Use of Graphics Processors for Agent-Based Simulations
A case study using Heatbugs model

Luis Jorge Antunes Ramalho
Supervisor: Prof. Agostinho Rosa
Co-Supervisor: Dr. Nuno Fachada
Instituto Superior Técnico
Portugal

November 26, 2018

Abstract

Agent-based modelling is a programming and simulation paradigm. It is a bottom-up approach where interacting agents are its basic building blocks. However, ABMs must undergo a process of independent implementation and verification to achieve credibility, where different replications of an ABM should provide similar statistical results. Yet the reality is quite different and too often we observe many replications of a given model presenting wildly different results. This may happen because ABMs are very sensitive to implementation details, and the lack of a complete and formal model description often leaves researchers with the burden of creating implementations based on personal interpretations of the model.

In this work we provide a complete description of the Heatbugs model based on its NetLogo implementation. To accomplish this, we use the ODD protocol, which allows us to formalize the model so that future independent replications are able to yield similar results. In addition, we provide results analysis for performance and statistical alignment for two Heatbugs model replications: 1) a single threaded CPU implementation in C; and, 2) a parallel GPU implementation in OpenCL.

Keywords Heatbugs, Agent-based Modelling, ABM, ODD protocol, parallel simulation, GPU simulation, OpenCL, model alignment, PCA.

1 Introduction

Agent-based models (ABMs) are a way to describe systems in terms of interactions between agents. We use ABMs to try and replicate the same complexity and adaptations of natural and social systems in computer environments and thus to try and study their evolution over time. These are stochastic models with a bottom-up approach where interacting agents are its basic building blocks. The agents are independent entities, to which specific characteristics and properties are given, and which are programmes in order to show specific behaviours and make decisions.

In ABMs, the systems global behaviour results from the collective action of many individuals (agents), which are guided by a very simple set of rules to organize themselves locally in order to perform tasks without a centralized control mechanism [8]. This emerging complex behaviour may be used in simulations to produce experience results similar to those that would on occur in natural conditions.

Agent-based modelling is under a growing boost motivated by sectors such as the pharmaceutical industry [19], biomedicine [3], [8], public health [5], [22], and by the growing need to rationalize resources in the occurrence of pandemics. In the latter cases, we combine spatial agent-based models (SABM) with geographical information systems (GIS) [1].

Platforms such as MASON [23], Swarm [25], Repast [26] and NetLogo [33], were especially designed for the development and simulation of ABMs, which are traditionally implemented with serialized models, frequently based in discrete spaced agents [23], [25]. The simulations of models with continuous spaced agents or with thousands of agents requires that the maximum is processed per time step [5]. The high processing demand of such models [15] exposes performance problems of the traditional simulation platforms due to an architecture that remains without being parallelized, except for Swarm [30], and executes its serial processes in the CPUs [5], [6].

In this context, the use of graphics processing units (GPUs) to simulate ABMs remains an under-explored niche, in spite of their growing demand, evidenced by the increase in the number of projects based in high performance processing developed in this architecture. GPUs have hundreds of processing elements (PEs) capable of executing code in parallel, a characteristic which may be used as an answer to the ABMs simulations scheduling and performance issues.

ABMs are sensitive to implementation details. Certain development decisions or the unintended inclusion of changes in the implementation of a model may change its dynamics and produce incorrect results [24], [35]. This issue is more relevant when the intention is to check a model independently, as literature often does nor exist both regarding to the detailed description of the model and the methodology used for the analysis of the results.

The ODD (Overview, Design concepts, Details) protocol is a frequently used tool for the detailed descriptions of the characteristics of any given agent-based model [14].

We propose to use a relatively simple ABM called Heatbugs to investigate the viability of its implementation in GPUs. Some goals have been established, one of which intends to fill a gap in the Heatbugs models documentation: a) to perform a complete and detailed description of the Heatbugs model using the ODD protocol; b) to investigate potential performance improvements of an implementation of the Heatbugs model in GPU, when compared to the implementations in CPU; and, c) to analyse the alignment of the models implementations through results comparison automatic tools.

We propose three implementations of the Heatbugs model: a) the existing implementation in NetLogo, changed in order to allow the collection of data. This is the reference implementation that serves as a comparison for the remaining implementations; b) an implementation in C, which runs in CPU in a single thread; and, c) an implementation in OpenCL used to test the viability of using the GPUs to simulate ABMs.

2 Methodology

2.1 Overview, concepts and details of the Heatbugs model

In the Heatbugs model [13], [34], agents with specific internal characteristics move between the heat cells in a virtual environment. The heat of the cells varies with time according to the environment's parameters and to the agents movement. The cells propagate heat through in the environment, which in turn loses part of that heat in a dynamic regulated by the heat equation, described in section 2.5. On the other hand, every cell in the path of an agent receives heat deposited by it.

The path of the agent through the cells is guided by its ideal temperature, i.e. that at which the agent prefers to be in order to minimize its degree of unhappiness. At a certain interaction, the state of the system is the result of the agents unhappiness average.

2.1.1 Goal

The goal of the model is to study how the changes in the environment condition the action of the agents. The dynamic of the Heatbugs system is the exclusive result of the agent-environment interaction, as the agents do not interact directly between themselves. This model captures some characteristics which are common to some ABMs, such as the search for a new location, preventing collisions and movement of the agents. The implementations of the Heatbugs model may differ among themselves in terms of choice of the system's architecture, the modelling platform, the programming languages or the selection of the random number generator. Nevertheless, they show similar statistical results, thus ensuring the equivalence of the implementations.

2.2 Entities, state variables and scales

Variables that depend on time and state variables are represented by an upper case letter, while time-independent parameters and variables are represented by a lower case letter.

A parameter is a variable that, upon being started, does not change during the execution of the simulation.

Parameters are used to establish the characteristics of the environment and of the agents and are provided at the time of simulation invocation.

The expression $U(a,b)$ produces a random value in the $[a,b[$ interval obtained from a uniform distribution.

The parameter n is the number of interactions and it indicates how many times the model's main cycle is executed. We admit time as being a discrete variable and that each time step is an interaction of that cycle.

The Heatbugs model is composed by three entity classes: *agentes*, *cells* e *environment*. Each entity class is defined by a set of state variables, as shown in the Table 1.

Except otherwise indicated, when mentioned in this section, the entities *agents*, *cells* and *environment*, are to be considered as being instances of their classes.

Class environment

There is only one instance of the Environment class which corresponds to a *toroidal square grid*, defined by dimensions (x_a, y_a), encompassing all the cells. In this environment, the parameter a represents the number of agents scattered among the cells.

The parameter d is the Heat Diffusion Rate representing the amount of heat lost by a cell and distributed among the adjacent cells, per time step. The parameter e represents the Evaporation Rate or heat loss of the cells, per time step.

The parameters t_{min} and t_{max} represent the preferential Minimum and Maximum Ideal Temperatures of the agents. The parameters c_{min} and c_{max} represent the minimum and maximum heat that an agent may release to the environment. These parameters are used to set the agents state variables.

Class cells

Each cell is conceptually perceived as being a square with a fixed position in the environment. The parameter x represents the Horizontal Position and the Parameter y represents the Vertical Position of the cell within the environment.

The state variable T represents the temperature of a cell and its value depends on the result of the agent-cell and cell-cell interactions, the latter being according to the environment characteristics (the Diffusion and Evaporation Rates).

An occupied cell may only have one agent. The state variable A is an indicator of the cells occupation state. If $A = 0$ then there is no agent in the cell, however, if $A = 1$ then the cell is occupied by an agent.

Class agents

Agents are independent individuals with similar behaviours, yet with overall specific intrinsic characteristics such as distinct requirements for the ideal temperature and different values for the heat released.

The state variable t is the ideal temperature in the cell minimizing the unhappiness of each agent, and the state variable c is the released heat that the agent deposits in the cell.

The state variable I is the agent's unhappiness degree indicating *when* it needs to move. Its value is the result of the difference between the occupied cell's temperature and the agent's ideal temperature, $I = |T - t|$.

The parameter p is the Random Movement Probability. This parameter guides the process of searching a new cell to where the agent might move into when it is unhappy ($I > 0$) (as seen in section 2.5).

The pair of state variables (X, Y) indicates the position occupied by the agent within the environment. The agent will occupy a (x, y) cell if $X = x$ and $Y = y$.

The agents do not reproduce themselves and, once instantiated, shall exist during the entire simulation.

2.3 Processes and Scheduling

A general perspective of the simulation's scheduling and of the related processes is presented in algorithm 1.

The programme starts with process `Init()`, which initializes the environment and randomly distributes the agents among the cells. Upon initialization, procedure `GetResult()` obtains the agents *unhappiness* at interaction zero.

The algorithm's main cycle executes the processes `Diffuse()`, `Evaporate()` and `AgentStep()` described in section 2.5 n times. At the end of each interaction, the process `GetResult()` is executed to obtain the new agent's unhappiness.

2.4 Design Concepts

Basic Principles The Heatbugs model is a simplified reproduction of some patterns observed in nature. The environment temperature is a metaphor used to model natural conditions such as stagnant water, which may accumulate or evaporate, but the existence of which

Entity	State Variable	Symbol	Value Range
Environment	Horizontal Dimension	x_a	$1, 2, ...$
	Vertical Dimension	y_a	$1, 2, ...$
	Diffusion Rate	d	$[0, 1]$
	Evaporation Rate	e	$[0, 1]$
	Minimum Ideal Temperature	t_{min}	$0, 1, ..., 200$
	Maximum Ideal Temperature	t_{max}	$0, 1, ..., 200$
	Minimum Released Heat	c_{min}	$0, 1, ..., 100$
	Maximum Released Heat	c_{max}	$0, 1, ..., 100$
	Number of Agents.	a	$1, 2, ..., x_a \cdot y_a$
Cells	Horizontal Position	x	$0, 1, ..., x_a - 1$
	Vertical Position	y	$0, 1, ..., y_a - 1$
	Cell Temperature	T	$[0, \infty[$
	Occupied by Agent	A	$0, 1$
Agents	Ideal Temperature	t	$0, 1, ..., 200$
	Released Heat	c	$0, 1, ..., 100$
	Random Move Prob. (%)	p	$0, 1, ..., 100$
	Unhappiness	I	$[0, \infty[$
	Horizontal Position	X	$0, 1, ..., x_a - 1$
	Vertical Position	Y	$0, 1, ..., y_a - 1$

Table 1: Table containing agents classes and their variables.

Algoritmo 1 Model's main cycle. At the first interaction, the environment has a temperature of zero until the execution of the process AgentStep() where the agents add heat to the environment.

```
1:  Init()
2:  GetResult()                          ▷ first result
3:  $i \leftarrow 1$
4:  while $i <= n$ do
5:      for all cells do
6:          Diffuse()
7:      end for
8:      for all cells do
9:          Evaporate()
10:     end for
11:     for all agents do                ▷ any order or parallel
12:         AgentStep()                  ▷ Compute unhappiness
13:     end for
14:     GetResult()                      ▷ $n$ results
15:     $i \leftarrow i + 1$
16: end while
```

influences the actions of some insects. The model is designed to show the agent's reaction to localized variability of environment characteristics.

In addition, the "search for food" model in the context of robot swarms shows similarities to those of the Heatbugs model. The same diffusion and evaporation processes are applied to pheromones, which is a characteristic of the environment manipulated by the agents and that act as motivator for their movement [16].

We are able to observe that some processes are transversal to multiple models, therefore, the Heatbugs model's study may represent a valuable source of information for the implementation of models in several architectures.

Emergent Behaviour In high temperature environments, the agents move around while keeping a great distance between themselves, but in low temperatures environments (i.e. with high rates of evaporation), the agents move around less frequently and are more willing to form groups to benefit from the heat released and diffused locally by the environment.

Perception The agents may feel the presence of other agents and will not move to already occupied cells. Agents also perceive the temperature in their Moore Neighbourhood with a radius of one.

Interaction The agents do not interact between themselves. Their interaction only occurs with the environment under the form of heat added to the occupied cells.

Stochasticity Agents are randomly positioned within the environment and their ideal temperature t and released heat c are randomly established from minimum and maximum values, as described in the Initialization paragraph (bellow).

In each intersection, the scheduling of the agents is random, i.e. none of the agents has always the priority to determine to which cell it will move into.

During the movement of the agents, the selection of cells may occur randomly if $p > 0$, if there are several cells with the same value for the same temperature or if the chosen cell is occupied and the agent needs to chose again.

Observation The process `GetResult()` calculates the unhappiness average of the agents and stores the result. That is the only output of the system.

2.4.1 Initialization

Upon initialization, $x_a \times y_a$ cells are created. Every cell is initialized with $T = 0$ and $A = 0$.

The total of a agents are instantiated. The position of each agent within the environment is randomly determined by the initialization of the parameters pair (X, Y), where $X = U(0, x_a)$ and $Y = U(0, y_a)$, and their state variables are also initialized: the ideal temperature $t = U(t_{min}, t_{max})$, the released heat $c = U(c_{min}, c_{max})$ and the initial unhappiness value $I = t$. The state of the cell occupied by this new agent is changed to occupied $A = 1$.

As each cell may only contain one agent, it is necessary to ensure that each (X, Y) pair is unique.

2.5 Submodels

The procedures `Diffuse()` and `Evaporate()` change the environment temperature by scattering and removing heat from it. These procedures implement the heat function represented in the equation 1.

$$h_i(x,y) = \Bigg[h_{i-1}(x,y) \cdot (1-d) + \frac{1}{8} \cdot d \cdot \sum_{(u,v) \in N(x,y)} h_{i-1}(u,v)\Bigg] \cdot (1-e) \tag{1}$$

where $h_i(x, y)$ is the (x, y) cell's heat at interaction i, after the execution of the diffusion and evaporation processes, $h_{i-1}(x, y)$ is the cell's heat at interaction $i - 1$ and $h_{i-1}(u, v)$ is the heat of a neighbour cell at interaction $i-1$. The parameter d (diffusion rate) indicates how much of its heat a cell diffuses to the neighbouring cells, and the parameter e (evaporation rate) the heat amount a cell loses.

The procedure `AgentStep()` may change the agent position and add heat to the environment.

Diffuse() Diffusion reduces the temperature spikes by scattering heat among the cells. However, it has no effect when occurring at the first iteraction, after initialization, as every cell in the environment has zero temperature, therefore, no energy is diffused from any cell to its neighbours. In any other $(i > 0)$ iteraction, part of the cells has a temperature of $T > 0$ and diffusion scatters heat among the neighbouring cells.

Evaporate() Evaporation causes loss of heat by the environment. Every cell loses heat and, if no repositioning happens, their temperature returns to the initial state $T = 0$.

AgentStep() The agent's movement is performed by the process `AgentStep()`, which consists on the sequence of steps each agent executes according to the following order: a) update the unhappiness value; b) search for a destination cell in the neighbour cells; c) if possible, move to the neighbour cell; and, d) add heat to the destination cell. This sequence of steps is detailed in the algorithm 2.

If $p > 0$ the probability of random movement exists and, in this case, the sub-process `GetAnyNeighCell()` return a randomly chosen cell.

The sub-process `GetMaxTempMCell()` return one of the neighbour cells with maximum temperature or the

Algoritmo 2 Agent Step.

```
procedure AgentStep()
    $I \leftarrow |T - t|$                                ▷ unhappiness
    if $I > 0$ then
        if $U(0,100) < p$ then                       ▷ Random move?
            $newCell \leftarrow$ GetAnyNeighCell()
        else if $T < t$ then                         ▷ Agent is cold?
            $newCell \leftarrow$ GetMaxTempMCell()
        else                                         ▷ Agent is hot?
            $newCell \leftarrow$ GetMinTempMCell()
        end if
        if $newCell \neq currentCell$ AND NOT Empty($newCell$) then
            $newCell \leftarrow$ GetAnyFreeNeighCell()
        end if
        if $newCell \neq currentCell$ then
            MoveAgentTo($newCell$)
            $currentCell \leftarrow newCell$         ▷ update agent position
        end if
    end if
    AddAgentHeat($currentCell$)                      ▷ Always add heat to current cell
end procedure
```

current cell if there is no neighbour cell with higher temperature.

The sub-process `GetMinTempMCell()` return one of the neighbour cells with minimum temperature or the current cell if there is no neighbour cell with a lower temperature.

The sub-process `GetAnyFreeNeighCell()` return any *free* neighbour cell. This sub-process is executed when the agent has found out, during the first movement attempt, that the previously considered cell was occupied.

Parametrization We propose two scenarios corresponding to two different parametrizations for the performance and alignment tests. The fixed parameters regarding dynamic are presented in Table 2. The dimension of the environment (parameters x_a and y_a) and the number of agents a are variable parameters that may modify the dynamic of each scenario.

In order to control the changes to the system dynamic within a single scenario, we introduced the meta-parameter named *agent density* (ρ), where $\rho = a/(x_a \times x_b)$. By maintaining the value of ρ constant, the number of agents is proportional to the dimension of the environment. This way, it is possible to test the performance of a processing system by using the same scenario with progressively higher values of the environment's dimension.

We propose 5 sizes per scenario and an agent density of $\rho = 0.01$ for scenario 1 and of $\rho = 0.05$ for scenario 2. These values, along with the associated number of agents in each scenario are shown in Table 3.

By choosing the above selected parameters, we intended to get quite different dynamics, which strengthens the later verification of model's replications and ensures the confidence degree in them, which is something that is possible to achieve if the model is executes with distinct parameters [7].

2.6 Heatbugs Model Dynamic

The traditional outputs of the simulation, representing the system dynamic for scenarios 1 and 2, with the size 100×100, are shown in image 1. The output of the system depends on the agent's unhappiness average.

The dynamic of the system depends on the parametrization and is independent of the environment size, as long as the agents density is constant (as mentioned before).

3 Model Comparison

The models were compared using the independent model technique proposed by Fachada et al. [11]. This technique is automated [12] thus simplifying the selection of focal measurements (FM) and the following analysis of the results. Some of the advantages include: a) the selection of FMs is obtained directly from the outputs; b) automatic selection of the characteristics that better explain potential differences between implementations; c) it does not depend on the properties of the distribution of outputs [11].

This method uses the Principal Component Analysis (PCA) [9], [20] to convert time series representing the output in a set of linear independent measurements, i.e. Principal Components (PCs), which may be analysed and compared [11] by multivariate statistical methods such as MANOVA [21], [32].

The score plot (Image 2) showing the positioning of each replication by using the first two PCs as axis. Two

Parameter	Symbol	Scenario1	Scenario2
Diffusion Rate	d	0.9	0.4
Evaporation Rate	e	0.01	0.03
Minimum Ideal Temperature	t_{min}	10	20
Maximum Ideal Temperature	t_{max}	40	30
Minimum Released Heat	c_{min}	5	15
Maximum Released Heat	c_{max}	25	25
Random Movement Prob.	p	0	0

Table 2: Parameters regarding the dynamic of each scenario.

Environment			No. Of Agents	
Size	$x_a \times y_a$	No. of Cells	Scenario 1 $\rho = 0.01$	Scenario 2 $\rho = 0.05$
100	100×100	10 000	100	500
200	200×200	40 000	400	2000
400	400×400	160 000	1600	8000
800	800×800	640 000	6400	32 000
1600	1600×1600	2 560 000	25 600	128 000

Table 3: Sizes of the model, associated number of cells and the corresponding number of agents per scenario. In scenario 1 we have $\rho = 0.01$ and in scenario 2 we have $\rho = 0.05$.

aligned models in which their outputs result from the same processes, show a score plot with two overlapped colours, as shown in Image 2a. On the other hand, a non-aligned model shows a score plot where dots of the same colour are grouped in different regions or there is a partial overlapping of both groups, as shown in Image 2b.

The performed statistical comparison using mainly MANOVA, usually applied to the set of PCs explaining at least 90 % of the variance ($ve = 0.9$), and also to the t-test [2] [4] applied to each individual PC. Two models may be considered *non-aligned* (i.e. *different*) if the MANOVA and/or the t-test produce a p-value bellow α, α being the level of statistical significance defined *a priori* (usually 1 % or 5 %).

The models comparison tests were performed with the software `micompr` [12] which implements the method described in this section.

4 Results and Discussion

4.1 Experimental Setup

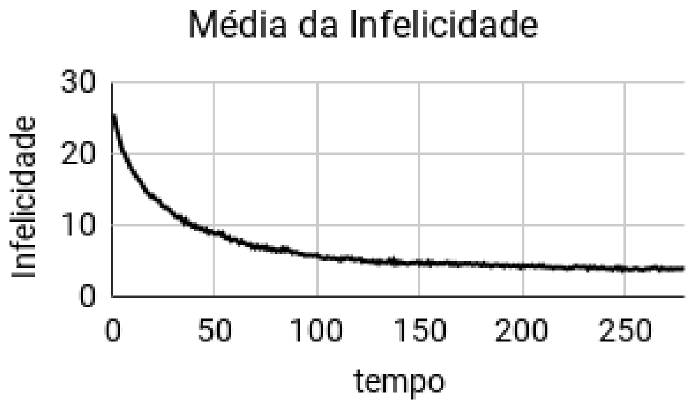
A total of $r = 1, \ldots, 30$ replications was performed for each scenario and model size, with the implementations in NetLogo, C and OpenCL. A PRNG seed was used for each r replication. The seed was obtained through the application of the MD5 dispersion function [31] to the value of r. This procedure ensures a certain level of independence between the generated seeds and the replications.

The sizes selected for the model were 100, 200, 400, 800, 1600 (Table 3). With these values, 30 replications $\times$ 2 scenarios $\times$ 5 sizes, each platform executes 300 replications.

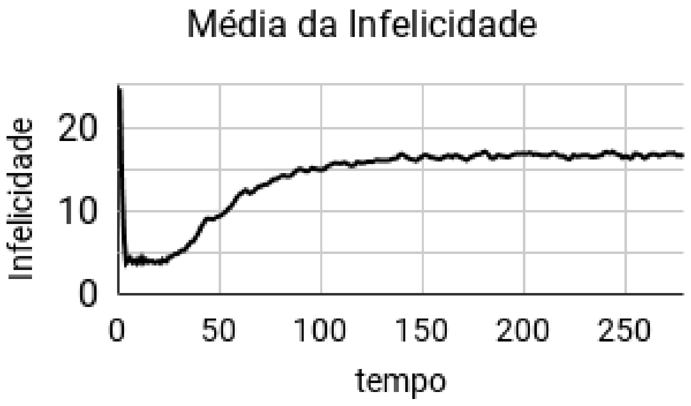
The simulations in NetLogo, C and OpenCL were performed in hardware with the following specifications:

Sistema 1

1. CPU: Intel®Core™ i7-3930K, 3.20GHz - 3.80GHz, 6×cores, 12×threads (two logic



(a) Agents Unhappiness Average, Scenario 1, 100 agents (densidade $\rho = 0.01$).



(b) Agents Unhappiness Average, Scenario 2, 500 agents (densidade $\rho = 0.05$).

Figure 1: Dynamic of the system for two different parametrizations (Scenarios 1 and 2) and environment with a size 100×100.

processors per core), 12MB SmartCache (bus speed 5 GT/s DMI2, max memory bandwidth 51.2GB/s), release Q4'11

2. Memory: 32GB RAM
3. GPU: AMD Radeon HD7970 (Tahiti XT), 3GB RAM @ 1375MHz (bandwidth 264GB/s, 384-bit bus), 2048 SU[1] organized in 32 UC[2] @ 925MHz (64 SU per CU), release *Jan 2012*

Sistema 2

1. CPU: Intel®Xeon™ E5-2650v3, 2.30GHz - 3.00GHz, 10×cores, 20×threads (two logic processors per core), 25MB SmartCache (bus speed 9.6CT/s QPI, max memory bandwidth 68GB/s), release Q3'14
2. Memory: 64GB RAM
3. GPU: Nvidia GeForce GTX 1080 (GP104-400-A1 GP104-410-A1), 8GB RAM @ 1607MHz (bandwidth 320GB/s, 256-bit bus), 2560 SUs organized in 64 UC (40 SU per UC), release *27 May 2016*

4.2 Performance Analysis

The speedup (increase) showing the relative performance of an implementation A of a model against that of an implementation B is achieved through the equation:

$$S_B^A = \frac{T^B}{T^A} \tag{2}$$

where T^A and T^B are the execution times of A and B, respectively, and the S_B^A speedup indicates how fast implementation A is when compared to implementation B.

The images and tables presented bellow were created with the software `PerfAndPubTools` [10].

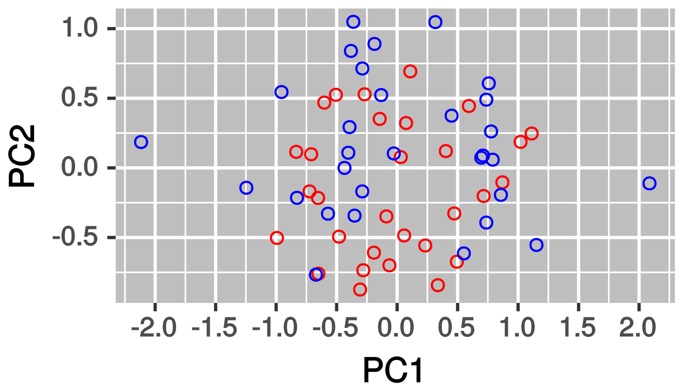
The comparisons between implementations are presented based on their performances in processing system 1, described in the previous section. The processing system 2 was used to compare the OpenCL implementation between different GPUs.

4.2.1 Scenario 1

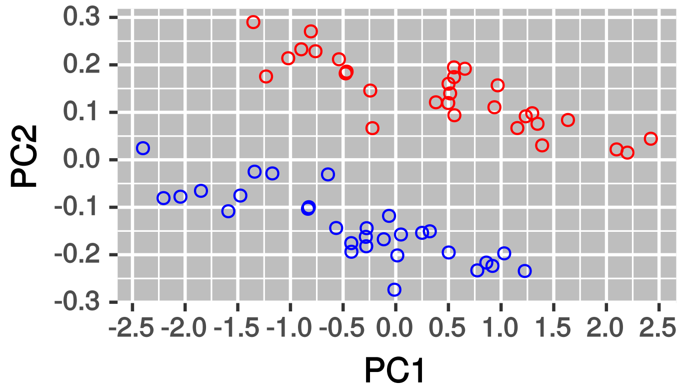
The average execution times and the performance speedup are presented in image 3, and the latter are also detailed in Table 4.

For size 1600, the OpenCL implementation is approximately 1508 times faster than NetLogo (Image 3b), and approximately 34 times faster than C. For that same size, the implementation in C is approximately 44 times faster than NetLogo. The performance speedup in OpenCL and C in comparison to NetLogo is shown in 4.

[1]SU - shader unit
[2]CU - compute unit



(a) Aligned Model.



(b) Non-Aligned Model.

Figure 2: Representation in score lot of the first two principal components of two models.

Size	100	200	400	800	1600
OpenCL	6.2	25.1	171.8	572.5	1508.2
C	26.9	27.2	50.1	50.8	44.1

Table 4: Scenario 1: OpenCL and C speedup using NetLogo as reference.

For sizes up to 200, C is faster than OpenCL. This may be because OpenCL requires preparation time in the CPU, and the kernel's Just-In-Time (JIT) compilation to be sent to the GPU is a lengthier procedure. For small size problems, the kernel compilation time represents a significant part of the simulation total execution time in the GPU.

The OpenCL's execution times have a slow growth, and that can be observed in sizes above 400 (160 000 cells) with 1600 agents. This is the point where the highest load is applied to the processing elements of the GPU, mainly during the calculation of the diffusion. Nevertheless, OpenCL speedup increases with the environment size due to the increasingly worst scheduling of NetLogo's performance (Image 3b).

NetLogo execution times are higher than the execution times of the implementations in C and OpenCL. This is because NetLogo is both an interpreter of *scripting* language and a JIT compilator running in a Java virtual machine (JVM). The weight of such platform, including the required compilation time for bytecode, adds up to the processing load required to execute the simulation. Furthermore, the NetLogo implementation runs with a single thread while in OpenCL the simulation is executed with thousands of threads, one for each agent or environment cell.

The implementation in C is always faster than NetLogo, and the performance speedup in C is relatively even due to the similar scheduling between both implementations.

4.2.2 Scenario 2

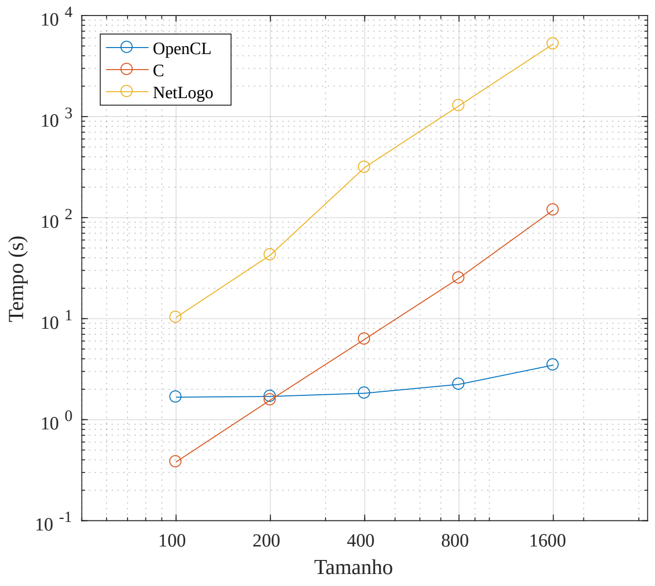
The average execution times and the performance speedup are presented in image 4, and the latter are also detailed in Table 5.

The changes in execution times in scenario 2 are exclusively due to the larger number of agents (5 time larger than in scenario 1), and not to the processing of the environment temperature, which shows the same processing load in both scenarios.

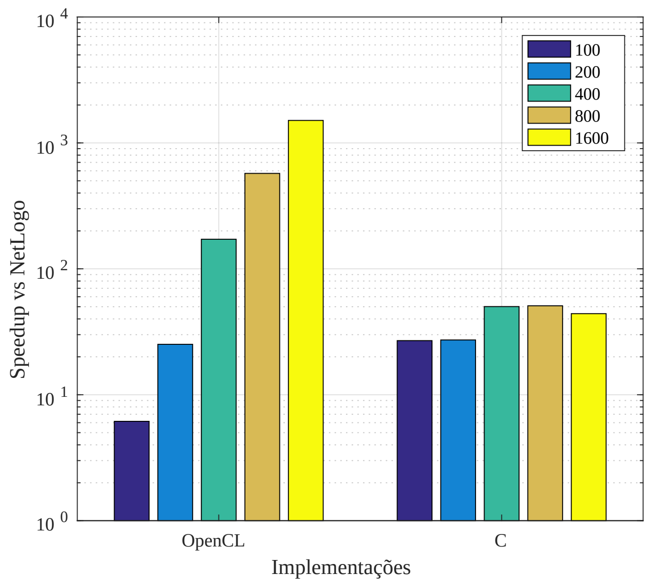
The performance speedup in OpenCL and in C when compared to NetLogo in scenario 2 are presented in Table 5.

For size 1600, we can observe that OpenCL is 1038 times faster than NetLogo and 68 times faster than C. The implementation in C is around 15 times faster than NetLogo, as seen in Image 4b.

We observed that introducing more agents in scenario 2 causes a greater impact in the execution times of C



(a) Average execution times of C, OpenCL and NetLogo's implementations.



(b) Representation of the speedup of the implementations in C and OpenCL in comparison to NetLogo.

Figure 3: Execution times and speedup, scenario 1.

Size	100	200	400	800	1600
OpenCL	6.7	28.8	174.0	545.1	1037.7
C	8.7	9.8	16.8	17.0	15.2

Table 5: Scenario 2: OpenCL and C speedup using NetLogo as reference.

than in the execution times of NetLogo or OpenCL. Taking the example of size 100, for which the execution time of C increased 252.14 %, but with a significantly lower impact in the times of OpenCL (4.42 %) and NetLogo (12.62 %). Nevertheless, for problems of size 100, the implementation in C remains faster than in OpenCL for the same reason previously presented for scenario 1.

Note that the execution time of OpenCL has a higher increase when compared to scenario 1, caused by the high number of agents present in environments of size equal or above 400.

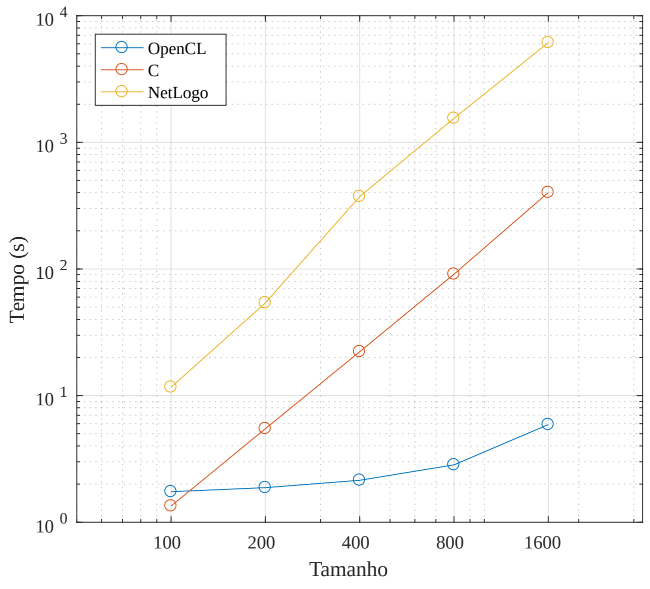
4.2.3 Scenario 1 vs. Scenario 2

The execution times of scenario 1 and scenario 2 overlap in Image 5 and the speedup of the implementations for size 1600 are shown for reference in Table 6.

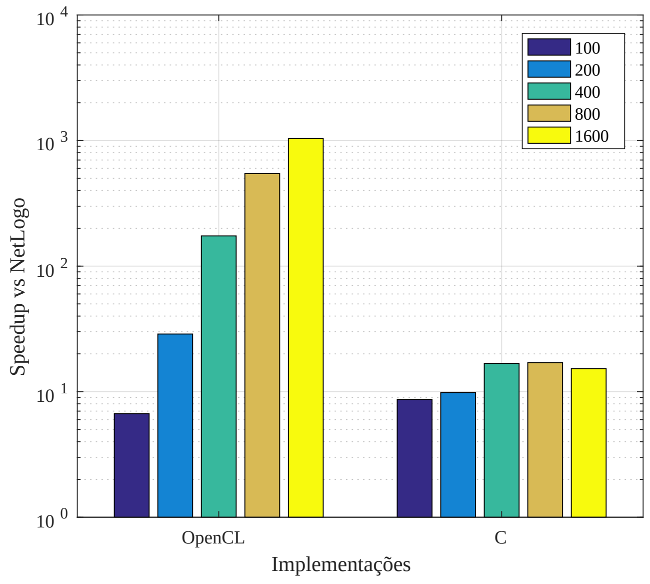
The average execution times in each scenario of the implementations in NetLogo, C and OpenCL, as well as the percentage increase of this time in scenario 2 in comparison to scenario 1 are shown in Table 7.

An analysis to the Image 5 and the Tables 6 and 7 allow the following conclusion:

- The execution times curves of scenarios 1 and 2 are almost exact copies of each other in the case of C and NetLogo.
- The increase of agents in scenario 2 has a negative impact in the execution times of all implementations. This impact is far greater in the implementation in C than in NetLogo, as may be seen in Table 7, in which $\Delta T(\%)$ is the percentage increase of the average execution time in scenario 2 in comparison to scenario 1, for a same environment size.
- As for the distance between the curves of both scenarios, a similar increase was expected in C and in



(a) Average execution times of C, OpenCL and NetLogo's implementations.



(b) Representation of the speedup of the implementations in C and OpenCL in comparison to NetLogo.

Figure 4: Execution times and speedup, scenario 2.

Speedup	Scenario 1	Scenario 2
OpenCL over NetLogo	1508	1038
OpenCL over C	34	69
C over NetLogo	44	15

Table 6: Approximate values of speedup in scenario 1 and scenario 2 for size 1600.

NetLogo. However, in C, scenario 2 has an increase of the execution time above 200 % in comparison to scenario 1, while in NetLogo that increase is lower than 30 %. This difference may be due to the initialization of the simulation, where the instantiation of agents takes as longer as its density is, and it is also necessary to avoid the overlapping of agents during its placement in the environment. In NetLogo the creation of agents is obvious and transparent through the language, eventually optimized during the implementation and compilation of the language into bytecode. In OpenCL the agents are instanced in parallel and any potential overlapping is also treated in parallel, thus the increase of the density of agents between scenarios is not significant in this implementation.

- In OpenCL the differences in execution times between scenarios are minimal and only start to visibly increase from size 400, when the processing load over the GPU increases. It is from that point that the increase in the number of agents causes greater impact, as may be seen in scenario 2.

4.2.4 OpenCL in different GPUs

The execution times of the implementation OpenCL were compared in two different GPU, one AMD Radeon HD7970 and a Nvidia GeForce GTX 1080, and the results are presented in Image 6.

We observe that the Nvidia GPU schedules very well and its performance is consistent for sizes up to 400, from that point, execution times increase. On the other hand, we can see that the older AMD GPU has consistently lower execution times than the more recent Nvidia GPU. This phenomenon is not strange and lit-

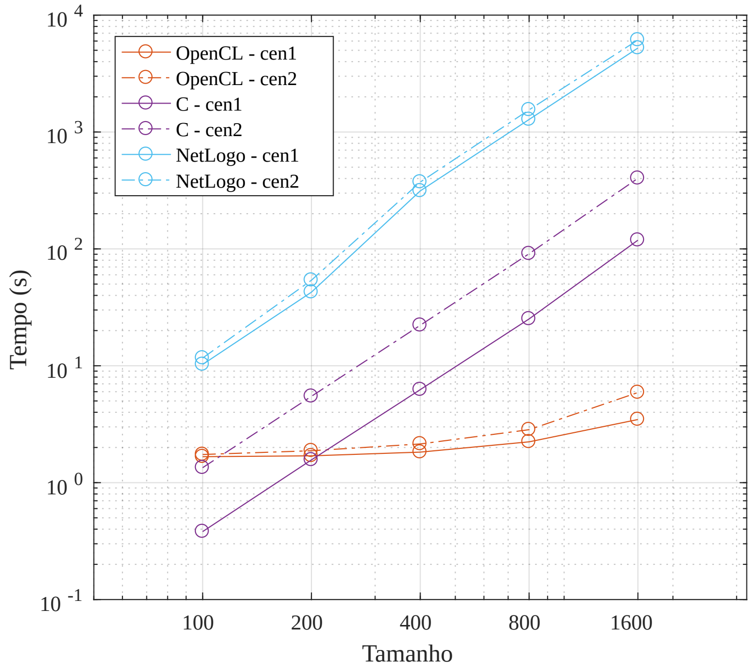


Figure 5: Overlapping of the execution times of scenario 1 and scenario 2.

erature exists to show that the implementation of optimized algorithms for a certain GPU used as reference show irregular performances or they do not run at all, between GPUs of the same manufacturer, between GPUs of different manufacturers and between GPUs of different generations [28]. Note that the implementation in OpenCL of the Heatbugs follow, as much as possible and having into account the complexity of the agent models, the good practices in programming for GPUs, and is not optimized for any specific architecture.

It may be observed in Image 6 that times are closer for larger sizes, and because the Nvidia GPU has an advantage in terms of memory size, bandwidth and number of CUs, it is acceptable to assume that it has an advantage over GPU AMD HD7970 for problems with greater dimensions.

4.3 Model Comparison

4.3.1 Model Alignment Test

Tables 8 to 11 show the statistical results of the comparison between the models. The first column shows the sizes of the model. The second column represents the number of principal components (PCs) that explain at least 90 % of the variance, obtained through the application of PCA on the simulation outputs for each size of the model. The third column, defined as “Tests”, shows the type of test performed on the PCs obtained from the outputs. These tests are:

- MNV (90% var.): p-value of the MANOVA multivariate parametric test applied to the principal components that explain at least 90 % of the variance.
- t-test (CP1): p-value of the t-test applied to the first principal component.
- MW-U test (CP1): p-value of the Mann-Whitney U test applied to the first principal component.

The fourth column shows the results of the above-mentioned tests on the outputs. The fifth column shows the score plot of the first two PCs, where the blue dots are the 30 replications of the implementation in NetLogo, and the red dots are the 30 replications of the implementation in C or in OpenCL.

It is worth noting that the blue dots of the NetLogo replications in Tables 8 and 10 are the same, as they derive from the same source of values. Any perceptual

	Size	100	200	400	800	1600
C	T_{cen1}	0.38	1.57	6.25	25.17	118.71
	T_{cen2}	1.34	5.48	22.21	90.94	401.88
	$\Delta T(\%)$	252.14	249.93	255.22	261.36	238.53
NetLogo	T_{cen1}	10.30	42.70	313.50	1279.60	5230.30
	T_{cen2}	11.60	54.00	373.10	1547.50	6126.50
	$\Delta T(\%)$	12.62	26.46	19.01	20.94	17.13
OpenCL	T_{cen1}	1.67	1.70	1.83	2.24	3.47
	T_{cen2}	1.74	1.88	2.14	2.84	5.90
	$\Delta T(\%)$	4.42	10.55	17.46	27.02	70.23

Table 7: Average execution times of scenarios 1 and 2, percentage increase due to an increase in the number of agents in every implementation.

difference is only due to changes in the axis scale introduced by the red dots of the replications of C and OpenCL. The same happens in Tables 9 and 11 in the case of scenario 2.

We also establish the value of 0.05 as the significance level for all all p-values.

4.3.2 Alignment of C implementation

The results obtained in scenario 1 (Tabela 8) and in scenario 2 (Tabela 9) show the similarity between the implementations in C and in NetLogo.

By analysing 8 we can observe that, for size 800×800, the parametric test (t-test) and the non-parametric Mann-Whitney U test, both on the first PC, show p-values bellow the level of significance of 0.05. It may be observed that the score plot reflects the p-values, as it shows a slight horizontal deviation at the overlapping of the colour spots, i.e. a deviation at the axis of the first PC.

A failure of the t-test in the size 800 model does not imply the existence of any kind of alignment problem with the implementation in C because: a) in every case of both scenarios, the p-values of MANOVA are above 0.05; b) there is no observable systematic pattern in every size of any of the scenarios; e, c) there is always the possibility of a failure in a statistic test, especially when several tests are applied simultaneously.

On the other hand, the p-values of MANOVA, t-test and U-test in scenario 2 are (with the exception of U-test for the dimension 200×200), above 0.05, as may be seen in Table 9.

It is possible that the combination of the number of agents with the sequence of generated random numbers in the size 800 model (scenario 1), has the unfortunate result of the failure of both tests, t-test (p-value $= 0.005$) and U-test (p-value $= 0.006$).

Upon analysing the results, it is reasonable to conclude that the implementation in C is aligned with the implementation in NetLogo.

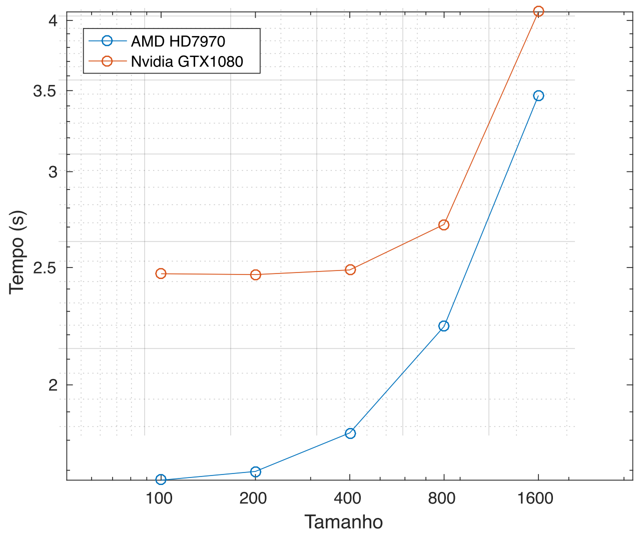
4.3.3 Alignment of OpenCL implementation

In Table 10, scenario 1, we can observe that sizes 100, 200 and 400 show MANOVA, t-test and U-test p-values above the level of significance of 0.05, apparently indicating an aligned model. However, these sizes correspond to the lowest number of agents present, respectively 100, 400 and 1600 agents, and we can see that for sizes 800 and 1600, MANOVA, t-tests and U-tests show p-values clearly bellow the significance level of 0.05.

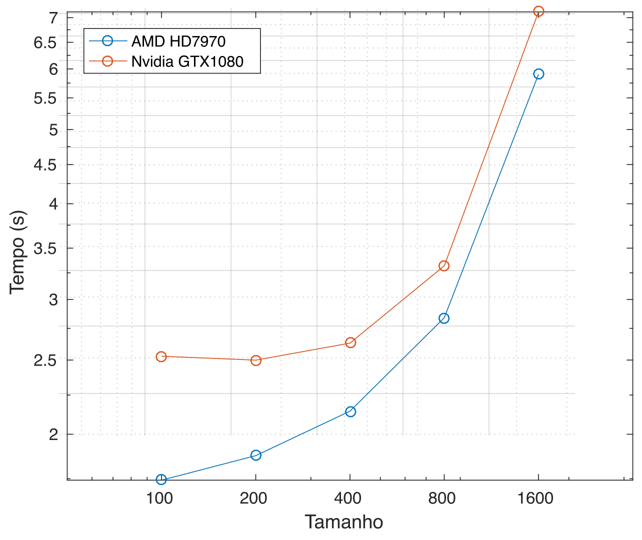
We can also see in the score plots of sizes 800 and 1600 in Table 10, which show the clear horizontal deviation between both colour spots, evidenced by the fact that t-test of the first PC shows values bellow 0.05.

The MANOVA p-values for all sizes of scenario 2 (Table 11) show values bellow 0.05. Furthermore, with the exception of sizes 100 and 200, the p-values of t-test are also bellow 0.05. The results of scenario 2 shown in Table 2 11 reflect a pattern that started in Table 10 and that confirm the non-alignment of the implementation in OpenCL.

The analysis of the results of Tables 10 and 11



(a) Scenario 1.



(b) Scenario 2.

Figure 6: Average execution times of OpenCL in the GPUs AMD Radeon HD7970 and Nvidia GeForce GTX 1080.

point towards a non-alignment of the implementation in OpenCL. This becomes even more clear when we compare them to the results of the implementation in C (Tables 8 and 9), particularly in the unique case of size 800 in scenario 1.

We can observe that the p-values of t-test, U-test mainly MANOVA, are consistently lower than the level of significance of 0.05, therefore, it is reasonable to conclude that the model implemented in OpenCL is not aligned with NetLogo.

4.3.4 Conclusions about the alignment of the models

The presented results point towards the alignment of the implementation in C with the implementation in NetLogo. A single p-value bellow 0.05 does not constitute an observable pattern that contradicts the remaining statistical results for that implementation (Section 4.3.2). On the other hand, we conclude that the model implemented in OpenCL is not aligned with NetLogo, as we could observe through a statistical analysis in Section 4.3.3.

The following question arises: how much is the implementation in OpenCL not aligned with the implementation in NetLogo and what is the dynamic of the implementation in OpenCL under these circunstances?

Two non-aligned models necessarily have different dynamics. By analysing the dynamics of OpenCL and NetLogo in scenario 2 and size 1600 (Image 7a), we can observe that NetLogo's curve (in green) totally surpasses OpenCL's curve (in red), at least at the scale considered for the image. The amplification of region 1 (Image 7b) enables to see that NetLogo's curve passes above OpenCL's curve very closely with occasional switches, as it can be observed in Image 7c.

In general, two implementations of the same model do not show dynamics with curves overlapping point-by-point. However, despite the great proximity of both dynamics (Image 7a), the implementation OpenCL is not aligned with NetLogo. We can therefore conclude that the alignment cannot be inferred through a simple visual inspection of the models dynamic.

The causes for the non-alignment in OpenCL may vary, including three that we point out:

1. The non-alignment is an intrinsic result of the model when it is parallelized.
2. The pseudo-random number generator is too partitioned regarding its maximum period.
3. Each pseudo-random number generator shows

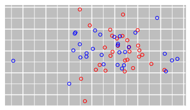
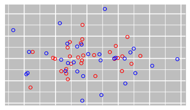
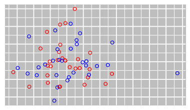
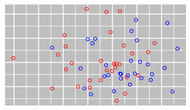
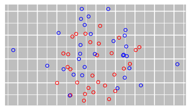
Size	#CPs (90% var.)	Tests	Output Results	PCs score plot
100×100	22	MNV (90% var.) t-test (PC1) MW-U test (PC1)	0.762 0.249 0.150	
200×200	26	MNV (90% var.) t-test (PC1) MW-U test (PC1)	0.185 0.792 0.832	
400×400	22	MNV (90% var.) t-test (PC1) MW-U test (PC1)	0.239 0.502 0.741	
800×800	26	MNV (90% var.) t-test (PC1) MW-U test (PC1)	0.473 $\underline{\underline{0.005}}$ $\underline{\underline{0.006}}$	
1600×1600	24	MNV (90% var.) t-test (PC1) MW-U test (PC1)	0.624 0.726 0.730	

Table 8: Results of the tests to the replications in scenario 1 in C and NetLogo. Size represents the dimension of the environment. #CPs are the number of principal components explaining at least 90% of the variance, obtained through the application of the method to 30 replications of $n = 3000$ interactions of each model. MNV(90% var.) is the MANOVA applied to the PCs. t-test (PC1) is the t-test applied to the first PC, while the MW-U test (PC1) is the Mann-Whitney U test applied to the first PC. ï£¡PCs score plotï£¡ is the score plot of PCs 1 and 2. Values bellow 0.05 are underlined, while values bellow 0.01 have a double underlining.

evenly distributed values, yet the distribution is not even over the multiple parallel generators.

Case 1

Model parallelization produces situations that are naturally different from those found when the model is executed in series. Taking the agents movement as an example; in series, each agent is executed in turn, and the decisions about their movement are taken based on a static state of the system, as seed by the agent at the instant of time it is in execution. This decision is considered final after the end of its time slot. This is how NetLogo and C work with a single thread.

In parallel, the agents move according to each other and their movement decisions are taken based on an "instant" of a permanently mutating system. Between the decision of moving and the execution of that decision, the system changes and what was previously true no longer is. In other words, at a certain instant, a certain agent discovers a free cell that matches its movement criteria and, at the moment of executing that movement, the intended cell becomes occupied, which forces the agent to restart the entire process of search and move, which may result in multiple failures.

In parallel, each agent has an elastic time slot and its decisions during the execution can never be considered final until the movement decision is actually executed. Thus, it is necessary to implement a stop mechanism in which the agent, after multiple unsuccessful attempts, abstains from trying to move and remains in the same cell. This result is not the same as the one observed in the case of series, therefore the series dynamics may be are intrinsically different *versus* parallel.

Case 2

The implementation in OpenCL uses a PRNG for each agent, as mentioned in Section **??**. For this purpose, a 32 bit XorShift PRNG with a period of 2^{32} was chosen.

Consider that in each i interaction, an agent a_1 receives a new random number from the generator previ-

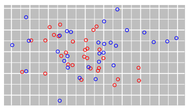
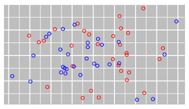
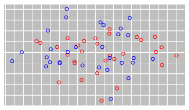
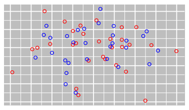
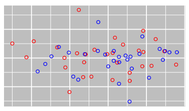
Size	#CPs (90% var.)	Tests	Output Results	PCs score plot
100×100	29	MNV (90% var.)	0.271	
		t-test (PC1)	0.278	
		MW U-test (PC1)	0.248	
200×200	32	MNV (90% var.)	0.563	
		t-test (PC1)	0.082	
		MW U-test (PC1)	$\underline{0.045}$	
400×400	34	MNV (90% var.)	0.954	
		t-test (PC1)	0.150	
		MW U-test (PC1)	0.273	
800×800	39	MNV (90% var.)	0.540	
		t-test (PC1)	0.788	
		MW U-test (PC1)	0.708	
1600×1600	34	MNV (90% var.)	0.854	
		t-test (PC1)	0.310	
		MW U-test (PC1)	0.350	

Table 9: Results of scenario 2 in C and NetLogo. Size represents the dimension of the environment. #CPs are the number of principal components explaining at least 90% of the variance, obtained through the application of the method to 30 replications of $n = 3000$ interactions of each model. MNV(90% var.) is the MANOVA applied to the PCs. t-test (PC1) is the t-test applied to the first PC, while the MW-U test (PC1) is the Mann-Whitney U test applied to the first PC. "PCs score plot" is the score plot of PCs 1 and 2. Values bellow 0.05 are underlined, while values bellow 0.01 have a double underlining.

ous state. Then, there is a possibility that other agents start to generate a sequence that was already allocated to agent a_1. For this to happen, is is only necessary that the sate of its PRNG belongs to the set of states generated by agent a_1.

The consequence of the parallelization of a PRNG, even if instanced with different seeds, is that all the agents have to share the maximum period sequence, in this case $2^{32} - 1$. At best, due to the existence of a agents, each of them will have access to a unique $(2^{32} - 1)/a$ period sub-sequence before generating values belonging to the sequence of another agent. The severity of the case increases with the number of agents, the number of the simulation interations and the amount of random values required in each interaction, as the probability is also greater for any agent to have its PRNG generating the value from which the obtained sequence was already generated or will be generated by another agent.

The division of the maximum sequence among the several agents, defined as generator partitioning and overlapping, even if partial, of the generated random value sequences, has the consequence of introducing unintended patterns in the agent's movement and in the results of the simulation [17], [18], [27].

Case 3

Each agent receives from its PRNG an even sequence of random values, which ensures that the actions executed by the agent do not display any patterns over the entire simulation. Even if all the PRNGs generate even sequences, the value sequences taken from multiple generators are not necessarily even. For a better understanding, let us imagine that each line of a matrix is an even sequence of random values. The sequences taken from that array in any other direction, vertical or oblique, cannot represent even sequences of random values.

The consequence is also the introduction of oblique patterns in the actions of multiple agents, and there

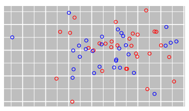
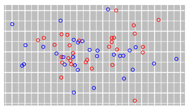
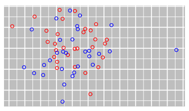
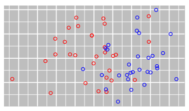
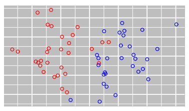
Size	#CPs (90% var.)	Tests	Output Results	PCs score plot
100×100	18	MNV (90% var.)	0.522	
		t-test (PC1)	0.256	
		MW U-test (PC1)	0.230	
200×200	23	MNV (90% var.)	0.738	
		t-test (PC1)	0.520	
		MW U-test (PC1)	0.708	
400×400	21	MNV (90% var.)	0.073	
		t-test (PC1)	0.969	
		MW U-test (PC1)	0.741	
800×800	18	MNV (90% var.)	$\underline{\underline{\text{4e-08}}}$	
		t-test (PC1)	$\underline{\underline{\text{2e-08}}}$	
		MW U-test (PC1)	$\underline{\underline{\text{3e-08}}}$	
1600×1600	7	MNV (90% var.)	$\underline{\underline{\text{1e-15}}}$	
		t-test (PC1)	$\underline{\underline{\text{9e-16}}}$	
		MW U-test (PC1)	$\underline{\underline{\text{1e-13}}}$	

Table 10: Results of the replications of scenario 1 in OpenCL and NetLogo. Size represents the dimension of the environment. #CPs are the number of principal components explaining at least 90% of the variance, obtained through the application of the method to 30 replications of $n = 3000$ interactions of each model. MNV(90% var.) is the MANOVA applied to the PCs. t-test (PC1) is the t-test applied to the first PC, while the MW-U test (PC1) is the Mann-Whitney U test applied to the first PC. "PCs score plot" is the score plot of PCs 1 and 2. Values bellow 0.05 are underlined, while values bellow 0.01 have a double underlining.

may be cases where the actions of some agents mirror the actions of other agents at any point of the simulation. Just as in case 2, the introduction of unintended patterns has negative consequences on the simulation's results.

5 Conclusions and Future Work

We expect this work may somehow contribute for the continuation of the study of agent-based models in general, and particularly for the study of biological models such as Heatbugs or other of its extended versions. This work has presented a set of challenges to which the solutions may be exported to other models and implementations. The model's formal and systemic description presented by this work was obtained through a process similar to the reverse engineering of the source code of NetLogo's implementation, which is the most known and used. It would certainly be more interesting to perform the inverse process, where the model could be reimplemented and even extended having its formal description as basis.

The reimplementation of source code models into new code, i.e. Starting from a source to be reimplemented in another platform is a fallible process, as little details may be missed unintentionally, thus producing different models. The implementation in C served as a middle point in order to gain familiarity with the source code in NetLogo, acting as a bridge for a description of the model using the ODD protocol and its implementation in the target architecture, namely the GPU. Nevertheless, the implementation in C stands for itself, as it is a new implementation of the Heatbugs model and it contains essential information: a) it externalizes opaque functionalities typical of the agent-based simulations such as NetLogo, and it "clearly" displays them while implemented; and, b) it shows results that may be analysed and compared, thus adding to the model's knowledge base.

In general, simulations are performed to predict the

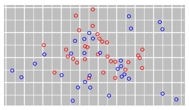
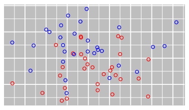
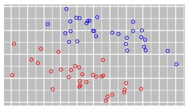
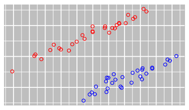
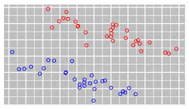
Size	#CPs (90% var.)	Tests	Output Results	PCs score plot
100×100	30	MNV (90% var.)	0.009	
		t-test (PC1)	0.967	
		MW U-test (PC1)	0.947	
200×200	34	MNV (90% var.)	0.001	
		t-test (PC1)	0.439	
		MW U-test (PC1)	0.286	
400×400	33	MNV (90% var.)	$\leq$1e-08	
		t-test (PC1)	0.003	
		MW U-test (PC1)	0.008	
800×800	31	MNV (90% var.)	$\leq$1e-08	
		t-test (PC1)	4e-05	
		MW U-test (PC1)	2e-04	
1600×1600	32	MNV (90% var.)	$\leq$1e-08	
		t-test (PC1)	0.002	
		MW U-test (PC1)	0.004	

Table 11: Results of the replications of scenario 2 in OpenCL and NetLogo. Size represents the dimension of the environment. #CPs are the number of principal components explaining at least 90% of the variance, obtained through the application of the method to 30 replications of $n = 3000$ interactions of each model. MNV(90% var.) is the MANOVA applied to the PCs. t-test (PC1) is the t-test applied to the first PC, while the MW-U test (PC1) is the Mann-Whitney U test applied to the first PC. "PCs score plot" is the score plot of PCs 1 and 2. Values bellow 0.05 are underlined, while values bellow 0.01 have a double underlining.

evolution of a certain system or to find solutions for several problems. In the cases where a system can be described by a set of simple elements with certain internal characteristics and following a set of rules, the agent-based models constitute a practical mean to perform the simulation of such systems because the results emerge from the agent-agent and agent-environment interactions. However, achieving reliable results implies: a) the use of a potentially high number of agents; b) potentially vast environments; and, c) execution of countless interactions. It is therefore a subject of greater interest to find means to accelerate the execution of these simulations, by maximizing the use of available processing resources and by taking advantage of the parallelism provided by the most recent processors.

GPUs are processing systems with architectural characteristics (the fact that they have hundreds and even thousands of SIMD execution units) that make them potentially suitable for the simulation of ABMs during which multiple elements of the model are executed in parallel by the execution units. In this work, we tested the viability of implementing ABMs in GPUs with the purpose of using their massive parallelism to achieve a substantial reduction of the simulation time.

We observed, in several performed tests in two different scenarios, the great acceleration potential of the Heatbugs model in the GPUs when compared to two implementations of the same model, in C and in NetLogo, running in a single thread in CPU. OpenCL is around 1000 faster than NetLogo when running in a Java Virtual Machine (JVM), and approximately 50 faster than the implementation in C when compiled to run directly in CPU. However, the performance analysis of the implementations in NetLogo and in C reveals some information about the advantages and disadvantages of these two platforms.

The complexity of NetLogo's platform adds processing weight with a negative impact on performance. However, this complexity has the advantage of providing NetLogo with several characteristics: a) automatic resource management and optimizations provided by the JVM; b) a vast set of functions common to any agent-based model; c) focusing on the model's descriptive development, hiding complexities inherent to the declaration and manipulation of data structures, interactions, enumerations, and others; and d) it shows a simple and high-level imperative language to describe the agent's actions and the environment's characteristics.

On the other hand, C and, by extension, OpenCL are platforms with a close relationship with the hardware, extremely fast in execution, but lacking high-level functionalities suitable for the implementation of agent-based models. In this situation, every detail and functionality of the model has to be implemented by the user, who also has to deal with several aspects not related with the model, such as memory management, data structures declaration and manipulation, agent sequencing, among other control mechanisms. In OpenCL, the user also has to make sure that the implementation is suitable for the platform, i.e. limitations such as remaining memory available in the device, absence of memory management mechanisms, as well as the programming language's characteristics as a sub-set of C, conditis the design of the GPU implementation.

We also approached the importance of model alignment. Two aligned models have statistically indistinguishable results, therefore their processes and sequence or scheduling are similar in both models. A suitable statistical method was used for ABMs comparison in order to infer on the similarity of C and OpenCL comparing to NetLogo. We observed that the implementation in C, with the exception of a result that does not constitute an observable pattern, is aligned with NetLogo. On the other hand, we also observed that OpenCL is not aligned with NetLogo in spite of the care in the implementation and observation of every process. Three possible causes were presented for the failure in the OpenCL's alignment, which may have to do with the pseudo-random number generator or with consequences intrinsic to the model's parallelization.

In conclusion, there is great potential in using GPUs to simulate ABMs in general, and particularly the Heatbugs model. The performance is promising, but there are problems that need to be worked on for a total viability.

5.1 Future Work

In order to achieve the viability of Heatbugs model's implementation in GPU, further work is required, specifically in identifying exactly the causes for the model's non-alignment. As these causes need to be determined, future works on this model in GPU should be performed in order to determine: a) the influence of parallelization on the results; and, b) how the pseudo-random number generators influence results in parallel systems. Such works may also benefit other models to be implemented in this platform.

ABMs may be used in some urgent situations, e.g. to determine potential risk areas through the dissemination of disease transmitting agents. In these circumstances where saving lives depends on the swiftness of actions by the sanitary authorities and on the previous knowledge on the most suitable way of mobilizing resources, the time factor associated to the development of models in a new platform is surely an unacceptable risk, as it requires: a) familiarity with architecture; and b) validating implementations in that platform. Another work we may consider consists in developing design patterns for GPU that may be common to several agent-based models, with the goal of expediting the development of new models. Finally, it is also possible to develop work to allow the existing platforms, e.g. NetLogo, to access the processing power of the GPUs by creating libraries that would bridge both. In the latter case, there are already some steps being taken [29].

References

[1] Arifin, S. N., Madey, G. R., and Collins, F. H. *Spatial agent-based simulation modeling in public health: design, implementation, and applications for malaria epidemiology.* John Wiley & Sons, 2016.

[2] Boneau, C. A. The effects of violations of assumptions underlying the t test. *Psychological bulletin 57*, 1 (1960), 49.

[3] Chiacchio, F., Pennisi, M., Russo, G., Motta, S., and Pappalardo, F. Agent-based modeling of the immune system: Netlogo, a promising framework. *BioMed research international 2014* (2014).

[4] Dodge, Y. *The concise encyclopedia of statistics.* Springer Science & Business Media, 2008.

[5] D'Souza, R. M., Lysenko, M., Marino, S., and Kirschner, D. Data-parallel algorithms for agent-

based model simulation of tuberculosis on graphics processing units. In *Proceedings of the 2009 spring simulation multiconference* (2009), Society for Computer Simulation International, p. 21.

[6] DâĂŹSouza, R. M., Lysenko, M., and Rahmani, K. Sugarscape on steroids: simulating over a million agents at interactive rates. In *Proceedings of Agent2007 conference. Chicago, IL* (2007).

[7] Edmonds, B., and Hales, D. Replication, replication and replication: Some hard lessons from model alignment. *Journal of Artificial Societies and Social Simulation 6*, 4 (2003).

[8] Fachada, N. Agent-based simulation of the immune system. *Master's thesis, Instituto Superior Técnico, Lisboa (July 2008)* (2008).

[9] Fachada, N., Figueiredo, M. A., Lopes, V. V., Martins, R. C., and Rosa, A. C. Spectrometric differentiation of yeast strains using minimum volume increase and minimum direction change clustering criteria. *Pattern Recognition Letters 45* (2014), 55–61.

[10] Fachada, N., Lopes, V., Martins, R., and Rosa, A. Perfandpubtools–tools for software performance analysis and publishing of results. *Journal of Open Research Software 4*, 1 (2016). DOI:`http://doi.org/10.5334/jors.115`.

[11] Fachada, N., Lopes, V. V., Martins, R. C., and Rosa, A. C. Model-independent comparison of simulation output. *Simulation Modelling Practice and Theory 72* (2017), 131–149.

[12] Fachada, N., Rodrigues, J., Lopes, V. V., Martins, R. C., and Rosa, A. C. micompr: An R Package for Multivariate Independent Comparison of Observations. *The R Journal 8*, 2 (2016), 405–420.

[13] Fulcher, J., and Jain, L. C. *Computational intelligence: a compendium*, vol. 21. Springer, 2008.

[14] Grimm, V., Berger, U., DeAngelis, D. L., Polhill, J. G., Giske, J., and Railsback, S. F. The odd protocol: a review and first update. *Ecological modelling 221*, 23 (2010), 2760–2768.

[15] Gulyás12, L., Szabó, A., Legéndi, R., Máhr, T., Bocsi, R., and Kampis, G. Tools for large scale (distributed) agent-based computational experiments.

[16] Hamann, H., and Wörn, H. An analytical and spatial model of foraging in a swarm of robots. In *International Workshop on Swarm Robotics* (2006), Springer, pp. 43–55.

[17] Hellekalek, P. Don't trust parallel monte carlo! In *ACM SIGSIM Simulation Digest* (1998), vol. 28, IEEE Computer Society, pp. 82–89.

[18] Hill, D. R. Practical distribution of random streams for stochastic high performance computing. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on* (2010), IEEE, pp. 1–8.

[19] Hunt, C. A., Kennedy, R. C., Kim, S. H., and Ropella, G. E. Agent-based modeling: a systematic assessment of use cases and requirements for enhancing pharmaceutical research and development productivity. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine 5*, 4 (2013), 461–480.

[20] Jolliffe, I. Principal component analysis. In *International encyclopedia of statistical science*. Springer, 2011, pp. 1094–1096.

[21] Krzanowski, W. *Principles of Multivariate Analysis: A User's Perspective*. Oxford science publications. Clarendon Press, 1990.

[22] Laskowski, M., Demianyk, B. C., Witt, J., Mukhi, S. N., Friesen, M. R., and McLeod, R. D. Agent-based modeling of the spread of influenza-like illness in an emergency department: a simulation study. *IEEE Transactions on Information Technology in Biomedicine 15*, 6 (2011), 877–889.

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

[24] Merlone, U., Sonnessa, M., and Terna, P. Horizontal and vertical multiple implementations in a model of industrial districts. *Journal of Artificial Societies and Social Simulation 11*, 2 (2008), 5.

[25] Minar, N., Burkhart, R., Langton, C., Askenazi, M., et al. The swarm simulation system: A toolkit for building multi-agent simulations.

[26] North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. Complex adaptive systems modeling with repast simphony. *Complex adaptive systems modeling 1*, 1 (2013), 3.

[27] Passerat-Palmbach, J., Mazel, C., and Hill, D. R. Threadlocalmrg32k3a: A statistically sound substitute to pseudorandom number generation in parallel java applications. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on* (2012), IEEE, pp. 543–550.

[28] Price, J., and McIntosh-Smith, S. Analyzing and improving performance portability of opencl applications via auto-tuning. In *Proceedings of the 5th International Workshop on OpenCL* (2017), ACM, p. 14.

[29] Procházka, J., and Štekerová, K. Opencl for large-scale agent-based simulations. In *Computational Collective Intelligence* (Cham, 2017), N. T. Nguyen, G. A. Papadopoulos, P. Jędrzejowicz, B. Trawiński, and G. Vossen, Eds., Springer International Publishing, pp. 351–360.

[30] Quinn, M. J., Metoyer, R. A., and Hunter-Zaworski, K. Parallel implementation of the social forces model. In *Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics* (2003), pp. 63–74.

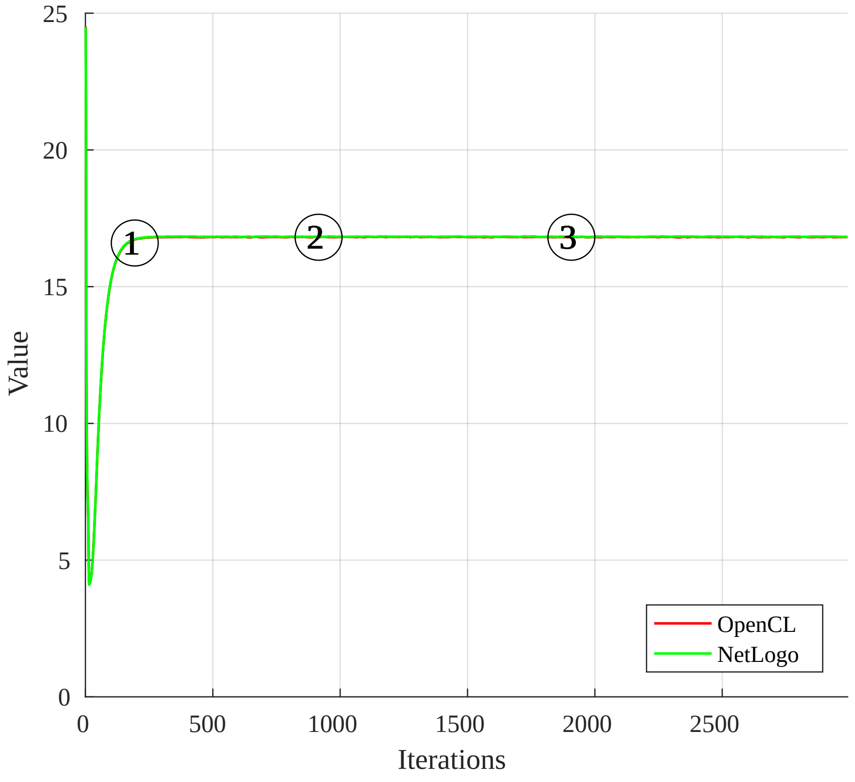
[31] Rivest, R. The md5 message-digest algorithm. Tech. rep., 1992.

[32] Tabachnick, B., and Fidell, L. *Using Multivariate Statistics: Pearson New International Edition*. Pearson Education Limited, 2013.

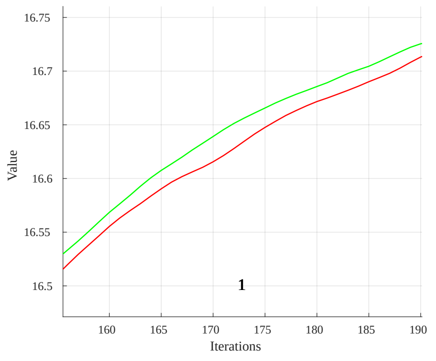
[33] Wilensky, U. Netlogo. `http://ccl.northwestern.edu/netlogo/`.

[34] Wilensky, U. Netlogo heatbugs model. *Web page http://ccl. northwestern. edu/netlogo/models/Heatbugs. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL* (2004).

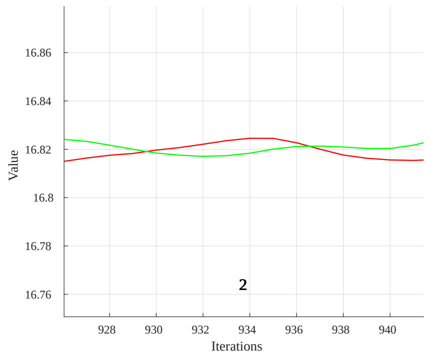
[35] Wilensky, U., and Rand, W. Making models match: Replicating an agent-based model. *Journal of Artificial Societies and Social Simulation 10*, 4 (2007), 2. `http://jasss.soc.surrey.ac.uk/10/4/2.html`.



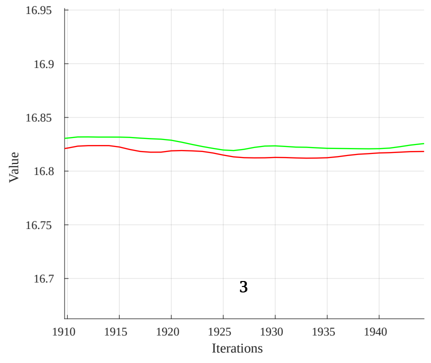
(a) Average of 30 replications of the output of the implementations in NetLogo and OpenCL, with three randomly chosen points.



(b) Amplification of point 1.



(c) Amplification of point 2.



(d) Amplification of point 3.

Figure 7: Dynamic of OpenCL vs. NetLogo. Each graphic corresponds to the average of 30 replications in scenario 2 size 1600.