. The two partitioning algorithms were developed by using C++ (gcc 4.3.2), STL (C++ standard template library), MPI namespace (openmpi 1.4.3). For the MPI version experiments, each MPI process was created per core. For MPI+OpenMP version experiments, we have two scenarios where each MPI process creates 8 and 16 OpenMP threads. In addition the number of MPI process created is indicated by: $total\ number\ of\ cores$ used divided by $number\ of\ threads$.

Architecture details		MPI experiments
Cores used	Node	MPI processes
32	1	32
64	1	64
128	2	128
256	4	256
512	8	512

Architecture details		MPI+OpenMP experiments			
		Scenario 1		Scenario 2	
Cores used	Node	MPI processes	OpenMP threads	MPI processes	OpenMP threads
32	1	2	16	4	8
64	1	4	16	8	8
128	2	8	16	16	8
256	4	16	16	32	8
512	8	32	16	64	8

Table 6.2: The Architecture details column contains information about number of cores used and the number of nodes. MPI Processes column indicates how many MPI processes were created. OpenMP Threads column indicates how many OpenMP threads were created by MPI process.

In order to analyze both versions, we compare the total execution time taking in consideration the total number of cores used on experiment. The details of experiments can be seen in Table 6.2. The statistical results are guaranteed by using the batch replication techniques described in [37]. In this first experiment, shown in Figure 6.12, we keep a constant number of agents and we vary the number of cores of both the MPI version and the MPI+OpenMP version, using 8 and 16 threads in the latter. This experiment is a proof of concept which aim is to analyze the behavior of both versions with different number of cores. In addition, we verify the influence of the number of MPI processes and threads on the hybrid version.

The MPI version has better results than the hybrid version up to 64 cores. There are no inter-node communication on 32 and 64 cores because these cores lie in the same node. Therefore, the overhead of OpenMP data structure's consolidation does not compensates the MPI communication time. This version scales very well up to 128 cores but it suffers with increasing the number of MPI processes as consequence of the broadcast communication.

This experiment (Figure 6.12) shows that the performance of hybrid version depends on the both number of MPI processes and OpenMP threads. The two MPI+OpenMP approaches have different execution times because their workloads are distinct. Thus, we have to watch out the quantity of agents inside the OpenMP data structure because it might become a

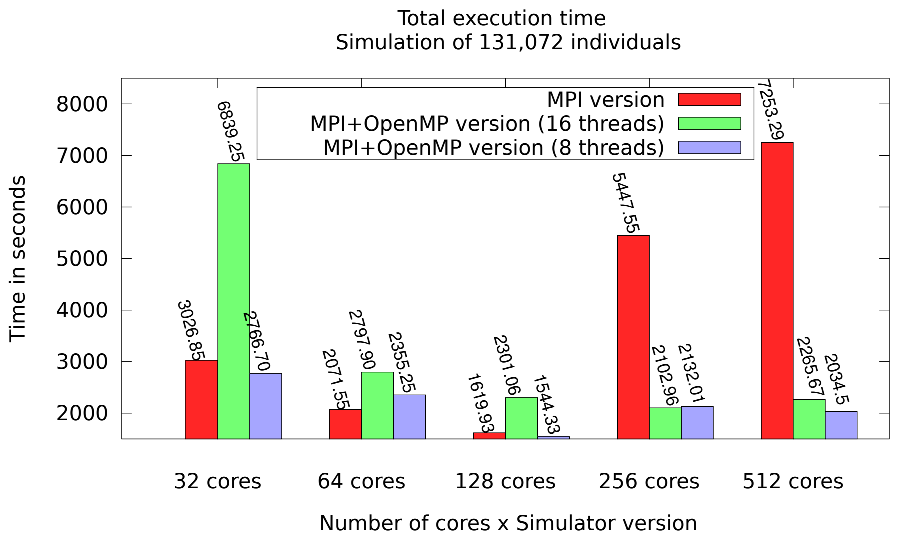


Figure 6.12: Total execution time comparison between the MPI and the MPI+OpenMP versions.

bottleneck. Since the consolidation process of the new list of cluster must copy many agents between the data structures.

Using more than 128 cores the hybrid version has better results. In this version, we have decreased the number of MPI process consequently the quantity of agents grew inside the process. In other words, each MPI process manipulates and controls more agents. This implies in decreasing the MPI broadcast communication between the processes in order to notify others process about the agents behaviors. Consequently, the hybrid version provides more intra-node communication rather than inter-node communication. Since intra-node communication is a memory copy among MPI processes it means less MPI communication overhead. On the other hand, the inter-node communication consumes memory bandwidth and slow down the communication because it requires an extra buffer.

In order to analyze the scalability of the hybrid version, we increase the quantity of agents to 262,144. As we can see in Figure 6.13, the hybrid version scale very well.

We are interested in realistic and complex simulations. Therefore, in the last experiment, we simulate 131,072, 262,144 and 524,288 agents in both versions by using 512 cores. As we can see in Figure 6.14, the hybrid version has better total execution time than the MPI version. In this experiment, we are using 512 cores and increasing the number of agents. The hybrid version has more agents together in same node thus these agents can read directly the memory through OpenMP threads and avoiding the memory copy among MPI processes. The MPI+OpenMP version reduces the memory requirement overhead from multiple processes.

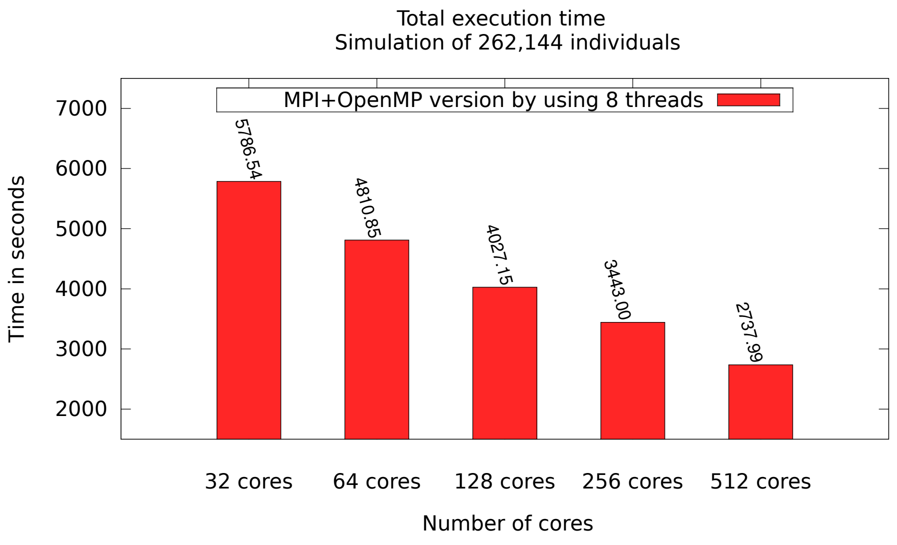


Figure 6.13: Scalability of the hybrid version by using 8 threads per MPI process.

We can observe that the hybrid version is a feasible solution for a high number of agents and cores, since the number of MPI processes and threads could be adjusted.

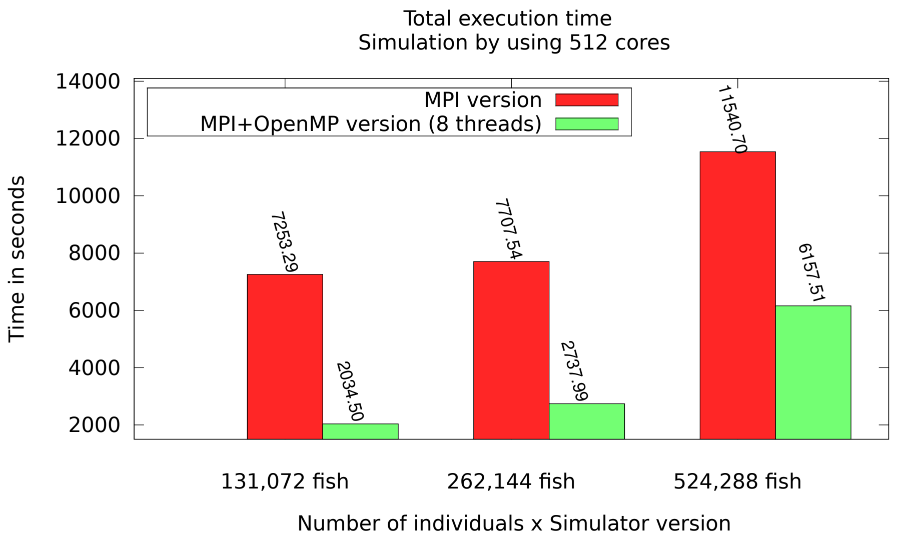


Figure 6.14: Total execution time comparison between the MPI and the MPI+OpenMP versions by using 512 cores.

In this contribution we have implemented a hybrid MPI+OpenMP version of a cluster-based partitioning. This solution was modeled by using the agent-based model where each agent interacts with another and a fine-grain parallelism can be applied. The hybrid parallelization of existing agent-based MPI implementation may bring excellent results depending on the communication pattern among the MPI processes; partitioning of agents through the architecture distributed and shared; computing between these agents; and the relation between the number of MPI processes and its OpenMP threads. Generally the data structure for distributed memory are not appropriate for shared memory because it has to be designed for simultaneous access and free of critical sections. Critical sections are main bottlenecks of OpenMP codes. In order to treat this problem, we create a new data structure to manipulate the data inside the OpenMP section. This new data structure is more efficient because it distributes evenly clusters among the threads enabling simultaneous operations over the agents without using critical sections. We believe that the data structure proposed can be used in other kind of scientific applications which have a similar data partitioning and have fine-grain parallelism code. In addition, we verified that replacing message passing communication with synchronized thread-level memory access is an interesting approach for applications which has intensive broadcast communication process. In our experiments, we reduce the total execution time of the hybrid version in comparison with the previous MPI version. We observed that the hybrid version is 3.56, 2.81 and 1.87 times better than the MPI version simulating 131,072, 262,144, and 524,288 agents by using 512 cores, respectively. Summarizing, the main contributions of these experiments are:

- We have developed a hybrid version that has better results than the MPI version for high number of cores and agents.

- In addition, a new data structure of clusters which avoids the critical section in OpenMP code was proposed (see Figure 5.5).

- Finally, we fit the features of our model with the underlying architecture which support the reduction of total execution time of simulation in the hybrid MPI+OpenMP version.

6.2.3 Optimal run length for simulations

This section presents the results of applying and adapting the statistical method based on [37] that was presented in Section 4.3.5. As presented before, this method is divided into three parts. Therefore, the results are presented following these steps.

This method was experimentally applied using 8192 agents running in four cores. The output variable of interest used in this method is the number of clusters that are created by cluster-based partitioning algorithm. In Figure 6.15 we can observe the behavior of cluster-based partitioning algorithm. Throughout the execution, the algorithm goes creating and deleting the clusters. The variation in number of clusters occurs because the agents constantly change their position. We can verify clearly two phase: transient state and steady state.

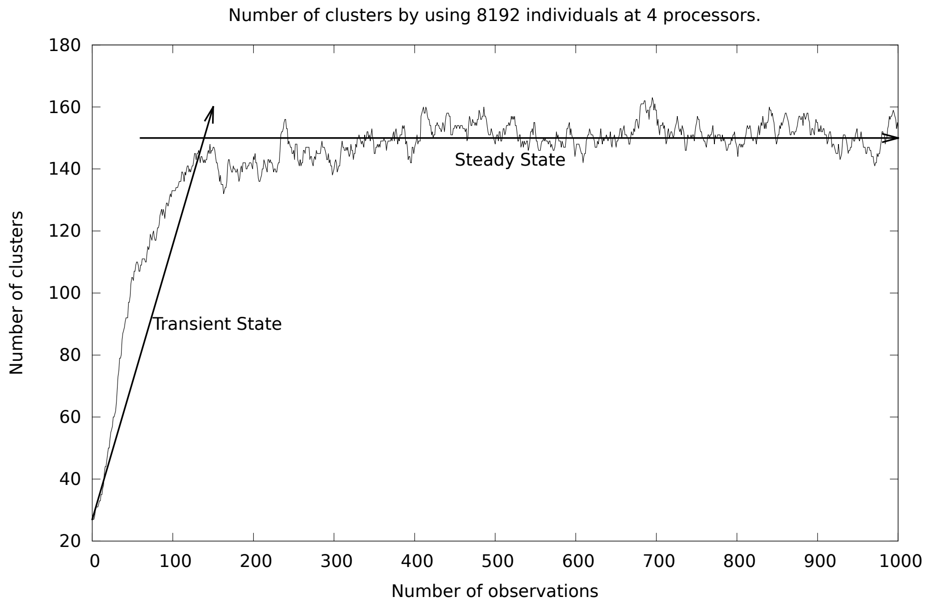


Figure 6.15: Transient and Steady State

As we can see, in the transient state, a sharp increase in the number of clusters occurs. It happens because the cluster-based partitioning algorithm verify that the number of clusters is not enough to represent all agents inside of the Voronoi diagram. In addition, the agents are uniformly distributed in space. Consequently, the simulation spends some steps trying to converge. In this phase, occurs the warm-up of the simulation. As suggested by [37], we use the linear regression approach to identify the transient state. The range of observations must be advances if the slope is not zero or insignificant. We consider 30 observations for each range and we increment always by 5 observations forward ever that the linear regression does not find the approximate end transient state.

The transient state finishes when the slope is equal to zero. The Table 6.3 presents the *P-value* obtained for some ranges. In the range 126-155, the *P-value* is more than α (0.05) this imply that the coefficient is near of zero. Therefore we can assume that after the 126 steps of simulation the transient state is finished.

The cluster-based partitioning algorithm has a strong correlation in each step simulation. Therefore, the this algorithm passes the current partitioning state to next step simulation. In this way, we have to identify the autocorrelation in order to estimate the variance.

For that, as presented in Section 4.3.5, we have to use the batch method to find the autocorrelation. We use the correlogram diagram by using 300 as observation size in order

Range	Coefficient	P-value
1-30	1.36059	2.23263e-23
6-35	1.65911	3.32428e-24
11-40	1.90394	8.23758e-26
...	...	...
111-140	0.242365	3.50807e-07
116-145	0.174877	6.4216e-05
121-150	0.162562	0.000136116
126-155	**0.0600985**	**0.103787**
131-160	-0.118227	0.0540278
136-165	-0.373892	2.46027e-05
...	...	...
996-1025	-0.9	0.107547

Table 6.3: Linear regression for transient state

to define lag size. As we can see in Figure 6.16, the non significant lag occurs at 223th observation where the correlation is equal to zero.

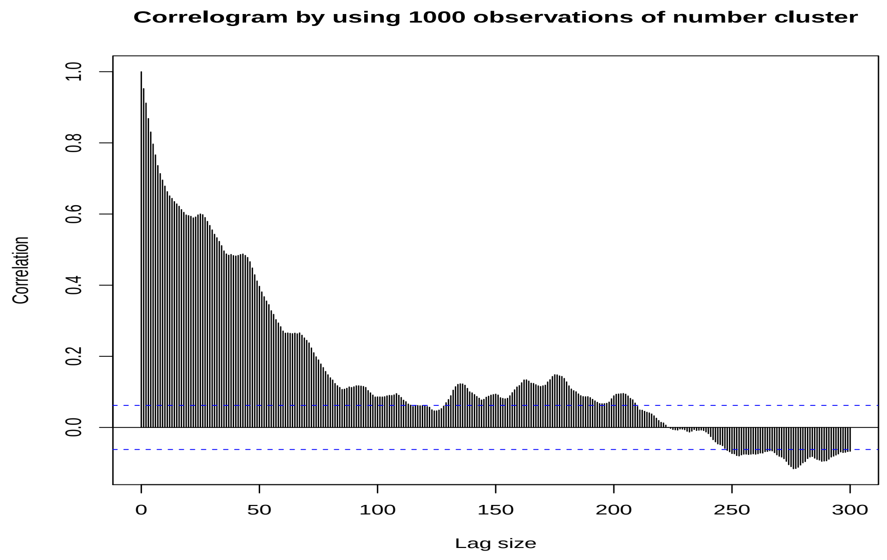


Figure 6.16: Correlogram

The next step is to figure out the correlation. These calculations are obtained using the equations 4.9, 4.10, 4.11 and 4.12 as presented in Section 4.3.5. Finally, we calculate the run length by using the equation 4.13. The Table 6.4 summarizes these calculations.

The run length found is approximately 22426 observations. We have to discard the transient state and the rest must be divided by the number of batch size, in this case 2230. Then, these batch are the individual replications that should be used to output analyses. Now,

Variable	Value
Total time execution	490.728017 s
Number of observations	1000
Nonsignificant lag size	223 observations
Batch Size	2230 observations
Single observation time	0.490728 s
Batch Time	1094.323477 s
Transient phase	126 observations
Length of run	**11005 s**
Number of simulation steps	22426

Table 6.4: Calculations of run length

we have to verify if these replications are statistically significant using ANOVA and Duncan multiple-range test. We have to ensure that the number of replication is enough. Then, we have to know the relative precision using Equation 4.15. And, we have to find out the required number of replications to specific level of relative precision using Equation 4.16. The Table 6.5 shows the results after apply these equations.

Variable	Value
s	6.105042
t	2.262157
Standard Error	4.367284
Relative Precision	0.026234
Required number of replication	0.849683

Table 6.5: Statistical summary of replications

As we can observe, in the Table 6.5, the Relative Precision is less than 0.10. It means that the current number of replications (10) is sufficient. In addition, in this experiment the number of replication required points out we need of one replication to obtain the relative precision wished. Since we have the enough replication number, then we have to use ANOVA to verify which means are different.

Our replications have binomial distribution, therefore we applied Equation 4.17 for each number of cluster of each replications in order to approximate to normal distribution.

In Table 6.6, the P-value is less than 0.05, thus we can conclude that the means are different. We can reject the hypothesis that all the replications have identical means. However, this does not mean that every average differs with every average. We will use the Duncan test to identify where the differences are. The Duncan test results are show in Table 6.7.

In Table 6.7, means with the same letter are not significantly different. Consequently the replications with the same letter cannot be used together to do output analysis. Because those

Groups	Count	Sum	Average	Variance		
B1	2230	211.509	0.09485	1.00444		
B2	2230	153.274	0.06873	1.00289		
B3	2230	84.1697	0.03774	1.00178		
B4	2230	413.192	0.18529	1.00553		
B5	2230	247.434	0.11096	1.00396		
B6	2230	52.6732	0.02362	1.00134		
B7	2230	205.19	0.09201	1.00329		
B8	2230	177.144	0.07944	1.00289		
B9	2230	329.878	0.14793	1.00460		
B10	2230	109.617	0.04916	1.00187		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F critical
Between Groups	49.6412	9	5.51569	5.49777	1.4E-07	1.8803
Within Groups	22362.7	22290	1.00326			
Total	22412.3	22299				

Table 6.6: ANOVA result

replications must be significantly different for representing an overall population. In Table 6.5 the *Required number of replication* points out that about one replication is sufficient in this experiment. Therefore, we could choose the B1 replication for simulation result analysis. For this reason, the simulation does not need to execute more steps in order to obtain other replications.

But what we should do if the *required number of replication* was three, for example? In this case, we need six replications. Because, as we can observe in this experiment B1, B4 and B6 replications are significantly different. However the replications B2, B3 and B5 must be discarded because they are equal to B1. Therefore, in this hypothetical situation,

Groups	Treatments	Means
a	B4	0.18530
ab	B9	0.14790
bc	B5	0.11100
bcd	B1	0.09485
bcd	B7	0.09201
cde	B8	0.07944
cde	B2	0.06873
cde	B10	0.04916
de	B3	0.03774
e	B6	0.02362

Table 6.7: Duncan test result

the experiment must execute 13506 observations ($(6 * batch\_size) + transient\_phase$) to get reliable results. Suppose another situation where the required number of replication is four. More replications would be necessary. Because replications B7, B8, B9 and B10 are not significantly different. As a result, quantity of the number of replications in simulation depends on if the replications are significantly different. Consequently, the total run length of simulation can be variable.

The run length, number of replication and steady state of simulation should be well specified in order to produce results that must be statistically reliable. In addition, the statistical methods have to give optimal results when we are treating with distributed realistic simulation. In this contribution we have applied and adapted the simple statistical approach proposed by Chung[37] for distributed cluster-based simulations. We propose the approximate approach to normal distribution instead of generate sufficiently large replications. The aim is to decrease the total simulation time but keeping the statistical confidence for output analysis. We observed that the run length depends on if the means represent statistically the overall population. The means that are not significantly different must be discarded and other replications must be used instead of them. In this way we can obtained experiments that are statistically trusted.

The main conclusions that can be extracted are:

- We have applied and adapted the simple statistical approach in order to define the optimal simulation length. We avoid the simulation runs steps of simulations beyond what is needed. Consequently, it save computational resources.
- In addition, we propose the approximate approach to normal distribution instead of generate sufficiently large replications. The method can be used in other kind of non-terminating science simulations where the distribution of data is normal or it can be approximated by a normal distribution.

6.2.4 Strip partitioning

This section presents the experiments of the strip partitioning algorithm proposed in Section 4.3.3. In these experiments, a statistical analysis of the partitioning is required, with the aim of analyzing which strategies have statistically different results. This enables us to effectively identify which strategies have different behaviors. We follow the output analysis defined by Chung[37] for simulations of type terminating.

A brief summary of the statistical steps followed: a) calculate the mean and standard deviation of the ten replication means; b) calculate the standard error; c) calculate the relative precision; d) apply ANOVA in order to find out if the solutions are statistically different;

and e) apply the Duncan test in order to find out which solutions are statistically different. This statistical study was executed at 64 cores using the same environment configuration in all tests. Fig 6.17 presents the average of the total execution time obtained from the ten replications of the statistical analysis. The histogram shows the horizontal and vertical approaches and the results of all of the strategies combined with the mappings. In addition, the graph has the standard deviation returned by the objective function of each partitioning strategy. We can see that all the results of the vertical approach are better than the horizontal approach; and we observe that, for the same strategy and mapping, lower standard deviation results in better total execution time. In addition, as shown in the graph, M2 mapping presents much better results than M1, mainly when the partition is in a vertical direction.

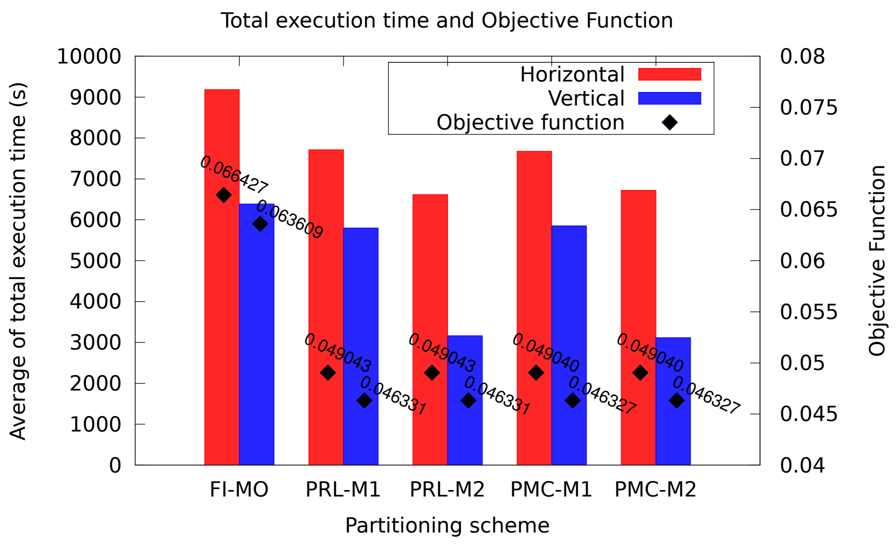


Figure 6.17: Total execution time average and objective function of the partitioning strategies

The statistical analysis says that the strategies are statistically different. This indicates that the partitioning solutions have different effects. However, we need to know which ones are different. Therefore, we have to perform the Duncan test [37]. This test shows that the pairs of strategies: (H-PRL-M1, H-PMC-M1), (V-PRL-M1, V-PMC-M1) and (V-PRL-M2, V-PMC-M2) are not statistically different. In other words, these pair strategies have no significant result differences, see Figure 6.18. This test also shows, that there are two outside values of H-PRL-M2 that make this strategy statistically different from H-PMC-M2.

We are interested in analysing the behaviour of strip partitioning, so we will use the TAU tool[139] in order to analyse the overall communication volume and its pattern in the partitioning algorithm for the worst (H-FI-M0) and best (V-PMC-M2) total execution times that we obtained in the previous experiment. The TAU provides users with the

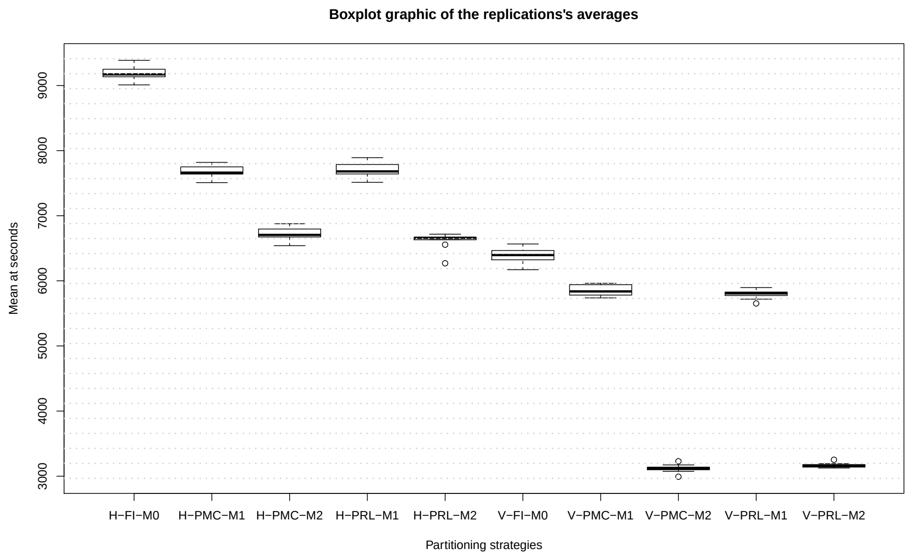


Figure 6.18: Duncan test for partitioning strategies

Communication Matrix graph (Figure 6.19), which gives information about the communication volume, such as Total Volume Bytes. The information is shown through heat maps. In Figure 6.19 (a1) and (b1), the y and x axes depict the sender and receiver processes, respectively. Figure 6.19 (a1) and (a2) represents the same information. However, (a2) shows the data in 3D in order to make the visualization of the results easier. The same process is applied for (b1) and (b2). The main diagonal and all black points have no volume byte information, because there is no process communication in these cores.

Figure 6.19 presents the exchange of total volume bytes among the processes by using the worst (a1 and a2) and best (b1 and b2) strategies. The two colorful diagonals (a1 and b1) represent the data sent by a specific core to its neighbors. The top blue line represents the initial data sent by rank zero for all of the cores at the start of the simulation. The V-PMC-M2 strategy has a better partitioning of data in the cores. The heat maps show that more cores receive more data in the V-PMC-M2 strategy than in the H-FI-M0 strategy. In addition, the H-FI-M0 strategy has more total volume data(4.11E6); therefore, the migration processes of agents from one partition to another are higher. The volume of data of the H-FI-M0 strategy is concentrated in a few cores. As our simulation is of a conservative synchronisation type [58], this results in a synchronization point among the processes, and a bottleneck is created when a few cores receive a much higher load than the other cores because these cores will take longer to finish their work [20]. In both strategies, we can observe that the volume of communication is concentrated in the central cores. The volume of communication is too low (6.47E4 of H-FI-M0 and 4.27E4 of V-PMC-M2) in the processes that have an MPI rank

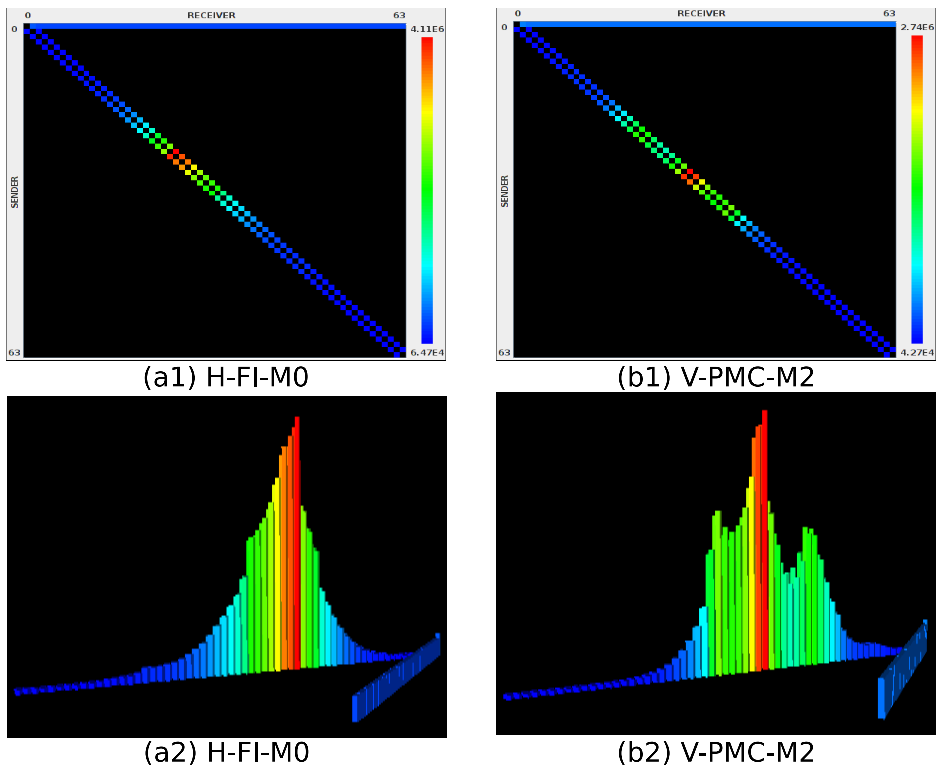


Figure 6.19: Comparison of total volume bytes with heat maps

at the start or end. This pattern occurs as a consequence of the observed environment layout (Figure 4.11(a)): the strips with more objects tend to have more agents. As the mapping starts to allocate the strips from left to right (vertical approach) and from bottom to top(horizontal approach), the objects tend to get allocated to the processes in the middle of the range. This information reveals that an additional load balance strategy might be applied in order to prevent this behaviour.

Communication and computing time, presented in Figure 6.20, also were obtained from the TAU. In spite of the overhead created by the dynamic instrumentation, we can use this data communication and computing to compare the strategies, since the overhead for both measures is constant. We consider this communication and computation with the TAU measures to be more reliable. The communication in the H-FI-M0 strategy is higher than the V-PMC-M2 strategy communication. We observe high communication volume with relation to computing, even in the best approach. We checked that the MPI_reduce messages were the cause of this communication overhead after analysing the TAU profile. This occurs as a consequence of the conservative protocol used in our simulation.

The partitioning of data is the key to improving the total execution time for parallel and distributed simulations. In this contribution, we have proposed a strip partitioning strategy to

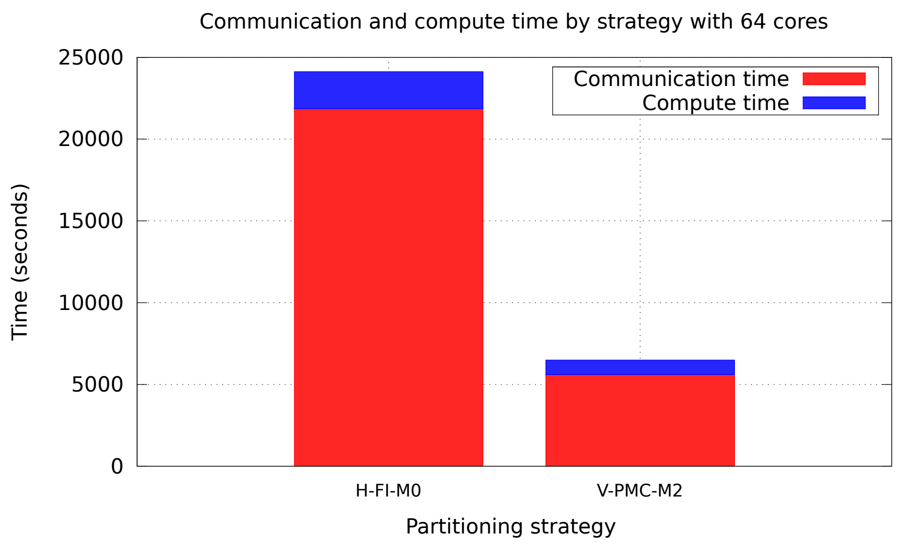


Figure 6.20: Communication and computing time of the worst and best strategies

a spatially dependent problem in agent-based model applications. The partitioning algorithm avoids sharing objects among the strips. As a result, we have observed a decrease in the communication volume and synchronization among the processes. Another important feature of this strategy is that we can find the best partitioning for a specific configuration by using the objective function proposed. In addition, this function returns the computing cost of each partition and allows for a computing balance through a mapping policy. These results were supported by statistical analysis and experimentation with an Ant Colony ABM application. We developed a solution where the partitioning strategy can be chosen dynamically, always returning the lowest total execution time in multi-core architecture. The scalability of the strip partitioning is presented in Subsection 6.4.

6.2.5 Hybrid Strip partitioning

This section presents the experiments of the hybrid strip partitioning algorithm proposed in Section 4.3.4. The hybrid strip partitioning creates threads dynamically in order to compensate the idle cores when there are concentration of agents in specific parts of the environment. These experiments were carried out for 1000, 2500, 5000 and 10000 agents in 64 cores for the pure MPI strip partitioning and hybrid strip partitioning. For hybrid partitioning, *n* threads were created between the intervals [2..8] for each MPI process; and the number of threads created depended on the load of MPI processes following the formulas defined in 5.5.2.

Figure 6.21 presents the total execution obtained from the two versions of the strip partitioning algorithm.

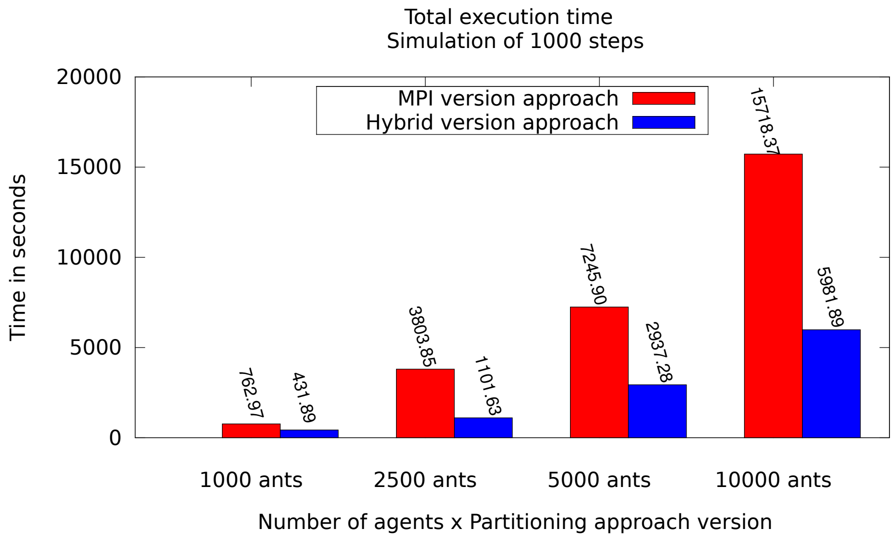


Figure 6.21: Total execution time of pure MPI and Hybrid strip partitioning algorithm.

As we can observe in Figure 6.21, the hybrid partitioning strip approach considerably reduces the total execution time of the simulation. Two aspects are important to note: 1) all ants in this experimentation emerge from the nest as specified in the Ant Colony model. This means that all of the ants in the simulation are in the same core at step zero. During the simulation execution, the agents start to migrate to other strips, but the cores which contain the nest and food always will be in higher demand, while others might not be completely busy; 2) we are executing the experimentation taking into account the worst partitioning heuristic in accordance with [18]. In the horizontal fixed heuristic, there is no special mapping policy nor is the computing cost of each strip taken into account. In this heuristic, each strip is allocated to one core.

After the results presented in Figure 6.21, we executed another test in order to confirm the overload effect of centralized agents in a few strips. We have executed the experimentation distributing the ants uniformly throughout the environment. As we can see in Figure 6.22, the performance results are completely different. Firstly, all of the total execution times are lower than the previous experimentation, because the ants are distributed over all of the strips and can therefore find all of the food in the environment faster. Secondly, the agents are better distributed throughout the environment; therefore, the dynamic creation of threads overloads the cores because the number of idle cores is low. As a consequence, it results in a worse total execution time for the hybrid approach.

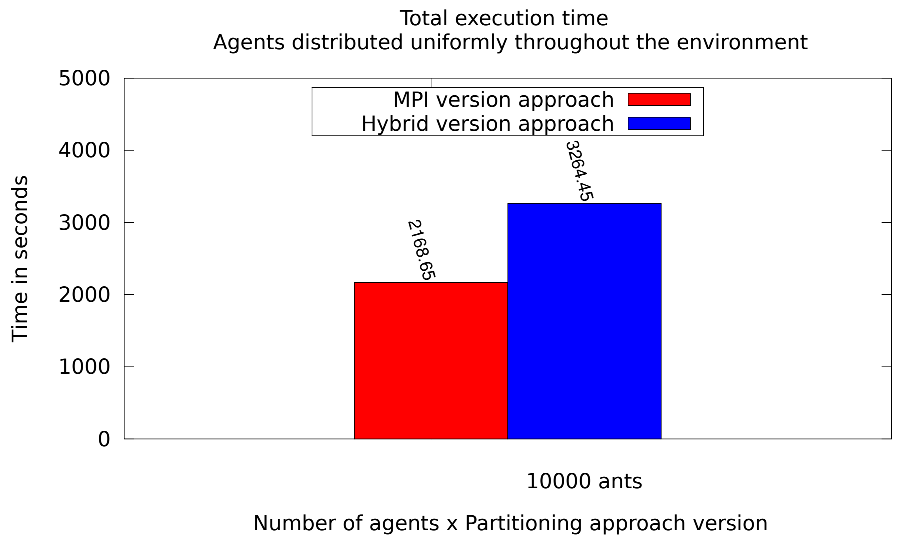


Figure 6.22: Total execution time of pure MPI and hybrid strip partitioning algorithm. Agents distributed uniformly throughout the environment.

These results show that the hybrid partitioning strip is an interesting approach when there is a high concentration of agents in the same region of a simulated space.

6.3 Care HPS features

In ABM, a collective behavior emerges as a result of local interactions between agents that are within a limited vision range. Another important issue is the environment where the agents live and interact with environment.

In the next experiments, we present two important things: First, we will show that Care HPS is able to reproduce an emergent collective behavior; and second, we will present an experimentation using our maths strategy to represent the environment and model the interaction between the agents and the environment.

6.3.1 Agent layer

In the first experimentation, we show that Care HPS is able to reproduce the iterations among the agents. We will compare the results of the simulation obtained by Care HPS with the results obtained using the Netlogo version of the Buyer model proposed by [61]. These authors compare three different versions of buyers. Gilbert and Troitzsch show the increase in agent intelligence by presenting the number of ticks required by all agents to buy all of their products.

ABM tools must be able to model agent roles and behaviours. So, this model can create collective and emergent behaviour. It is very important that these tools can reflect the iteration among agents. Moreover, these tools must enable researchers to improve and analyse the agent behaviour under study.

In this experimentation, we have implemented the Shopping model proposed by Gilbert and Troitzsch [61]. We have implemented three versions of buyer agent. In Version 1, a buyer walks randomly throughout the environment. Version 2 has a buyer that can see the stores in its neighborhood. And, lastly, in Version 3, the buyer talks with other buyers and exchanges information about the locations of the stores. The behaviours implemented in a specific version include all of the behaviours of the previous one. This means that the Buyer Version 1 is the least intelligent one; Version 2 is a little more intelligent; and, finally, Version 3 is the smartest.

The faster, the buyers buy all of their products, the smarter the model is. Therefore, the smartest version tends to finish the simulation in less simulation steps (ticks). Gilbert and Troitzsch [61] executed their simulation 100 times and got an average of 14310 ticks for the first buyer agent implementation. For the second buyer agent version, the authors decreased the number of ticks to 6983. And, the last version of the buyer agent obtained approximately 2000 ticks. Figure 6.23 shows the ticks obtained for Care HPS model execution and the results from Netlogo described in [61]. We are not concerned with making a comparison of the absolute values of ticks obtained in these experimentations. More than that, we want to show that Care HPS can model different behaviours.

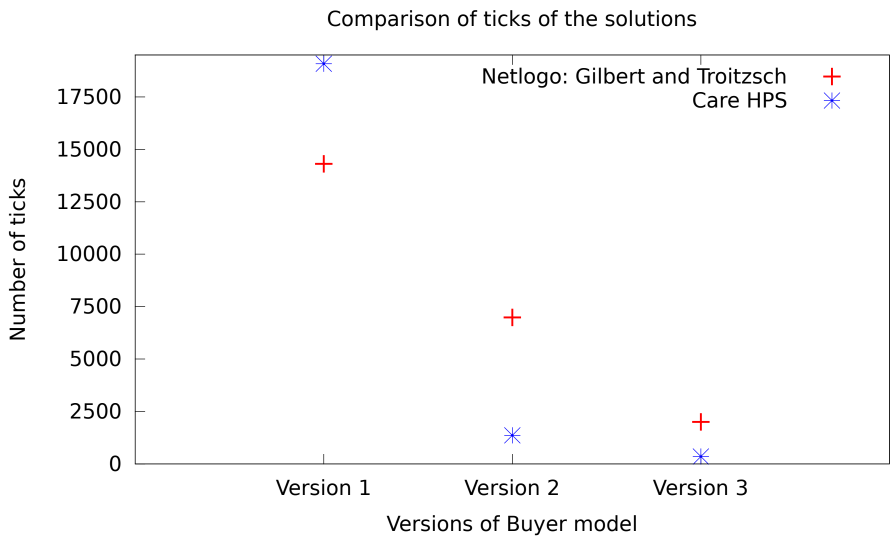


Figure 6.23: Ticks of Gilbert and Troitzsch [61] Netlogo version and Care HPS.

As we can see in Figure 6.23, the number of ticks in Care HPS version decreases as well in the different versions. Therefore, this experimentation shows that Care HPS can model agents and their behaviours. The decrease in ticks in the versions shows that there are interactions among the agents, as well as showing, that changes in the agent's behaviour influence the results of the simulation.

6.3.2 Environment layer

In this experimentation, we will present the result of the implementation of the fish repulsion behaviour for avoiding collision using our math strategy to represent the environment. Figure 6.24 shows the result of the fish repulsion behaviour for avoiding collision. In this figure, a sequence of images shows the school swimming towards an obstacle that is represented by a plan. When the fish "see" the obstacle (around steps 205 and 375), each fish executes its repulsion behaviour and the collision is avoided.

Each fish can see the obstacle because each fish asks the environment if there is some obstacle inside its radius. In terms of code, it is implemented just by checking the intersection between the plan created and the vector equation of a line that represents the agent's position and direction, see 5.5.1. This approach can be used to model other behaviours and other types of interactions between agents and the environment. As an example, we can cite the Ant Colony model where we can find different interactions reproducing different behaviours: 1) the ants drop a chemical in the environment. The chemicals are objects of a type circle that are created dynamically to represent the pheromone. The smell of the pheromone is reduced, decreasing the radius of circle; 2) the ants "eat" the food when they find it. The food is represented per circle. The radius of the circle is decreased when an ant finds it. This means that the food was eaten by an ant; 3) the ants carry the food to the nest. Another example is the Shopping model, where the buyer buys its product when it finds the store.

This experimentation highlights three things: 1) Care HPS is able to represent an environment using a math approach; 2) Care HPS enables the interaction between agents and the environment; and 3) Care HPS enables the representation of simple roles in agents and these simple roles can generate a collective behavior after agent iterations.

In addition, it is important to note that the user can choose between two built-in (circle and linear plan) objects or create their own object to represent their model, and the user defines the behavior executed after the interaction with the environment. Therefore, Care HPS offers many possibilities for representations of the environment and its interactions with agents. Additional built-in objects will be included in future versions of Care HPS.

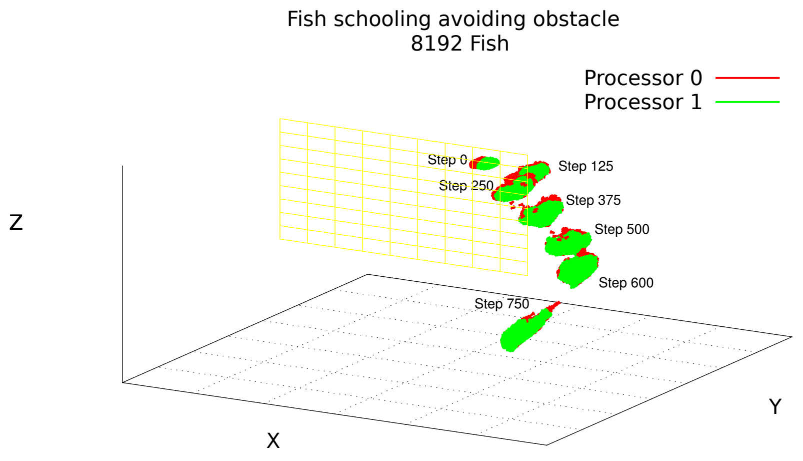


Figure 6.24: Fish repulsion behavior to avoid the collision. This experimentation was executed in two cores using 8192 agents.

6.4 Care HPS scalability

In this experiment, we will approach scalability. Scalability is an important measure for parallel and distributed simulation. Simulations require a minimum number of executions in order to obtain data that is statistically reliable for a correct output analysis. In [37], it is possible to find some techniques for identifying the appropriate number of steps or the number of complete executions required for a simulation. Generally, this number of required simulations is high enough to require a fast single execution. Therefore, the solution must be scalable in order to adjust the number of cores to the time restrictions. This can be illustrated briefly by Figure 6.25. The time required to execute in one core is around 4610 seconds. Suppose this model requires at least 10 executions of the simulation to give an appropriate output. This same result could be obtained using 64 cores, but the total execution time would decrease from almost 13 hours to approximately 1 hour.

We have executed all experiments of this section with Buyer Version 3, where each one had to buy 10 products. Figures 6.25 and 6.26 provide the results obtained from the execution of the model. They show the decrease in time when we include more processors. This occurs because more processors are available to execute the same workload and the total workload is divided among more units of processing. If the solution is scalable, the execution time will decrease when the number of processors increases. As it is possible to note, the decrease in time is not proportional to the number of cores because of communication between the processes, the random movement of agents in the environment and the idleness of the

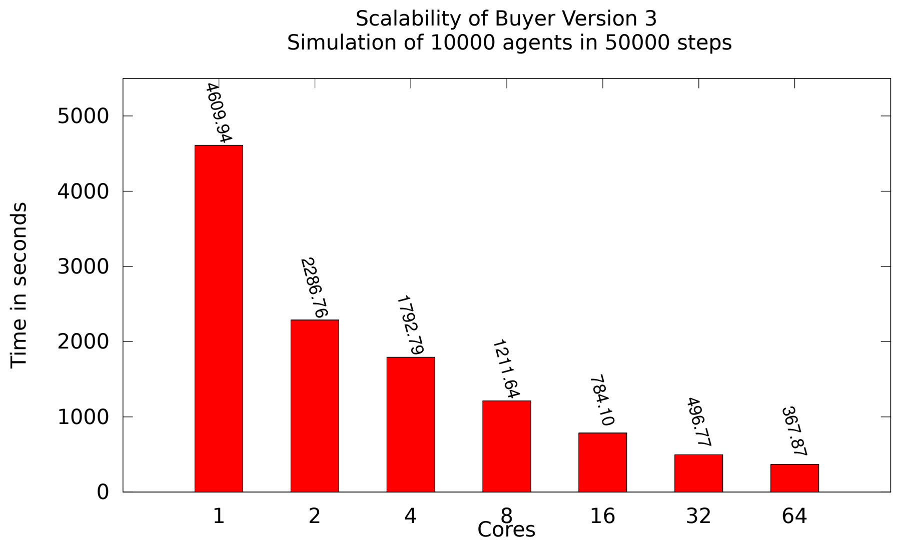


Figure 6.25: Scalability of Buyer Version 3 with 10000 agents in 50000 steps.

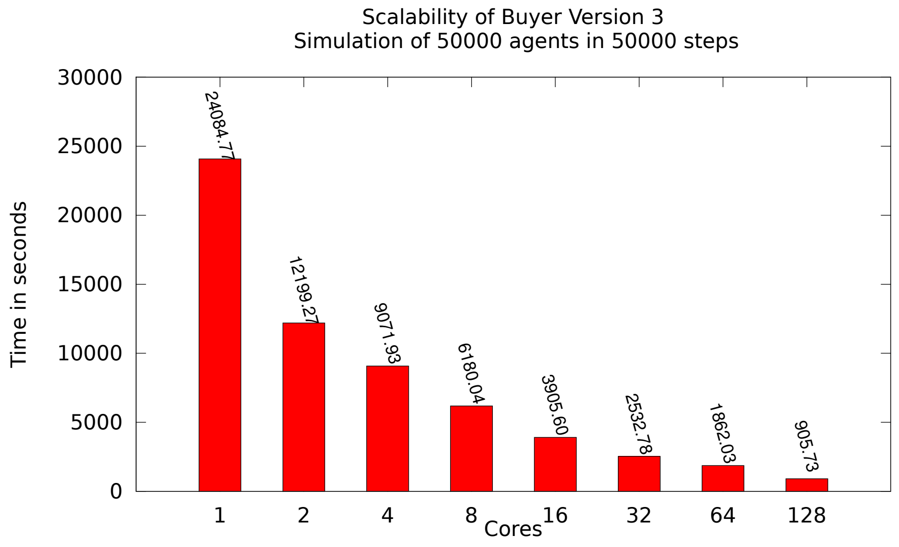


Figure 6.26: Scalability of Buyer Version 3 with 50000 agents in 50000 steps.

processors due to the movement of agents on the partitions. All these issues are recurrent aspects in parallel and distributed research for ABM.

In order to show that Care HPS can handle large models, we executed another test with a greater number of agents (Figure 6.27). If we consider a fixed number of cores (64 cores) and we increment the complexity of agents or the number of agents in the simulation, the execution time will increase. As we can see, Care HPS is able to carry out large numbers of agents. Therefore, Care HPS can execute larger simulations by just including more cores to solve the problem modeled, as depicted in Figure 6.28 where we execute the simulation for 200000 and 250000 agents and increase the number of cores to get a lower execution time. The results show that scalable behavior is sustained in Care HPS. Besides scalability, Care HPS can handle high quantities of agents and models with high complexity. Also, it enables the user to adjust the parallel and distributed simulation to their needs and computational resources.

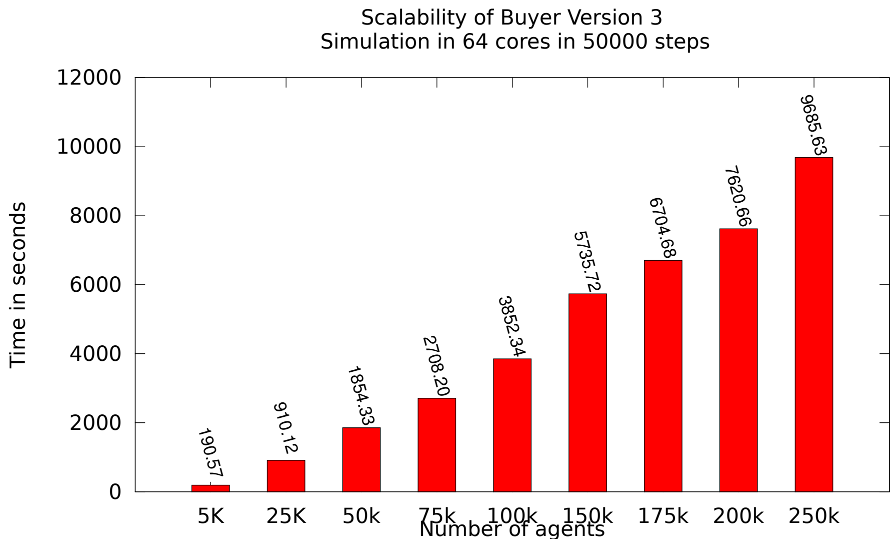


Figure 6.27: Scalability of Buyer Version 3 in 50000 steps executed in 64 cores.

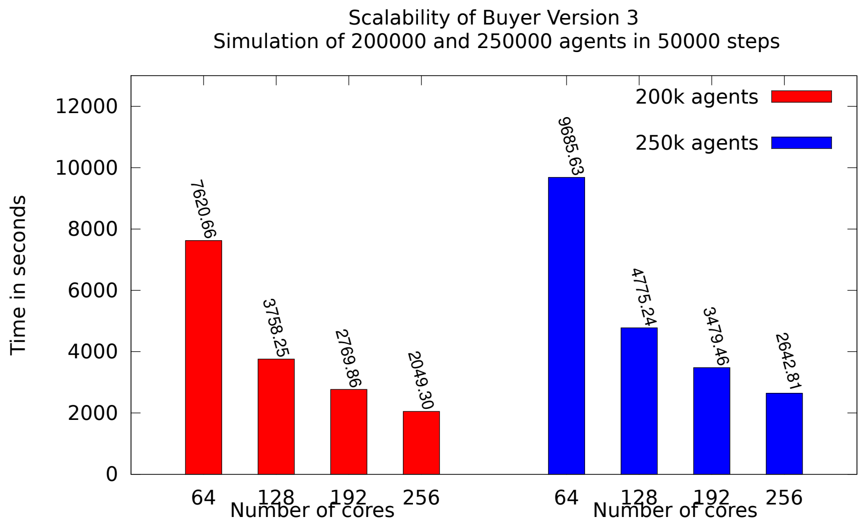


Figure 6.28: Scalability of Buyer Version 3 in 50000 steps to 200k and 250k agents executed in 64, 128, 192 and 256 cores.

Chapter 7

Conclusion

7.1 Introduction

In this chapter, we present the final conclusions reached from this research. Also, we suggest a list of future works and open lines that can be used to improve Care HPS. Lastly, we present our publications where we shared with the scientific community our contributions as presented in this document.

7.2 Final conclusion

In this thesis, we introduce Care High Performance Simulation (HPS). The initial idea of Care HPS comes up as a methodology to support our research group, with the aim of answering the following question: how can we generalize our HPC techniques and approaches for agent-based models that demand high performance? Care HPS is a methodology to execute agent-based modeling and simulation in a parallel and distributed architecture. Care HPS is a scientific instrument to do research on HPC for agent-based models that demand high performance solutions. Care HPS enables both:

- application area researchers to gain knowledge about the system under study using ABMs that require high performance computing solutions. This is possible because Care HPS offers well-defined and simple interfaces for this type of user in which all HPC complexity is hidden;
- and, HPC expert users to develop approaches of high performance parallel and distributed simulation for ABM problems without high programming effort. Care HPS was projected using good object-oriented design practices which allow for the extension and reuse of the main HPS features.

Care HPS supports a distributed memory system which allows for the distribution of the simulation space per several processors using MPI. In addition, it may use the shared memory paradigm to compute agents which are located at the same processor through OpenMP in order to increase the efficiency of the execution of a model. Care HPS was designed with a focus on good object-oriented programming practices and it is composed of several layers and components coded in C++ language.

In the literature, we find several ABM tools which address and support HPC features using generic approaches. However, they do not allow the user to extend the HPC features without a huge programming time investment. Thus, Care HPS was born as a scientific instrument that supports HPC experimentation for ABMS that demand high performance solutions. We can consider this is to be the main novelty of this research. The design of Care HPS permits all of the complexity of the HPC to remain hidden for application area researchers. This means that application area researchers do not need to worry about: how to distributed the agent in different cores; or how to balance the agent over processes; or how to synchronize the processes. These are a few examples of HPC issues that are hidden by Care HPS through the use of its layers and components. This enables the users that are not familiar with but that need HPC to be able to use it without a huge programming effort or know-how. In addition, the design of Care HPS enables HPC experts to extend and reuse previously written code. This is possible because Care HPS uses several design patterns. These design patterns allow the HPC users to improve their HPC solutions, and also easily allow for the addition of new features and resources. In summary, these layers and components have been implemented:

- Three partitioning algorithms: cluster-based partitioning, strip partitioning and hybrid strip partitioning;
- Load balancing algorithms: re-size cluster, re-size strip and number of agents;
- Communication using three patterns: asynchronous, synchronous and BSP;
- Serialization of agents;
- Random number generation;
- Synchronization facility with callback function;
- Representation of environment object through math approach;
- Support to any agent;
- Well-defined classes and interfaces to model agent behavior and environment;

- Low coupling between the layers and between components.

From this point on, Care HPS can be used by other researchers in order to execute experimentations with agent-based models that require high performance simulations for both the application area user and for the HPC expert user. With Care HPS, the user does not need to develop a solution from scratch and with high programming effort. Therefore, as part of our main findings and contributions, we present Care HPS, and we show through experimentation that Care HPS meets its objective and can be used as a scientific instrument for agent-based modeling that requires high performance parallel and distributed simulations.

7.3 Future work and open lines

Currently, we are doing a comprehensive study of the ABM for the assessment of *Aedes Aegypti* pupal productivity [19] in a large real area of São Sebastião (São Paulo, Brazil) using Care HPS. The aim is to use Care HPS to simulate a dengue outbreak that is a demand computing problem and is a real problem. At present, Care HPS continues in the development and improvement of its features. Care HPS is a tool that gives us many possibilities for future works and open lines. This is as true for improvements in the tool as for new models and HPC solutions, techniques and approaches. The literature presents some "development priorities" that are not yet implemented in Care HPS, but that are present in our plan for future work and open lines. Also, we include other actions that we consider important:

- We believe that two comparisons should be carried out between Care HPS and other ABM tools:(1) The first one is comparing the scalability, speedup and efficiency; (2) the other important comparison is about the facility of development and maintenance of these tools.

- It is very important to propose and develop a visual layer for Care HPS. The visual layer should be composed of: 1) automatic visualization generation of the simulation on-line and off-line; and 2) integrated development environment (IDE) for Care HPS.

- Include a new layer in Care HPS to support the execution of the application area researcher models in a cloud. This layer would enable the execution of Care HPS in the cloud.

- Create an output analysis component. The lack of an output analysis module is a limitation in several ABMS tools. The aim of this component would be to provide statistical information for the user with data that comes from the simulation execution.

- Develop a Graphical User Interface (GUI) to define the agent and environment. A GUI can facilitate the development of agent and environment for the user. Some strategies such as data flow and drag-drop are means to decrease the gap between the non-expert user and the code programming.
- Provide more built-in load balancing, load computing, partitioning strategies, environment and objects;
- Provide to the user with templates/scripts that would facilitate the definition of the model. Today, the user has the support of code example and API documentation to develop their model. The idea is for some user input to automatically create the model classes for the application area researchers.
- Provide affinity of processes and threads component. HPC solutions can take advantages of affinity; therefore, a component with this purpose may bring a performance improvement for some solutions.

7.4 List of publications

This section presents our publications in chronological order. At the time of printing this document, we have 12 publications.

The following papers are strictly related with this research.

- Roberto Solar, **Francisco Borges**, Remo Suppi, Emilio Luque. **Improving Communication Patterns for Distributed Cluster-Based Individual-Oriented Fish School Simulations**. ICCS 2013: 702-711. (CORE A)
- **Francisco Borges**, Roberto Solar, Remo Suppi y Emilio Luque. **Performance and scalability in distributed cluster-based individual-oriented fish school simulations**. Actas de las XXIV Jornadas de Paralelismo, Madrid (Madrid), 17-20 Septiembre 2013:401-406. (Unranked)
- Gallo, S., **Borges, F.**, Suppi, R., Luque Fadón, E., De Giusti, L. C., and Naiouf, M. (2013). **Mejoras en la eficiencia mediante Hardware Locality en la simulación distribuida de modelos orientados al individuo. In XVIII Congreso Argentino de Ciencias de la Computación**. (Unranked)
- **Francisco Borges**, Albert Gutierrez-Milla, Remo Suppi, Emilio Luque. **A Hybrid MPI+OpenMP Solution of the Distributed Cluster-Based Fish Schooling Simulator**. ICCS 2014. (CORE A)

- **Francisco Borges**, Albert Gutierrez-Milla, Remo Suppi, Emilio Luque. **Optimal Run Length for Discrete-Event Distributed Cluster-Based Simulations**. ICCS 2014. (CORE A)
- **Francisco Borges**, Albert Gutierrez-Milla, Remo Suppi, Emilio Luque. **Strip Partitioning for Ant Colony Parallel and Distributed Discrete-Event Simulation**. ICCS 2015. (CORE A)
- **Francisco Borges**, Albert Gutierrez-Milla, Remo Suppi, Emilio Luque. **An Agent-Based Model for Assessment of Aedes Aegypti Pupal Productivity**. WSC 2015. (CORE B)

The following papers are other publications with our research group related with ABM and HPC.

- Albert Gutierrez-Milla, **Francisco Borges**, Remo Suppi, Emilio Luque. **Individual-Oriented Model Crowd Evacuations Distributed Simulation**. ICCS 2014. (CORE A)
- Albert Gutierrez-Milla, **Francisco Borges**, Remo Suppi, Emilio Luque. **Crowd evacuations SaaS: an ABM approach**. ICCS 2014. (CORE A)
- Albert Gutierrez-Milla, **Francisco Borges**, Remo Suppi, Emilio Luque. **Simulació de evacuaciones multitudinarias basadas en modelos orientados al individuo**. Actas de las XXV Jornadas de Paralelismo, Valladolid, 17-19 Septiembre 2014. (Unranked)
- Albert Gutierrez-Milla, **Francisco Borges**, Remo Suppi, Emilio Luque. **Crowd Dynamics Modeling and Collision Avoidance with OpenMP**. WSC 2015. (CORE B)
- Albert Gutierrez-Milla, **Francisco Borges**, Remo Suppi, Emilio Luque. **Crowd turbulence with ABM and Verlet Integration on GPU cards**. ICCS 2016. (CORE A)

In addition, we submitted the following papers and they are under review:

- FGCS - Journal Future Generation Computer Systems (Q1 Impact factor of 2.786). We are in the 2nd revision (minor review).

Bibliography

[1] Adhianto, L. and Chapman, B. (2006). Performance modeling of communication and computation in hybrid mpi and openmp applications. In *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, volume 2, pages 6 pp.–.

[2] Alexopoulos, C. (2006). A comprehensive review of methods for simulation output analysis. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 168–178.

[3] Alexopoulos, C. and Goldsman, D. (2004). To batch or not to batch? *ACM Trans. Model. Comput. Simul.*, 14(1):76–114.

[4] Allan, R. J. (2010). Survey of agent based modelling and simulation tools. Technical report, Science and Technology Facilities Council Daresbury Laboratory.

[5] Aoki, I. (1982). A simulation study on the schooling mechanism in fish. *Bulletin of the Japanese Society of Scientific Fisheries*, 48(8):1081–1088.

[6] Argonne National Laboratory (2015). Repasthpc tutorial. Accessed Sep. 02, 2015. http://repast.sourceforge.net/hpc_tutorial/main.html.

[7] Aurenhammer, F. (1991). Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405.

[8] Axtell, R., Axelrod, R., Epstein, J. M., and Cohen, M. D. (1996). Aligning simulation models: A case study and results. *Computational & Mathematical Organization Theory*, 1(2):123–141.

[9] Bahulkar, K., Wang, J., Abu-Ghazaleh, N., and Ponomarev, D. (2012). Partitioning on dynamic behavior for parallel discrete event simulation. In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, PADS '12, pages 221–230, Washington, DC, USA. IEEE Computer Society.

[10] Banks, J. (1998). *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.

[11] BBC News (2016). Zika-linked condition: Who declares global emergency. Accessed Feb. 18, 2016. http://www.bbc.com/news/health-35459797.

[12] Behdani, B. (2012). Evaluation of paradigms for modeling supply chains as complex socio-technical systems. In *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–15.

[13] Blaise Barney, L. L. N. L. (2015). Openmp tutorial. Accessed Apr. 11, 2016. https://computing.llnl.gov/tutorials/openMP/.

[14] Blaise Barney, L. L. N. L. (2016). Introduction to parallel computing. Accessed Apr. 11, 2016. https://computing.llnl.gov/tutorials/parallel_comp/.

[15] Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA.

[16] Borges, F., Gutierrez-Milla, A., Suppi, R., and Luque, E. (2014a). A hybrid mpi+openmp solution of the distributed cluster-based fish schooling simulator. *Procedia Computer Science*, 29(0):2111 – 2120. 2014 International Conference on Computational Science.

[17] Borges, F., Gutierrez-Milla, A., Suppi, R., and Luque, E. (2014b). Optimal run length for discrete-event distributed cluster-based simulations. *Procedia Computer Science*, 29(0):73 – 83. 2014 International Conference on Computational Science.

[18] Borges, F., Gutierrez-Milla, A., Suppi, R., and Luque, E. (2015a). Strip partitioning for ant colony parallel and distributed discrete-event simulation. *Procedia Computer Science*, 51(0):483 – 492. International Conference On Computational Science, {ICCS} 2015 Computational Science at the Gates of Nature.

[19] Borges, F., Gutierrez-Milla, A., Suppi, R., Luque, E., and de Brito Arduino, M. (2015b). An agent-based model for assessment of aedes aegypti pupal productivity. In *Proceedings of the 47th conference on Winter simulation*.

[20] Bragard, Q., Ventresque, A., and Murphy, L. (2014). Synchronisation for dynamic load balancing of decentralised conservative distributed simulation. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '14, pages 117–126, New York, NY, USA. ACM.

[21] Brailsford, S. (2014). Discrete-event simulation is alive and kicking! *Journal of Simulation*, 8(1):1–8.

[22] Brailsford, S. and Hilton, N. (2001). A comparison of discrete event simulation and system dynamics for modelling health care systems. In *Proceedings of the 26th Meeting of the ORAHS Working Group 2000*, pages 18–39. Glasgow Caledonian University.

[23] Brasil (2013). *Larval Index Rapid Assay of Aedes aegypti (LIRAa) for Entomological Surveillance of Aedes aegypti in Brazil: methodology for assessment of Breteau and Building's indexes and type of containers*. Ministério da Saúde. Secretaria de Vigilância em Saúde. Departamento de Vigilância das Doenças Transmissíveis., Brasília, 1 edition.

[24] Brito-Arduino, M. (2014). Assessment of aedes aegypti pupal productivity during the dengue vector control program in a costal urban centre of são paulo state, brazil. *Journal of Insects*, 2014:9.

[25] Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. (2010). hwloc: A generic framework for managing hardware affinities in hpc applications. In *Proceedings of the 2010 18th Euromicro*

Conference on Parallel, Distributed and Network-based Processing, PDP '10, pages 180–186, Washington, DC, USA. IEEE Computer Society.

[26] Buyya, R. (1999). *High Performance Cluster Computing: Programming and Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[27] Cappello, F. and Etiemble, D. (2000). Mpi versus mpi+openmp on the ibm sp for the nas benchmarks. In *Supercomputing, ACM/IEEE 2000 Conference*, page 12.

[28] Casanova, H. (2001). Simgrid: a toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 430–437.

[29] Casanova, H., Giersch, A., Legrand, A., Quinson, M., and Suter, F. (2014). Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899 – 2917.

[30] Chan, M. K. and Yang, L. (2011). Comparative analysis of openmp and mpi on multi-core architecture. In *Proceedings of the 44th Annual Simulation Symposium*, ANSS '11, pages 18–25, San Diego, CA, USA. Society for Computer Simulation International.

[31] Chan, W., Son, Y.-J., and Macal, C. (2010). Agent-based simulation tutorial - simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. In *Winter Simulation Conference (WSC), Proceedings of the 2010*, pages 135–150.

[32] Chavez, E. and Navarro, G. (2000). An effective clustering algorithm to index high dimensional metric spaces. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, pages 75–, Washington, DC, USA. IEEE Computer Society.

[33] Chin, A. L., Worth, A. D., Greenough, A. C., Coakley, S., Holcombe, M., and Kiran, M. (2012a). Flame: An approach to the parallelisation of agent-based applications. Technical report, Science and Technology Facilities Council Rutherford Appleton Laboratory.

[34] Chin, L., Science, and Britain), T. F. C. G. (2012b). Flame-ii: A redesign of the flexible large-scale agent-based modelling environment. Technical report, Science and Technology Facilities Council Rutherford Appleton Laboratory.

[35] Chorley, M. J. and Walker, D. W. (2010). Performance analysis of a hybrid mpi/openmp application on multi-core clusters. *Journal of Computational Science*, 1(3):168–174.

[36] Chow, E. and Hysom, D. (2001). Assessing performance of hybrid mpi/openmp programs on smp clusters.

[37] Chung, C. A. (2003). *Simulation modeling handbook: a practical approach*. CRC press.

[38] Coakley, S., Gheorghe, M., Holcombe, M., Chin, S., Worth, D., and Greenough, C. (2012). Exploitation of high performance computing in the flame agent-based simulation framework. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 538–545.

[39] Coakley, S., Smallwood, R., and Halcombe, M. (2006). Using x-machines as a formal basis for describing agents in agent-based modelling. In *Agent-Directed Simulation, SpringSim 06*, Huntsville, AL, USA.

[40] Collier, N. (2010). Repast hpc manual. Accessed Aug. 27, 2015. http://repast.sourceforge.net/docs.php.

[41] Collier, N. and North, M. (2013). Parallel agent-based simulation with repast for high performance computing. *SIMULATION: Transactions of the Society for Modeling and Simulation International*, 89(10):1215–1235.

[42] Cordasco, G., Chiara, R. D., Mancuso, A., Mazzeo, D., Scarano, V., and Spagnuolo, C. (2013). Bringing together efficiency and effectiveness in distributed simulations: The experience with d-mason. *Simulation*, pages 1236–1253.

[43] Dalforno, C., Mostaccio, D., Suppi, R., and Luque, E. (2008). Increasing the scalability and the speedup of a fish school simulator. In Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y., and Gavrilova, M., editors, *Computational Science and Its Applications – ICCSA 2008*, volume 5073 of *Lecture Notes in Computer Science*, pages 936–949. Springer Berlin Heidelberg.

[44] Deelman, E. and Szymanski, B. K. (1998). Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. *SIGSIM Simul. Dig.*, 28:46–53.

[45] Deissenberg, C., van der Hoog, S., and Dawid, H. (2008). Eurace: A massively parallel agent-based model of the european economy. *Applied Mathematics and Computation*, 204(2):541 – 552. Special Issue on New Approaches in Dynamic Optimization to Assessment of Economic and Environmental Systems.

[46] Diaz, J., Munoz-Caro, C., and Nino, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. *Parallel and Distributed Systems, IEEE Transactions on*, 23(8):1369–1386.

[47] Drosinos, N. and Koziris, N. (2004). Performance comparison of pure mpi vs hybrid mpi-openmp parallelization models on smp clusters. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 15.

[48] Fauci, A. S. and Morens, D. M. (2016). Zika virus in the americas — yet another arbovirus threat. *New England Journal of Medicine*. PMID: 26761185.

[49] Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960.

[50] Focks, D. A. (2003). *A review of entomological sampling methods and indicators for dengue vectors*. Document TDR/IDE/Den/03, World Health Organization, Geneva, Switzerland.

[51] Focks, D. A. and Alexander, N. (2006). *Multi-country study of Aedes aegypti pupal productivity survey methodology: findings and recommendations*. Document TDR/IRM/DEN/06, World Health Organization, Geneva, Switzerland.

[52] Focks, D. A., Brenner, R. J., Hayes, J., and Daniels, E. (2000). Transmission thresholds for dengue in terms of aedes aegypti pupae per person with discussion of their utility in source reduction efforts. *The American Journal of Tropical Medicine and Hygiene*, 62(1):11–18.

[53] Focks, D. A. and Chadee, D. D. (1997). Pupal survey: An epidemiologically significant surveillance method for aedes aegypti: An example using data from trinidad. *The American Journal of Tropical Medicine and Hygiene*, 56(2):159–167.

[54] Focks, D. A., Daniels, E., Haile, D. G., Keesling, J. E., et al. (1995). A simulation model of the epidemiology of urban dengue fever: literature analysis, model development, preliminary validation, and samples of simulation results. *American Journal of Tropical Medicine and Hygiene*, 53(5):489–506.

[55] Focks, D. A., Haile, D. G., Daniels, E., and Mount, G. A. (1993a). Dynamic life table model for aedes aegypti (diptera: Culicidae): analysis of the literature and model development. *Journal of medical entomology*, 30(6):1003–1017.

[56] Focks, D. A., Haile, D. G., Daniels, E., and Mount, G. A. (1993b). Dynamic life table model for aedes aegypti (diptera: Culicidae): simulation results and validation. *Journal of medical entomology*, 30(6):1018–1028.

[57] Foster, I. (1995). *Designing and Building Parallel Programs*. Addison-Wesley Longman Publishing Co., Inc.

[58] Fujimoto, R. M. (1999). *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[59] Gallo, S., Borges, F., Suppi, R., Fadón, E. L., Giusti, L. C. D., and Naiouf, M. (2013). Mejoras en la eficiencia mediante hardware locality en la simulación distribuida de modelos orientados al individuo. In *XVIII Congreso Argentino de Ciencias de la Computación*.

[60] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[61] Gilbert, N. and Troitzsch, K. G. (2005). *Simulation for the Social Scientist*. Open University Press, 2nd edition edition.

[62] Ginot, V., Page, C. L., and Souissi, S. (2002). A multi-agents architecture to enhance end-user individual-based modelling. *Ecological Modelling*, 157(1):23 – 41.

[63] González, J. C., Dalforno, C., Suppi, R., and Luque, E. (2009). A fuzzy logic fish school model. In *ICCS*, volume 5544 of *Lecture Notes in Computer Science*, pages 13–22.

[64] Gueron, S., Levin, S., and Rubenstein, D. (1996). The dynamics of herds: From individuals to aggregations. *Journal of Theoretical Biology*, 182:85–98(14).

[65] Hager, G., Jost, G., and Rabenseifner, R. (2009). Communication characteristics and hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. *Proceedings of Cray User Group Conference*, 4(d):54–55.

[66] Helbing, D. and Balietti, S. (2012). *How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design*, chapter Agent-Based Modeling, pages 25–70. Springer, Berlin. Available at SSRN: http://ssrn.com/abstract=2339770.

[67] Hiebeler, D. (1994). The swarm simulation system and individual-based modeling. Santa Fe Institute.

[68] Hill, J. M. D., McColl, B., Stefanescu, D. C., Goudreau, M. W., Lang, K., Rao, S. B., Suel, T., Tsantilas, T., and Bisseling, R. H. (1998). Bsplib: The bsp programming library. *Parallel Comput.*, 24(14):1947–1980.

[69] Holcombe, M., Coakley, S., and Smallwood, R. (2006). A general framework for agent-based modelling of complex systems. In *Proceedings of the 2006 European Conference on Complex Systems*.

[70] Honório, N. A., Silva, W. d. C., Leite, P. J., Gonçalves, J. M., Lounibos, L. P., and Lourenço-de Oliveira, R. (2003). Dispersal of aedes aegypti and aedes albopictus (diptera: Culicidae) in an urban endemic dengue area in the state of rio de janeiro, brazil. *Memórias do Instituto Oswaldo Cruz*, 98:191 – 198.

[71] Huth, A. and Wissel, C. (1992). The simulation of the movement of fish schools. *Journal of Theoretical Biology*, 156(3):365 – 385.

[72] Huth, A. and Wissel, C. (1994). The simulation of fish schools in comparison with experimental data. *Ecological Modelling*, 75-76:135 – 146. State-of-the-Art in Ecological Modelling proceedings of ISEM's 8th International Conference.

[73] Icons8 (2016). Largest colletion of icons. Accessed Jun. 09, 2016. https://icons8.com. Some icons/maps/draws under: CC BY-ND 3.0.

[74] Inria (2015). Simgrid versatile simulation of distributed systems. Accessed Sep. 17, 2015. http://simgrid.gforge.inria.fr/.

[75] Isidoro, C., Fachada, N., Barata, F., and Rosa, A. (2009). Agent-based model of aedes aegypti population dynamics. In Lopes, L., Lau, N., Mariano, P., and Rocha, L., editors, *Progress in Artificial Intelligence*, volume 5816 of *Lecture Notes in Computer Science*, pages 53–64. Springer Berlin Heidelberg.

[76] Jacintho, L. F. O., Batista, A. F. M., Ruas, T. L., Marietto, M. G. B., and Silva, F. A. (2010). An agent-based model for the spread of the dengue fever: A swarm platform simulation approach. In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim '10, pages 2:1–2:8, San Diego, CA, USA. Society for Computer Simulation International.

[77] Jafer, S., Liu, Q., and Wainer, G. (2013). Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, 30(0):54 – 73.

[78] Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323.

[79] Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., and Chapman, B. (2011). High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562 – 575. Emerging Programming Paradigms for Large-Scale Scientific Computing.

[80] Kennedy, J. and Eberhart, R. C. (2001). *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[81] Kiran, M., Richmond, P., Holcombe, M., Chin, L. S., Worth, D., and Greenough, C. (2010). Flame: Simulating large populations of agents on parallel hardware architectures. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 1633–1636, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[82] Law, A. M. (2010). Statistical analysis of simulation output data: The practical state of the art. In *Winter Simulation Conference*, pages 65–74.

[83] Lima, T., Carneiro, T., Faria, S., Silva, P., and Pessoa, M. (2013). Terrame gims: An eclipse plug-in for environmental modeling. In *Developing Tools as Plug-ins (TOPI), 2013 3rd International Workshop on*, pages 37–42.

[84] Lima, T., Carneiro, T., Silva, L., Lana, R., Codeco, C., Reis, I., Maretto, R., Santos, L., Monteiro, A., Medeiros, L., and Coelho, F. (2014). A framework for modeling and simulating aedes aegypti and dengue fever dynamics. In Tolk, A., Diallo, S. Y., Ryzhov, I. O., Yilmaz, L., Buckley, S., and Miller, J. A., editors, *Proceedings of the 2014 Winter Simulation Conference*, pages 1481–1492, Piscataway, New Jersey. Institute of Electrical and Electronics Engineers, Inc.

[85] Louis, V., Phalkey, R., Horstick, O., Ratanawong, P., Wilder-Smith, A., Tozan, Y., and Dambach, P. (2014). Modeling tools for dengue risk mapping - a systematic review. *International Journal of Health Geographics*, 13(1):50.

[86] Luke, S., Cioffi-Revilla, C., Panait, L., and Sullivan, K. (2004). Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 swarmfest workshop*, volume 8, page 44.

[87] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005a). Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

[88] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005b). Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

[89] Macal, C. and North, M. (2009). Agent-based modeling and simulation. In *Winter Simulation Conference (WSC), Proceedings of the 2009*, pages 86–98.

[90] Macal, C. M. and North, M. J. (2010). Tutorial on agent-based modelling and simulation. *Journal of simulation*, 4(3):151–162.

[91] Macal, C. M. and North, M. J. (2014). Introductory tutorial: agent-based modeling and simulation. In *Winter Simulation Conference*, pages 1456–1469.

[92] Madella, M., Rondelli, B., Lancelotti, C., Balbo, A., Zurro, D., Rubio-Campillo, X., and Stride, S. (2014). Introduction to simulating the past. *Journal of Archaeological Method and Theory*, 21(2):251–257.

[93] Magori, K., Legros, M., Puente, M. E., Focks, D. A., Scott, T. W., Lloyd, A. L., and Gould, F. (2009). Skeeter buster: A stochastic, spatially explicit modeling tool for studying aedes aegypti population replacement and population suppression strategies. *PLoS Negl Trop Dis*, 3(9):e508.

[94] Mahajan, P. and Ingalls, R. (2004). Evaluation of methods used to detect warm-up period in steady state simulation. In *Simulation Conference, 2004. Proceedings of the 2004 Winter*, volume 1, pages –671.

[95] Maidstone, R. (2012). Discrete event simulation, system dynamics and agent based simulation: Discussion and comparison. *System*, pages 1–6.

[96] Márquez, C., César, E., and Sorribes, J. (2013). A load balancing schema for agent-based spmd applications. *International Conf. on Parallel and Distributed Processing Techniques and Applications, PDPTA*.

[97] Márquez, C., César, E., and Sorribes, J. (2014). *Euro-Par 2013: Parallel Processing Workshops: BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers*, chapter Agent Migration in HPC Systems Using FLAME, pages 523–532. Springer Berlin Heidelberg, Berlin, Heidelberg.

[98] Merchant, F., Bic, L. F., and Dillencourt, M. B. (1998). Load balancing in individual-based spatial applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT'98)*, pages 350–357.

[99] Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996). The swarm simulation system: A toolkit for building multi-agent simulations. Santa Fe Institute Santa Fe.

[100] Mostaccio, D., Dalforno, C., Suppi, R., and Luque, E. (2006). Distributed simulation of large-scale individual oriented models. *Journal of Computer Science & Technology*, 6(2):59–65.

[101] Mostaccio, D., Suppi, R., and Luque, E. (2005a). Simulation of ecologic systems using mpi. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 449–456. Springer.

[102] Mostaccio, D., Suppi, R., and Luque, E. (2005b). Using distributed events simulation for individual oriented models. In *International Mediterranean Multimodeling Multiconference*, volume I, pages 105–110.

[103] Muetzelfeldt, R. I. and Massheder, J. (2003). The simile visual modelling environment. *European Journal of Agronomy*, 18:345–358.

[104] Muller, G., Grébaut, P., and Gouteux, J.-P. (2004). An agent-based model of sleeping sickness: simulation trials of a forest focus in southern cameroon. *Comptes Rendus Biologies*, 327(1):1–11.

[105] Nakayama, M. K. (2006). Output analysis for simulations. In *Proceedings of the 38th Conference on Winter Simulation*, WSC'06, pages 36–46. Winter Simulation Conference.

[106] Nelson, B. (2011). Thirty years of "batch size effects". In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, pages 393–400.

[107] Niazi, M. and Hussain, A. (2011). Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, 89(2):479–499.

[108] Nicol, D. (1994). Rectilinear partitioning of irregular data parallel computations. *Journal of Parallel and Distributed Computing*, 23(2):119 – 134.

[109] Nikolai, C. and Madey, G. (2009). Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2.

[110] North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):1–26.

[111] North, M. J., Collier, N. T., and Vos, J. R. (2006). Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 16(1):1–25.

[112] North, M. J., Tatara, E., Collier, N. T., and Ozik, J. (2007). Visual agent-based model development with repast simphony. In *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*.

[113] OpenStreetMap (2016). Openstreetmap powers map data on thousands of web sites, mobile apps, and hardware devices. Accessed Jun. 09, 2016. https://www.openstreetmap.org. Some maps under CC BY-SA.

[114] Ozik, J., Collier, N., Combs, T., Macal, C. M., and North, M. (2015). Repast simphony statecharts. *Journal of Artificial Societies and Social Simulation*, 18(3):11.

[115] Pacheco, P. (2011). *An Introduction to Parallel Programming*. An Introduction to Parallel Programming. Morgan Kaufmann.

[116] Pacheco, P. S. (1996). *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[117] Paredes, R. U., Marquez, C., and Solar, R. (2009). Construction strategies on metric structures for similarity search. *CLEI Electron. J.*, 12(3).

[118] Parker, M. T. (2001). What is ascape and why should you care. *Journal of Artificial Societies and Social Simulation*, 4(1):5.

[119] Parrish, J. K., Viscido, S. V., and Grünbaum, D. (2002). Self-organized fish schools: An examination of emergent properties. *Biol. Bull*, 202:296–305.

[120] Pasupathy, R. and Schmeiser, B. (2010). The initial transient in steady-state point estimation: Contexts, a bibliography, the mse criterion, and the mser statistic. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 184–197.

[121] Peschlow, P., Honecker, T., and Martini, P. (2007). A flexible dynamic partitioning algorithm for optimistic distributed simulation. In *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, PADS '07, pages 219–228, Washington, DC, USA. IEEE Computer Society.

[122] Rabenseifner, R., Hager, G., and Jost, G. (2009). Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 427 –436.

[123] Railsback, S. F., Lytinen, S. L., and Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *SIMULATION*, 82(9):609–623.

[124] Rao, D. M. (2014). Accelerating parallel agent-based epidemiological simulations. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '14, pages 127–138, New York, NY, USA. ACM.

[125] Rao, D. M. and Chernyakhovsky, A. (2008). Parallel simulation of the global epidemiology of avian influenza. In Mason, S. J., Hill, R. R., Mönch, L., Rose, O., Jefferson, T., and Fowler, J. W., editors, *Proceedings of the 2008 Winter Simulation Conference*, pages 1583–1591, Piscataway, New Jersey. Institute of Electrical and Electronics Engineers, Inc.

[126] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21:25–34.

[127] Roche, B., Drake, J., and Rohani, P. (2011). An agent-based model to study the epidemiological and evolutionary dynamics of influenza viruses. *BMC Bioinformatics*, 12(1):87.

[128] Roche, B., Guégan, J.-F., and Bousquet, F. (2008). Multi-agent systems in epidemiology: a first step for computational biology in the study of vector-borne disease transmission. *BMC Bioinformatics*, 9(1):435.

[129] Ropella, G. E., Railsback, S. F., and Jackson, S. K. (2002). Software engineering considerations for individual-based models. *Natural Resource Modeling*, 15(1):5–22.

[130] Rubio-Campillo, X. (2014). Pandora: A versatile agent-based modelling platform for social simulation. In *Conference Proceedings SIMUL 2014, The Sixth International Conference on Advances in System Simulation*, pages 29–34.

[131] Rubio-Campillo, X. (2015). *Large Simulations and Small Societies: High Performance Computing for Archaeological Simulations*, chapter Part II, page 119–137. Springer, advances in geographic information science edition.

[132] Rubio-Campillo, X., Matías, P. V., and Ble, E. (2015). Centurions in the roman legion: Computer simulation and complex systems. *Journal of Interdisciplinary History*, 46(2):245–263.

[133] Saber, R. O. and Murray, R. M. (2003). Flocking with obstacle avoidance: cooperation with limited communication in mobile networks. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 2, pages 2022–2028 Vol.2.

[134] Saule, E., Bas, E., and Çatalyürek, U. (2011). Partitioning spatially located computations using rectangles. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 709–720.

[135] Schmeiser, B. (1982). Batch size effects in the analysis of simulation output. *Operations Research*, 30(3):556–568.

[136] Science and Technology Facilities Council (2014). Flame user manual. Accessed Jul. 23, 2015. http://www.flame.ac.uk/docs/user_manual.html.

[137] Segovia-Juarez, J. L., Ganguli, S., and Kirschner, D. (2004). Identifying control mechanisms of granuloma formation during m. tuberculosis infection using an agent-based model. *Journal of Theoretical Biology*, 231(3):357–376.

[138] Shannon, R. E. (1975). *Systems Simulation: The Art and Science*. Prentice Hall.

[139] Shende, S. S. and Malony, A. D. (2006). The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311.

[140] Siebers, P.-O., Macal, C. M., Garnett, J., Buxton, D., and Pidd, M. (2010). Discrete-event simulation is dead, long live agent-based simulation! *Journal of Simulation*, 4(3):204–210.

[141] Skillicorn, D. B., Hill, J. M. D., and McColl, W. F. (1997). Questions and answers about bsp. In *Scientific Programming*, volume 6, pages 249–274.

[142] Smith, L. and Bull, M. (2001). Development of mixed mode mpi / openmp applications. *Scientific Programming*, 9(2,3):83–98.

[143] Solar, R., Borges, F., Suppi, R., and Luque, E. (2013). Improving communication patterns for distributed cluster-based individual-oriented fish school simulations. *Procedia Computer Science*, 18(0):702 – 711. 2013 International Conference on Computational Science.

[144] Solar, R., Suppi, R., and Luque, E. (2010). High performance individual-oriented simulation using complex models. *Procedia Computer Science*, 1(1):447 – 456. ICCS 2010.

[145] Solar, R., Suppi, R., and Luque, E. (2011). High performance distributed cluster-based individual-oriented fish school simulation. *Procedia CS*, 4:76–85.

[146] Solar, R., Suppi, R., and Luque, E. (2012). Proximity load balancing for distributed cluster-based individual-oriented fish school simulations. *Procedia Computer Science*, 9(0):328 – 337. Proceedings of the International Conference on Computational Science, ICCS 2012.

[147] Standish, R. K. and Leow, R. (2004). Ecolab: Agent based modeling for C++ programmers. *CoRR*, cs.MA/0401026.

[148] Steiger, N. M., Lada, E. K., Wilson, J. R., Joines, J. A., Alexopoulos, C., and Goldsman, D. (2005). Asap3: A batch means procedure for steady-state simulation analysis. *ACM Trans. Model. Comput. Simul.*, 15(1):39–73.

[149] Suijlen, W. J. (2006). Bsponmpi. Accessed May. 29, 2016. http://bsponmpi.sourceforge.net.

[150] Tafazzoli, A., Wilson, J., Lada, E., and Steiger, N. (2008). Skart: A skewness- and autoregression-adjusted batch-means procedure for simulation analysis. In *Simulation Conference, 2008. WSC 2008. Winter*, pages 387–395.

[151] Tobias, R. and Hofmann, C. (2004). Evaluation of free java-libraries for social-scientific agent based simulation. *Journal of Artificial Societies and Social Simulation*, 7(1).

[152] TOP500.org (2015). Top500 supercomputer sites. Accessed Sep. 03, 2015. http://www.who.int/denguecontrol/en/.

[153] Tun-Lin, W., Kay, B. H., and Barnes, A. (1995). Understanding productivity, a key to aedes aegypti surveillance. *The American Journal of Tropical Medicine and Hygiene*, 53(6):595–601.

[154] Tun-Lin, W., Lenhart, A., Nam, V. S., Rebollar-Téllez, E., Morrison, A. C., Barbazan, P., Cote, M., Midega, J., Sanchez, F., Manrique-Saide, P., Kroeger, A., Nathan, M. B., Meheus, F., and Petzold, M. (2009). Reducing costs and operational constraints of dengue vector control by targeting productive breeding places: a multi-country non-inferiority cluster randomized trial. *Tropical Medicine & International Health*, 14(9):1143–1153.

[155] Tuncer, Ö. (2011). The many facets of simulation through a collection of about 100 definitions. *SCS M&S Magazine*, 2(2):82–92.

[156] Vabø, R. and Skaret, G. (2008). Emerging school structures and collective dynamics in spawning herring: A simulation study. *Ecological Modelling*, 214(2-4):125–140.

[157] van Heesch, D. (1997). Doxygen. Accessed Feb. 06, 2016. http://www.stack.nl/~dimitri/doxygen/index.html.

[158] Villela, D. A. M., Codeço, C. T., Figueiredo, F., Garcia, G. A., Maciel-de Freitas, R., and Struchiner, C. J. (2015). A bayesian hierarchical model for estimation of abundance and spatial density of aedes aegypti. *PLoS ONE*, 10(4):e0123794.

[159] Wang, Y., Lees, M., Cai, W., Zhou, S., and Low, M. (2009). Cluster based partitioning for agent-based crowd simulations. In *Winter Simulation Conference (WSC), Proceedings of the 2009*, pages 1047 –1058.

[160] Welch, P. D. (1983). The statistical analysis of simulation results. In Lavenberg, S., editor, *The Computer Performance Modeling Handbook*, pages 268–328. Academic Press, San Diego, California.

[161] White, K.P., J., Cobb, M., and Spratt, S. (2000). A comparison of five steady-state truncation heuristics for simulation. In *Simulation Conference, 2000. Proceedings. Winter*, volume 1, pages 755–760 vol.1.

[162] Wilensky, U. (1997). Netlogo ants model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Visited on 30-11-2014.

[163] Wilensky, U. (1999a). Netlogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Accessed Apr. 02, 2015. http://ccl.northwestern.edu/netlogo/.

[164] Wilensky, U. (1999b). Netlogo programming guide. Accessed Aug. 10, 2015. http://ccl.northwestern.edu/netlogo/docs/programming.html.

[165] Williams, C. R., Johnson, P. H., Long, S. A., Rapley, L. P., and Ritchie, S. A. (2008). Rapid estimation of aedes aegypti population size using simulation modeling, with a novel approach to calibration and field validation. *Journal of Medical Entomology*, 45(6):1173–1179.

[166] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer-Verlag Berlin Heidelberg.

[167] World Health Organization (2009). *Dengue: Guidelines for Diagnosis, Treatment, Prevention and Control.* World Health Organization Press, Switzerland. Accessed Mar. 07, 2015. http://www.who.int/tdr/publications/documents/dengue-diagnosis.pdf.

[168] World Health Organization (2015). Dengue control. Accessed Mar. 25, 2015. http://www.who.int/denguecontrol/en/.

[169] World Health Organization (2016). Zika virus. Accessed Feb. 18, 2016. http://www.who.int/mediacentre/factsheets/zika/en/.

Appendix A

Diagram of classes

In this Appendix we present UML class diagram overview of Care HPS.

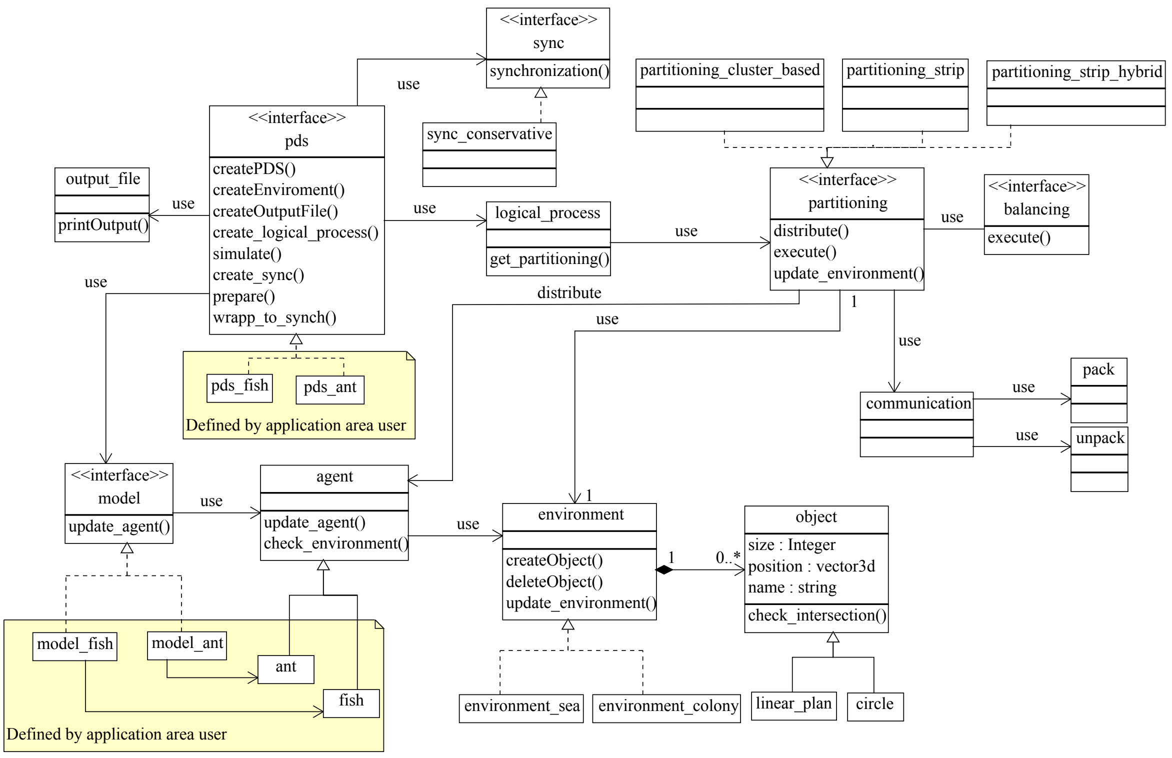


Figure A.1: UML Class diagram overview. We present as example only the Ant and Fish model classes.

Appendix B

Netlogo version:Assessment of *Aedes Aegypti* pupal productivity

```
globals[

 day   ;; Relation between the tick and real time (second, minute or hour)
 time_egg_stage      ;; Lifespan of the mosquito in the egg stage.
 time_larvae_stage   ;; Lifespan of the mosquito in the larvae stage.
 time_pupae_stage    ;;it represents the lifespan of the mosquito in the pupae stage.
 time_total_lifespan_min ;;it represents the total lifespan minimum of the mosquito. After that time it dies.
 time_total_lifespan_max ;;it represents the total lifespan maximum of the mosquito. After that time it dies.

 time_min_extrinsic_incubation    ;;it represents the minimum time required of the extrinsic incubation in
      the mosquito.
 time_max_extrinsic_incubation    ;;it represents the maximum time required of the extrinsic incubation in
      the mosquito.

 time_min_intrinsic_incubation    ;;it represents the minimum time required of the intrinsic incubation in the
       person.
 time_max_intrinsic_incubation    ;;it represents the maximum time required of the intrinsic incubation in
      the person.

 total_number_eggs_deposited_environment ;; It represents the total number of eggs deposited by
      mosquitoes in the environment.

 number_mosquito_die ;;it represents the total number of mosquitoes has died.
 radius ;;it represents the radius flight of of mosquitoes.

 min_number_container_deposit ;; it represents the minimum number of constainers where the mosquitoes
      will try to deposit the eggs.
```

```
  simulation−steps ;; It represents the number of simulations steps

]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; AGENTS DEFINITIONS
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

breed [people person]
breed [mosquitoes mosquito]

turtles−own [
  infected? ; it indicates if the agent is infected.
  age ;it represents the lifespan of the mosquito and the person.
]

mosquitoes−own[
  extrinsic_incubation?
  has−to−deposit−eggs? ; It indicates if the mosquito bite a person and now it has to deposite eggs.
  batch ;It indicates how many times the mosquito deposited its eggs in its life.
  number−deposited−eggs ; It indicates the quantity of eggs already deposited by mosquito.
;;  number−of−container−with−deposits−done ; Number of deposit the mosquito has done for one batch
;;  number−eggs−must−deposit ; It indicates the number of eggs that the mosquito must to deposit.
  pxcor−was−born ; It indicates if the pxcor where the mosquito was born.
  pycor−was−born ; It indicates if the pycor where the mosquito was born.
  radius−flight  ; It indicates the radius of distance that a mosquito can fly.
  female? ;It indicates if the mosquito is female.

  already−counted−as−egg?
  already−counted−as−larvae?
  already−counted−as−pupae?
  already−counted−as−adult?
  my_lifespan ; Lifespan of the mosquito range is a number between the interval time_total_lifespan_min
      and time_total_lifespan_max
]

people−own[
  intrinsic_incubation?
]

;; ENVIRONMENT DEFINITIONS
```

```
patches−own [
 type_container ; Bromeliads = 1, Ships = 2, Water tanks = 3, Drains = 4, Sanitation fixtures = 5, Concrete
     structures = 6, Plastic items = 7, Metallic items = 8,
          ; Vases = 9, Tyres = 10, Aquatic plants = 11, Glass items = 12 or Other = 13
 %productivity−container
 positive? ;it indicates if the container has eggs.
 num−egg
 num−larvae
 num_pupae
 num_adult
]

to setup
 clear−all
 setup−globals−variables
 setup−patches
 setup−population
 reset−ticks
end

to setup−population
 setup−population−people
 setup−population−mosquitoes
end

to setup−globals−variables

 set day 10 ;1440 ;; we consider one tick corresponds to one minute therefore one day requires 1440 ticks
 set time_egg_stage (day ∗ 2)
 set time_larvae_stage ( time_egg_stage + (day ∗ 10) )
 set time_pupae_stage ( time_larvae_stage + (day ∗ 2))

 ;;TODO should be between 6 and 8 weeks
 set time_total_lifespan_min (day ∗ 42) ;; 42 is six weeks
 set time_total_lifespan_max (day ∗ 56) ;; 56 is eight weeks

 set time_min_extrinsic_incubation (day ∗ 8)
 set time_max_extrinsic_incubation (day ∗ 12)

 set time_min_intrinsic_incubation (day ∗ 3)
 set time_max_intrinsic_incubation (day ∗ 15)

 set total_number_eggs_deposited_environment 0
 set number_mosquito_die 0
 set radius 10 ; 100 meters

 set min_number_container_deposit num−containers−deposit

 set simulation−steps 3000

end

to setup−patches

 ifelse group−container?
 [
  ask patch 0 12  [set pcolor gray  set type_container 1 set %productivity−container Bromeliads−
     productivity]
  ask patch 0 10  [set pcolor orange  set type_container 2 set %productivity−container Ships−productivity]
  ask patch 0 8   [set pcolor brown  set type_container 3 set %productivity−container Water−tanks−
     productivity]
  ask patch 0 6   [set pcolor lime  set type_container 4 set %productivity−container Drains−productivity]
  ask patch 0 4   [set pcolor turquoise  set type_container 5 set %productivity−container Sanitation−fixtures
     −productivity]
  ask patch 0 2   [set pcolor cyan  set type_container 6 set %productivity−container Concrete−structures−
     productivity]

  ask patch 0 0   [set pcolor sky set type_container 7 set %productivity−container Plastic−items−
     productivity]
  ask patch 0 −2  [set pcolor 7 set type_container 8 set %productivity−container Metallic−items−
     productivity]
  ask patch 0 −4  [set pcolor 27 set type_container 9 set %productivity−container Vases−productivity]
  ask patch 0 −6  [set pcolor 37 set type_container 10 set %productivity−container Tyres−productivity]
  ask patch 0 −8  [set pcolor 67 set type_container 11 set %productivity−container Aquatic−plants−
     productivity]
  ask patch 0 −10 [set pcolor 77 set type_container 12 set %productivity−container Glass−items−
     productivity]
  ask patch 0 −12 [set pcolor 87 set type_container 13 set %productivity−container Other−productivity]

  ]
 [
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor gray set type_container 1 set %productivity
     −container Bromeliads−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor orange set type_container 2 set %
     productivity−container Ships−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor brown set type_container 3 set %
     productivity−container Water−tanks−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor lime set type_container 4 set %productivity
     −container Drains−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor turquoise set type_container 5 set %
     productivity−container Sanitation−fixtures−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor cyan set type_container 6 set %productivity
     −container Concrete−structures−productivity ]]

  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor sky set type_container 7 set %productivity
     −container Plastic−items−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor 7 set type_container 8 set %productivity−
     container Metallic−items−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor 27 set type_container 9 set %productivity−
     container Vases−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor 37 set type_container 10 set %productivity
     −container Tyres−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor 67 set type_container 11 set %productivity
     −container Aquatic−plants−productivity ]]
  repeat 1 [ask patch random−pxcor random−pycor [ set pcolor 77 set type_container 12 set %productivity
     −container Glass−items−productivity ]]

  if has−other−category
  [repeat 1 [ask patch random−pxcor random−pycor [ set pcolor 87 set type_container 13 set %productivity
     −container Other−productivity ]]]
 ]

 ask patches [set num_pupae 0]


end

to inc−age
  set age (age + 1)
end

to go
    if (ticks = simulation−steps ) [stop]
    tick
    ask mosquitoes
      [ update−mosquito ; Update the state of mosquito
        move_mosquito
        if ( age > time_pupae_stage ) ; If the mosquito is in the adult stage
         [bite−person
          deposit−eggs ; After bite a person the mosquito deposit eggs
          ]
      ]
    ask people
      [move_person
        inc-age
        ]

  if print-output-file?
  [print-result]

end

;;;;
;;;; MOSQUITO AGENT METHODS
;;;;

to move_mosquito  ;; move the mosquito

 if ( age > time_pupae_stage ) ; If the mosquito is in the adult stage then it can fly.
 [
   rt random 50
   lt random 50
   fd 1

   let x-was-born pxcor-was-born
   let y-was-born pycor-was-born

   let inside_radius false
   ask patch pxcor pycor [
       if distance patch x-was-born y-was-born < radius
         [set inside_radius true]

     ;set pcolor white - for debug just to see if the move is right
     ]
    if (not inside_radius)
     [ fd -1 ]


 ]
end

to-report mosquito_stage  ;; check if the agent is in the mosquito stage. The mosquito flies just in adult fase.

 ifelse ( age > time_pupae_stage )
    [ report true ]
   [ report false ]
end



to infect_mosquito [m]  ;; infect the mosquito
    ask m [
      set infected? true
      set color green
    ]
end

to setup−population−mosquitoes

  set−default−shape mosquitoes "dot"  ;; TODO Trocar dot por uma imagem de mosquito
  let cont_prod 0   let x 0  let y 0

  foreach [1 2 3 4 5 6 7 8 9 10 11 12 13] [
    let p [self] of patches with [type_container = ?] ;para pegar a colecao de patches
    foreach p [
    let population−mosquito−per−container  0
    let xx 0
    let yy 0
    ask  ? [
;       print self
    set x (get_pxcor_container_productivity )
    set y (get_pycor_container_productivity )

    set xx x
    set yy y
;     print yy
    set cont_prod (get_container_productivity )

   set population−mosquito−per−container  (population−mosquito ∗ cont_prod / 100)
    ]
 ;        print population−mosquito−per−container
;   print population−mosquito−per−container print x print y


   ; CREATE MOSQUITO MALE − we consider 1:1 the relation between male and female mosquitoes.
   let population−mosquito−male round ( population−mosquito−per−container / 2 )
  ;print "ser"
  create−new−mosquitoes population−mosquito−male xx yy yellow false false true ;; Create male
      mosquitoes. These mosquitoes are not infected, cannot transmit and neither be infected by dengue
      because they dont bite to people.

  ; CREATE MOSQUITO FEMALE NO INFECTED
  let population−mosquito−female round (population−mosquito−per−container / 2)



  let population−mosquito−female−no−infected population−mosquito−female −( round ( (population−
      mosquito−female ∗ initial−infected−mosquito ) / 100))
  create−new−mosquitoes population−mosquito−female−no−infected xx yy yellow true false true ;;
      Create female mosquitoes no infected mosquitoes and initialize their variables

  ; CREATE MOSQUITO FEMALE INFECTED
  let population−mosquito−female−infected  round(population−mosquito−female − population−
      mosquito−female−no−infected)
  if population−mosquito−female−infected = 0
  [set population−mosquito−female−infected 1] ; At least one mosquito must be infected per container
  create−new−mosquitoes population−mosquito−female−infected xx yy green true true false ;; Create
      infected female mosquitoes and initialize their variables
  ]
 ]

end

to create−new−mosquitoes [number_new_mosquitoes x y my_color is_female? is_infected?
      is_extrinsic_incubation?] ;; Create mosquitoes

 create−mosquitoes number_new_mosquitoes  ;; Create mosquitoes with different parameter
 [
   setxy x y
   set color my_color
   set infected? is_infected?
   set age 0 ;; it is egg
   set extrinsic_incubation? is_extrinsic_incubation? ;; if false then the mosquito can transmit the virus
   set batch 0
   set has−to−deposit−eggs? false
   set pxcor−was−born x
   set pycor−was−born y

;    set age (random ( (56 ∗ day) − (42 ∗ day) + 1)) + (42 ∗ day) ;; (56 days − 42 days + 1) + 42 days
;    set age (random ( 56 − 42  + 1) + 42 ) ;; (56 days − 42 days + 1) + 42 days

   set radius−flight radius ;
  set female? is_female?
  if not is_female?
    [set number−deposited−eggs 0]


  set already−counted−as−egg? false
  set already−counted−as−larvae? false
  set already−counted−as−pupae? false
  set already−counted−as−adult? false
  set my_lifespan −1 ;; Indicate that the lifespan of mosquito MUST BE generated
 ]
end

to create−new−eggs [number_new_eggs x y my_color is_female? is_infected? is_extrinsic_incubation?] ;;
    Create eggs

 hatch−mosquitoes number_new_eggs  ;; Create mosquitoes in the egg stage with different parameter
 [
  setxy x y
  set color my_color
  set infected? is_infected?
  set age 0 ;; It is an egg mosquito
  set extrinsic_incubation? is_extrinsic_incubation? ;; If false then the mosquito can transmit the virus
  set batch 0
  set has−to−deposit−eggs? false
  set pxcor−was−born x
  set pycor−was−born y

  set radius−flight radius ;
  set female? is_female?
  if not is_female?
    [set number−deposited−eggs 0]

  set already−counted−as−egg? false
  set already−counted−as−larvae? false
  set already−counted−as−pupae? false
  set already−counted−as−adult? false

  set my_lifespan −1 ;; Indicate that the lifespan of mosquito MUST BE generated
 ]
end


to bite−person  ;; Mosquito bite a person
```

```
 if female? and not has−to−deposit−eggs? ; Just the female mosquito that bites the person and the
      mosquito does that just
 [
   let p one−of people−here ;; Person is in same patch of mosquito so the mosquito bite the person
   let m self ;; Mosquito
   if p != nobody
   [
     ask p [
      update−person
      ifelse [ infected? and not extrinsic_incubation?] of m ;; If the mosquito is infected and it is not in
      extrinsic incubation stage.
      [ infect_person]
      [ if [not infected?] of m ;; Else if the mosquito is not infected
        [
          if infected? and not intrinsic_incubation? ;; But if the person is infected and it is not in intrinsic
      incubation stage.
          [ infect_mosquito m]
        ]
       ]
       ask m [ set has−to−deposit−eggs? true] ;; If the mosquito bite a person then it has to deposit eggs
     ]
   ]
  ]
end

to update−mosquito
 inc−age

 foreach [1 2 3 4 5 6 7 8 9 10 11 12 13] [ account−stages ? ]

 if (infected? and extrinsic_incubation?)
 [
   if ( age > time_min_extrinsic_incubation )
     [ set extrinsic_incubation? false ]
   ]

 if color = violet and age > time_pupae_stage
 [set color yellow]

 if (my_lifespan = −1)
 [set my_lifespan (random ( time_total_lifespan_max − time_total_lifespan_min  + 1) +
      time_total_lifespan_min ) ]
 if ( age > my_lifespan )  ;; the mosquito must die. Sorry.
 [ set number_mosquito_die number_mosquito_die + 1
   die
   ]
end

to account−stages [c]

 if age < time_pupae_stage and (xcor = pxcor−was−born) and (ycor = pycor−was−born) and
     type_container = c
 [
  if (not already−counted−as−egg?) and (age <= time_egg_stage)
  [set num−egg num−egg + 1
   set already−counted−as−egg? true]

  if (not already−counted−as−larvae?) and (age > time_egg_stage and age <= time_larvae_stage)
  [set num−larvae num−larvae + 1
   set already−counted−as−larvae? true]

   if (not already−counted−as−pupae?) and (age > time_larvae_stage and age <= time_pupae_stage)
   [set num_pupae num_pupae + 1
    set already−counted−as−pupae? true]

   if (not already−counted−as−adult?) and (age > time_pupae_stage)
   [set num_adult num_adult + 1
    set already−counted−as−adult? true]
   ]
end


to deposit−eggs ;; The mosquito deposit eggs

 if female? and look−another−container
 [
   if  batch < 5 ; Number of batch of mosquito in the life
   [
    if (type_container = 1 or type_container = 2 or type_container = 3 or type_container = 4 or
     type_container = 5 or type_container = 6 or type_container = 7 or
       type_container = 8 or type_container = 9 or type_container = 10 or type_container = 11 or
     type_container = 12 or type_container = 13 ) ; check if the patch is a container.
    [
     set positive? true ; The container now is a breeding site

     let eggs number−eggs−for−this−container
```

```
        create−new−eggs eggs / 2 pxcor pycor violet false false true ;; Create male mosquitoes in the egg
        stage
        create−new−eggs eggs / 2 pxcor pycor violet true false true ;; Create female mosquitoes in the egg
        stage
        set total_number_eggs_deposited_environment ( total_number_eggs_deposited_environment + avg−
        eggs−per−batch )

        ifelse number−deposited−eggs + 1 > avg−eggs−per−batch
        [set has−to−deposit−eggs? false
         set number−deposited−eggs 0
         set batch batch + 1 ]
        [set has−to−deposit−eggs? true]
      ]
    ]
  ]
end

to−report look−another−container ;; The mosquito must looking for by other container

  report has−to−deposit−eggs? and ( number−deposited−eggs < avg−eggs−per−batch )
end

to−report number−eggs−for−this−container ;; The mosquito deposits one part of its eggs in different
        containers

 let number−eggs random−normal ( avg−eggs−per−batch / min_number_container_deposit ) ( std−eggs−
        per−batch / min_number_container_deposit )

 if number−deposited−eggs + number−eggs > avg−eggs−per−batch
 [ set number−eggs avg−eggs−per−batch − number−deposited−eggs ]

 set number−deposited−eggs  ( number−deposited−eggs + number−eggs) ;; NUmber of eggs deposited in
        this batch

 if productivity−container? ;; Check if the model will consider the productivity of the container.
 [set number−eggs round(number−eggs ∗ %productivity−container / 100)] ; This is the effective number of
        eggs that will survive in accord with container productivity
;;TODO tirar esse show
;; show  number−deposited−eggs  write "+" show number−eggs
 report number−eggs

;  report number−deposited−eggs < eggs−per−batch )
end
```

```
;;;;
;;;; PERSON AGENT METHODS
;;;;

to move_person  ;; move the person
  rt random 50
  lt random 50
  fd 1
end

to infect_person  ;; Infect the person
  set infected? true
  set color red
  set shape "face␣sad"
end

to update−person
  if (infected? and intrinsic_incubation?)
  [
    if ( age > time_min_intrinsic_incubation )
      [ set intrinsic_incubation? false ]
    ]
end

to setup−population−people

  set−default−shape people "face␣happy" ;; TODO Trocar por uma imagem de apropiada
  let population−no−infected population−person −((population−person ∗ initial−infected−person ) / 100)
  create−people population−no−infected  ;; create the people no infected and initialize their variables
  [
    setxy random−xcor random−ycor
    set color blue
    set infected? false
    set intrinsic_incubation? true ;; the person can not transmit the virus
    ;;TODO colocar uma idade randomica na criacao desse mosquito
  ]

  set−default−shape people "face␣sad" ;; TODO Trocar por uma imagem de apropiada
  let population−infected (population−person − population−no−infected)
  create−people population−infected  ;; create the people infected and initialize their variables
  [
    setxy random−xcor random−ycor
    set color red
    set infected? true
    set intrinsic_incubation? false ;; the person can transmit the virus
    ;;TODO colocar uma idade randomica na criacao desse mosquito
  ]

end



;;;;
;;;; REPORTS
;;;;
to-report quantity_people_infected
  ifelse any? people
    [ report (count people with [infected?])]
    [ report 0 ]
end

to-report quantity_people_no_infected
  ifelse any? people
    [ report (count people with [not infected?])]
    [ report 0 ]
end



to-report quantity_people
  ifelse any? people
    [ report (count people)]
    [ report 0 ]
end

to-report quantity_mosquito_infected
  ifelse any? mosquitoes
    [ report (count mosquitoes with [infected?])]
    [ report 0 ]
end

to-report quantity_mosquito_no_infected
  ifelse any? mosquitoes
    [ report (count mosquitoes with [not infected?])]
    [ report 0 ]
end



to-report quantity_mosquito ;; It reports the total number of agent mosquitoes
```

```
  ifelse any? mosquitoes
    [ report (count mosquitoes)]
    [ report 0 ]
end

to−report quantity_egg_mosquito ;; It reports the total number of the mosquitoes in the egg stage
  ifelse any? mosquitoes
    [ report (count mosquitoes with [age <= time_egg_stage])]
    [ report 0 ]
end

to−report quantity_larvae_mosquito ;; It reports the total number of the mosquitoes in the larvae stage
  ifelse any? mosquitoes
    [ report (count mosquitoes with [age > time_egg_stage and age <= time_larvae_stage])]
    [ report 0 ]
end

to−report quantity_pupae_mosquito ;; It reports the total number of the mosquitoes in the pupae stage
  ifelse any? mosquitoes
    [ report (count mosquitoes with [age > time_larvae_stage and age <= time_pupae_stage])]
    [ report 0 ]
end

to−report quantity_adult_mosquito ;; It reports the total number of the mosquitoes in the adult stage
  ifelse any? mosquitoes
    [ report (count mosquitoes with [age > time_pupae_stage])]
    [ report 0 ]
end

to−report pupae_per_container [c]
  let sum_num_pupae  0;
  ask patches with [type_container = c]
   [set sum_num_pupae sum_num_pupae + num_pupae]

  report sum_num_pupae
end

to−report total_pupae_per_container
  let sum_num_pupae  0;
  ask patches [set sum_num_pupae sum_num_pupae + num_pupae]

  report sum_num_pupae
end
```

```
;to−report get_container_productivity [c]
;  let productivity  0;
;  ask patches with [type_container = c]
;  [set productivity %productivity−container]
;
;  report productivity
;
;end

to−report get_container_productivity
  let productivity %productivity−container
  report productivity

end



;to−report get_pxcor_container_productivity [c]
;  let x 0;
;;  let c patches with [type_container = c] para pegar a colecao de patches
;  ask patches with [type_container = c]
;  [  set x  [pxcor] of self ]
;
;  report x
;end

to−report get_pxcor_container_productivity
  let x  [pxcor] of self

  report x
end

;to−report get_pycor_container_productivity [c]
;  let y 0;
;  ask patches with [type_container = c]
;  [set y [pycor] of self]
;
;  report y
;end



to−report get_pycor_container_productivity
  let y [pycor] of self
  report y
end
```

```
to print−result
  if (ticks = simulation−steps ) ;; If the ticks is the last step then prints the simulation results.
                               ;; The format of file is: Number of container, number of pupae per container, total
      number of pupae, pupae per container, pupae per person,
                               ;; number of infected person and the number of simulation steps
  [ file−open "result.txt"
    foreach [1 2 3 4 5 6 7 8 9 10 11 12 13] [file−write ? file−write pupae_per_container ?]
    file−write total_pupae_per_container
    file−write total_pupae_per_container / 13 ; 13 is the number of container
    file−write total_pupae_per_container / population−person
    file−write quantity_people_infected
    file−write ticks
    file−print ""
    file−close]
end
```