Final Report
Agent-based application development and performance analysis using EcoLab

Daniel Beltrán Mora

Abstract

High Perfomance Computing(HPC) is a distributed computational technique that provides a solution to computational problems that are too complex or large enough for desktop computers. Although there are so many computational methods out there for develop applications within a HPC approach, in this paper, only agent-based modeling(ABM) is treated. ABM is a computational method used to create and experiment with models composed by a set of entities called agents which make decisions autonomously. This paper is mainly about the development of a benchmark application for the EcoLab Framework which is a agent-based modeling and simulation(ABMS) platform for HPC system . In where the simulation part, consists in the experimentation with a computational type model that simulates the actions and interactions of autonomous agents with the goal of assessing their effects on a system as a whole. The resulting application is used for perform a performance analysis within the Ecolab framework and also provides a way to validate it and the agent-based model implementation.

Keywords– EcoLab, Agent-based modeling, High Performance Computing , Distributed, Application Performance

1 INTRODUCTION

HIGH PERFOMANCE COMPUTING(HPC)[8] is a computational technique, used when the performance matters , that provides a solution to computational problems that are too complex or large enough for desktop computers. A HPC environment is essentially a fast local area network of computational nodes where the nodes are composed by one or more processing chips as well as its own memory. It is for this reason that the computational problems programmed for a HPC must be split into many sub-problems so every node has a task to realize. This form of programming is known as parallel computing in which a sequential application is divided into many smaller applications called threads that work simultaneously. Developer, on the other hand, will have to develop a way for communicate , synchronize and organize these threads to piece together the application in order to have the same result than in the sequential application. There are many ways to develop a parallel application and in this paper it is proposed to use an Agent-based modeling(ABM)[3] framework.

Agent-based modeling(ABM) is a computational method used to create and experiment with models composed by a set of entities called agents which make decisions autonomously. The behaviour of these agents are determined by a set of a individual agent preferences and makes decisions based in these preferences and the iterations between agents which can influence in these decisions. There are a lots of frameworks that implement this computational method with a HPC approach like RepastHPC[11] and flame[5] but this paper is about the development of a particular benchmark for agent-based models in the EcoLab framework [19] which is a agent-based modeling and simulation platform for high performance computing where the simulation part consist in the experimentation with a computational type model that simulates the actions and interactions of autonomous agents with the goal of assessing their effects on a system as a whole. EcoLab needs an agent-based model to function in a HPC environment. In order to define an agent-based model, EcoLab , comes with Classdesc[12] and Graphcode[20] libraries that together provides a toolkit which greatly reduces the complexity of the communications and interactions methods.

As a Proof of concept(PoC), I implemented in EcoLab framework a benchmark application, which is a test used to measure the relative performance of a system, specifically designed for an ABM. This benchmark name is prisoner's dilemma model and it has been implemented in two others similar frameworks called Repast HPC and Flame. The grace of this benchmark is that it takes into consideration most of common characteristics of these ABM applications and includes parameters for influencing their relevant performance aspects. The goals of implement this benchmark are to have a relevant way to validate the framework and the model implementation and also to collect data for the prisoner's dilemma model.

This paper is divided in the sections: EcoLab Descrip-

• Email: daniel.beltranm@e-campus.uab.cat
• Specialization in Computer Engineering
• Work tutored by : Anna Sikora
• Course 2017/18

tion which introduces the EcoLab framework ,Methodology which explains the steps in order to perform the PoC, Background which contains concepts related to EcoLab framework, Benchmark which explains the PoC , Results which explains the PoC performance analysis , Conclusions which explains the personal ideas and thoughts about the PoC and EcoLab framework and lastly the ACKNOWLEDGEMENTS.

2 ECOLAB DESCRIPTION

EcoLab is an agent-based modeling and simulator framework designed by Russell K. Standis designed for C++ programmers that allows to the user to implement his model in C++ and simulate them in a script language called TCL[**tcl**] which is capable of access to the methods and variables of the model allowing to perform experiments dynamically. TCL also provides a series of instruments that can be coupled together with an agent-based model at runtime. EcoLab has support for HPC environment in which the agents are distributed over an arbitrary topology graph.

In order to serialize the agents of the model it makes use of an included library called Classdesc which is done by the same author. It extends the functionality of C++ providing a reflection system[13] which allows to serialize in a binary way every agent of the models. This mechanism is useful for the communication between the agents of the model. The implementation of the Message Passing between different computers is done by another library called Graphcode, included in the framework, which is based on the MPI[9] protocol. The most interesting characteristics are as follows:

- The model is implemented as a C++ object and so is capable of use all of the C++ standard library without limitation.
- It uses a scripting language TCL to access to the model's methods and instance variables allowing to perform experiments re-coding for it only the TCL scripts without recompile the model.
- EcoLab models can use Graphcode and Classdesc libraries to implement a distributed network of agents over an MPI-based cluster[6] computer.

2.1 Classdesc

Object reflections is a mechanism for facilitating the implementation of serialization[14], which is the process of create binary data representing a object and in order to works it requires a knowledge of the structure of the object.

Classdesc is a program that has been included in the EcoLab platform and describes a set of functions that implements an object reflection class for C++. Classdesc parses an input program and emits function declarations that know about the structure of the objects. These functions only needs to handle class, struct and union definitions and what is emitted in the object descriptor, is a sequence of functions calls for each base class and member. It also implements a TCL_obj class descriptor that creates a set of functions for the TCL interpreter that allows to query or set C++ object's members.

Once a class definition has been parsed by Classdesc and the class descriptors are embedded into the original program. An object of that class can be serialized into a buffer object. Once the buffer is packed it can be transferred within the MPI-based cluster machines.

Classdesc is mainly utilized in Graphcode in order to provide a message passing and serialization mechanisms for complex objects necessary for the the agents allocated in the distribute graph of Graphcode.

2.2 Graphcode

Graphcode abstracts the message passing codes that provides Classdesc in order to provide a richer framework that is easier to use.

It is a graph that contains a set of distributed objects where the computation takes place within the vertex of the graph and the communication takes place along the edges of the graph. The objects inside a graph are interconnected via a neighbourdhood.

Graphcode objects may be located on any processors and are capable of be migrated dynamically to any other. These can be acceded through variables of type Classdesc::objref that contains the object ID, it's location (proccesor rank) and can be dereferenced in order to obtain access to the object variables. Since the objects and be anywhere each process maintains a map in order to keep track of all the objects in the graph which is necessary for regenerate the neighbourhood linklist after a migration.

To create a graph, it is necessary to call to method Graph::AddObject and adding the links to each object to form the graph.

In order to migrate objects between processors it is only necessary to change the object location and Graphcode will arrange automatically a set of MPI Classdesc functions to achieve the change and so the developer only needs to call to synchronize methods. Additionally it can call to PARMETIS[10] parallel graph partitioner to partition the graph across the available processors in an efficient way.

2.3 Communication

While working under Graphcode the communication pattern is defined by the graph links. If an object computation involves the neighbouring objects, it is necessary to perform an update into this neighbourhood. In an updating to a neighbourdhood, Graphcode stores the results into a backing buffer graph and then it swaps the backing buffer with the original graph. The only communication required is to ensure that a copy of all neighbours that resides on remote process are being transferred to the process in which the object resides. It can be done calling to Prepare_Neighbours() before starting the computation.

There are another types of distributed call such as gather(), partition_objects() , distribute_objects().

2.4 Structure

EcoLab was mainly designed for a specify model called EcoLab model and in the recent years, according to the author, it has been the basis of a program called Minsky, which supports dynamic systems approaches to economics that don't need HPC.

However the author affirms that it can be adapted to any other agent-based model and is willing to test it performance in a HPC system.

This is possible because it was designed in a generic way and a few already models, like jellyfish, demonstrates this statement.

Figure 1 presents a file structure to design an HPC agent-based model.

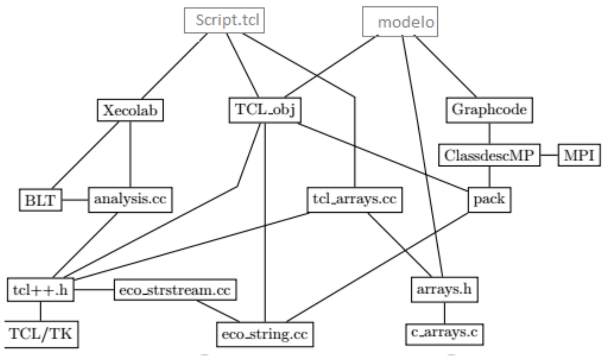

Fig. 1: File Structure

From figure 1 the following include files was utilized:

- **Tcl++**: Serve as a binder of TCL.
- **TCL_obj**: Allows to tcl the usage of objects created in C++.
- **Pack**: Performs the object serialization and allows to perform checkpoints.
- **ClassdescMp**: Extends Pack functionality providing functions to serialize the objects within a distributed platform.
- **Graphcode**: Allows the abstraction of the objects that forms part of a distributed Graph.

In order to perform some execution. It is mandatory build an experiment in a TCL script that acts as the main code of the application. This TCL binds the C++ model and his agents, the parameters for initialize the model and a toolkit of instrumentation.

3 Background

- **RepastHPC:** Is a free open source toolkit designed for High Perfomance Computing written in C++ that and makes use of Message Passing Interface (MPI) in order to handle the interactions and actions of the agent's within the system. It provides a support for the development of extremely flexible agent-base models and allows to dynamically access and modify the parameters of the model. Because repast Hpc is a tool for make agent-based models it provides a suit of features for evaluate, execute and creating these models with a certain degree of ease.
- **Flame:** is a template driven framework for agent-based modelling on parallel architectures.It provides a flexible agents modulation which allows to implement a diverse set of models in mostly of the systems.
- **Flame and RespastHPC evaluations:** Both frameworks have a similar evaluation explained in the paper Designing a Benchmark for Assessing the Performance of Parallel Agent-based Simulation Applications[1] from the authors Andreu Moreno, Anna Sikora and Eduardo César.
- **MPI:** is a message-passing standard used for the communication between machines within a cluster environment . All operations are expressed as functions, subroutine or methods linked to C++ and FORTRAN languages. There are several well-tested and efficient implementations of MPI. One of these is OpenMP which is used in this PoF.
- **Cluster:** Set of computers connected to each others through fast local area networks. These computers act like one large machine that internally plans the workload in a distributed way.
- **C++:** It is a multi-paradigm programming language that allows to use the pragmatics of structured,imperative, genetic and oriented to objects programming.
- **TCL:** it is an interpreter which contains embedded a command oriented toolkit language. In this Proof of concept TLC is used for call functions and variables of the C++ EcoLab model.
- **HPC:** Is a computational technique that , given a huge application, relies in parallel computation in order to resolve it in a efficient,reliable and quickly way.

4 Methodology

The main goal of this PoC is to develop the prisoner's dilemma model[1] in the EcoLab framework in order to perform a performance analysis in the resulting application.

This goal was reached following a set of tasks in sequential order as follows:

Firstly, is the research of documentation about EcoLab the reason behind this is because the EcoLab is a bit unknown so there are essential things to know like for what it is, how can it be installed in a centOS system without root privileges, what are the packet dependencies that it requires in order to work in a HPC, which libraries it uses, and how implement and simulate own user models. However there is a lack of documentation in reference of how to implement EcoLab within in centOS mpi-based cluster. That is caused by a set of diverse factors which involve that EcoLab is still in developing, there are very few users, so it isn't very well know, and the lasted years it hasn't been used in HPC environments.

Secondly, Is the Configuration and installation of EcoLab platform in the CentOS cluster provided by DASCO department.

Thirdly, Is the acquisition and development of the prisoner's dilemma model in EcoLab. The development consists in replicate, in the most accurate way, the same behaviour and interactions of the agents that are developed in prisoner's dilemma model of RepastHPC version but using for it the tools that EcoLab provides through Graphcode.

Fourthly, Is the validation test performed to ensure that the first approach is valid, which consist in the search and resolve of all possible bugs from the development of the previous task.

Fifthly , Is the performance test performed to ensure that there isn't anomalies in the results.

Sixty, Is the acquisition of results and conclusions.

Finally, is the writing of this paper which explain the work done and results obtained.

5 Benchmark

All Proof of concept is done in a centOS mpi-based cluster. This cluster has twelve nodes of two sockets in which there are six cores, provided by DASCO department. Also, it has a list of modules available which includes gcc/6.10 , openmpi/1.8.1 or/and 1.10.1 and papi/5.4.3. That are necessary for the running of EcoLab.

The version installed of EcoLab platform is 5.51 which contains version 3 of Graphcode and version 3.35 of Classdesc.

5.1 Installation

In order to install EcoLab, there are a set of mandatory requirements.

The installation is performed locally because it was been taken over in a environment without root privileges. This has resulted a set of problems which are:

1. Dependencies has to be searched and installed manually. These are installed in $home/makeInstall.
2. EcoLab isn't thought to be installed locally and it doesn't recognize the file structure of the DASCO cluster.
3. Because the EcoLab is in a long-term state developing it uses a old-fashioned makefile this means that mostly of the makefiles that EcoLab uses must be adapted to a local installation manually.
4. It is mandatory to load the modules of openmpi,papi and gcc before perform any installation.
5. Lastly the environment variables of the DASCO cluster must be configured so that all local packages can be recognized by the various makefiles of EcoLab and it dependence's.

The solution to these problems are mostly resolved using the listing 1 script at start of the session in which /home-/pfc/dbeltran must be equal at your $HOME path.

```bash
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
module unload openmpi/1.8.1
module unload gcc/7.2.0
module unload papi/5.4.3
module unload likwid/4.0.1
module load gcc/6.1.0
module load openmpi/1.8.1
module load papi/5.4.3
module load likwid/4.0.1
export PKG_CONFIG_PATH=$HOME/makeInstall/lib/
    pkgconfig:$PKG_CONFIG_PATH
export INCLUDE_PATH=/usr/include:$HOME/
    makeInstall/include:$INCLUDE_PATH
export CPATH=/usr/include:$HOME/makeInstall/
    include:$CPATH
export PATH=/usr/bin:/usr/sbin:$HOME/makeInstall/
    bin:$HOME/makeInstall/sbin:$PATH
export LD_LIBRARY_PATH=/usr/lib:/usr/lib64:$HOME/
    makeInstall/lib:$HOME/makeInstall/lib64:
   $LD_LIBRARY_PATH
export LIBRARY_PATH=/usr/lib:/usr/lib64:$HOME/
    makeInstall/lib:$HOME/makeInstall/lib64:
   $LIBRARY_PATH
export LD_RUN_PATH=$LD_LIBRARY_PATH:
export EcoLab_HOME=$HOME/makeInstall
export VPATH=$EcoLab_HOME/include
export Classdesc=$EcoLab_HOME/bin
GIT_DISCOVERY_ACROSS_FILESYSTEM=true
export GIT_DISCOVERY_ACROSS_FILESYSTEM
DIRS=$DIRS:$EcoLab_HOME:$HOME/usr:$EcoLab_HOME/
    usr:$EcoLab_HOME/X11R6:/usr/X11R6

```

Listing 1: confenv.sh

Once the environment is configured, the next step is locate the most important EcoLab packages dependencies which are:

- **GCC 4.5**[7]: Allows to compile the C++ code. 4.5 is the minim version, The installation was performed with 6.1.
- **TCL/TK 8.5 and BLT 2.4** [15][22]: TCL is already explained in section two meanwhile TK and BLT extends TCL functionally for graphical applications.
- **Cairo**[4]: A graphical library to perform png images.
- **Openmpi 1.8.1**: It is a free implementation of the message-passing standard(MPI). EcoLab only works in 1.X.X version.

However, it also requires **Zlib**[23],**Readline**[17],**UNURAN+PRNG**[16][21] and **Berkley DB**[2] in order to be successfully installed. Although, the only ones that are used in the developed model are the important ones. Note that there could be others dependencies for these packages which relies in the user OS.

Once located all packages. In order to install them, is necessary use the bash commands of make, make install and configure. Generally, the packages are installed by this set of instructions that download,uncompressed, choose the installation path and install them.

1. WGET direct_link https://packages.ubuntu.com/ got plenty of them
2. tar fx path of downloaded package
3. cd path of uncompress package
4. ./configure –prefix=$home/makeInstall
5. make -j && make install

So now, the EcoLab framework can be installed by the following set of instructions:

1. Download it from github repository along side with Classdesc and Graphcode
2. tar fx path and cd into.
3. mkdir Graphcode and decompress Graphcode there.
4. mkdir Classdesc and decompress Classdesc there.
5. cd into models folder and customize the makefile deleting the setting $EcoLab_HOME line
6. cd .. and compile EcoLab with make -j PREFIX=$HOME/makeInstall MPI=1 NOGUI= GCC= UNURAN=1 PRNG=1 AEGIS= && make install
7. In order to generate any example model you can access into pathEcoLabPackage/models and perform make -j MPI=1 NOGUI= GCC= UNURAN=1 PRNG=1 AEGIS= . This instruction creates a binary associate to one model.

5.2 Model development

Prisoner's dilemma model is a benchmark model for parallel Agent-based modeling and simulation(ABMS) applications which is used to measure a set of relevant factors that affects to the performance of ABMS applications. These factors are Communication volume, frequency and pattern , Amount of computation , Distribution and evolution of the workload and Size of the test.

5.2.1 Structure Description

In order to successful compile the model in the Ecolab framework is necessary to code the logic of the model and the behaviour of his agents in one file TCL, one file cc and one file h in which:

- file.tcl: Contains the main file of the application.
- file.cc: Contains the logic that defines the model and his agents.
- file.h: Contains the structure of the model and his agents.

Figure 2 presents a class diagram that can be looked for get an idea of the model structure.

5.2.2 Class EcoLabDemoAgent

The behaviour and data of the agents are defined in the class EcoLabDemoAgent. it contains a set of variables that are used for identify an agent in the model such as id and rank where rank determine the original creator of the agent and a set of variables to store results about the interactions between agents. This class also contains a array of N size in order to increment the cost of communication. The functions and methods that describe the behaviour of the agents are play, compute, cooperate and setm. in order to initialize the agents it is done with EcoLabDemoAgent newagent(Agent ID,Processor RANK,type) which creates a new agent newagent.setm(newm) which initialize the buffer array.

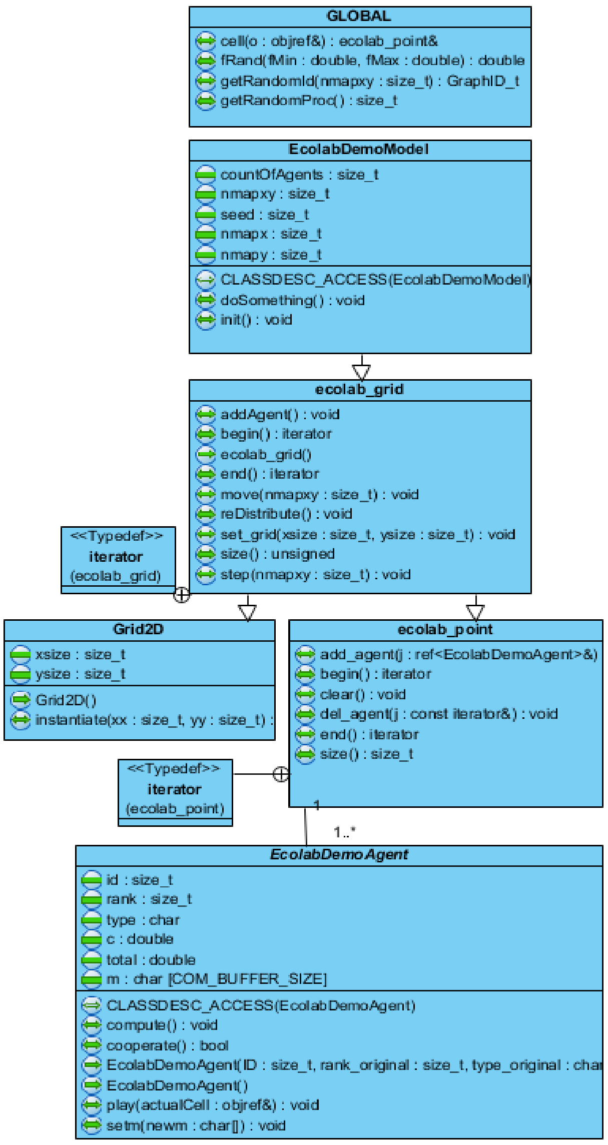

Fig. 2: Class Diagram

5.2.3 Class Ecolab_point

In order to add agents into EcoLab platform these must be attached to a virtual cell that form part of a distributed graph served by Graphcode_NS::graph. This virtual cell is called EcoLab_point which has access into a vector of agents and is converted into a GRAPHCODE_NS::Object type. Also, it haves a set of methods used for iterate through these agents and add/delete new ones.

5.2.4 Class Grid2D

The initialization of the grid is done with the function instantiate(size x,size y) from the class Grid2D which inherits GRAPHCODE_NS::graph. Every process that calls this function is gonna create a part of the grid and will create the neighbourhood of every own cell. The grid size is determined by the variables nmapx , nmapy and the amount of MPI processors. Finally, in order to be visible by all processors, it calls to this-¿rebuild_local_list(); and this-¿Partition_Objects()

5.2.5 Class Ecolab_Grid

This class inherits from Grid2D and EcoLab_point. The existence of this class is for have a mechanic that allows to the model to iterate through the Graph Grid and also through graph cell agents. The iteration through cells is done with a set of methods alike to the ones defined in EcoLab_point but this time are created for the Grid2D class. Because it also inherit from EcoLab_point, the iteration methods defined inside it, can be used by this class. Additionally, it has methods for redistribute the graph which is a method to distribute the agents along a grid space of determined processor and also it migrates cells between processors. Finally, in the method step(size_t nmapxy);, Is defined the schedule of one iteration that is composed by EcoLabDemoAgent::play, EcoLabDemoAgent::compute and EcoLab_point::move.

5.2.6 Class EcoLabDemoModel

This class inherit EcoLab_grid and Classdesc::TCL_obj_t. It has the initial model parameters and it is mainly used to start the program with the method init() which initialize the grid and also has doSomething() that executes EcoLab_grid::step(size_t nmapxy). Through the use of the inherit Classdesc::TCL_obj_t and the macro Classdesc_ACCESS(EcoLabDemoModel); it can be accessed by a TCL script that has access into the entire C++ model,

5.2.7 Communication Description

Because this model is implemented in a cluster, the processors could be allocated in a different machines. So now the model has to have a communication mechanic to acknowledge the different grid states in a running execution. This communication mechanic is ruled by OpenMPI which is an implementation of MPI. However, though the use of the tools that EcoLab provides, The model implementation uses a high-level abstraction of openmpi which is the Graphcode class and the graph structure that it uses. In a graph, each process has a map of the graph whether they own the cell or are in a remote processor so if a local cell requires the information of a another cell, Graphcode already knows how to handle the communication. However, these cell objects need to be encapsulated as a Classdesc pointer aka Classdesc::ref in order to be serialize. Otherwise, the information of the cell could be lost because every process has different memory map. The map is represented by objref which is a class pointer that inherits from Classdesc::ref to actual cell object and additionally it has proc member which is attached to actual processor that own the cell and ID member that represent a id within the graph.

A short synopsis of Graph communication tools is as follows:

- **rebuild_local_list()**: Reconstruct the list of objrefs local to the current processor, according to the proc member of the objrefs.
- **Prepare_Neighbours()**: For each object on the local processor, ensure that all objects connected to it are brought up to date, by obtaining data from remote processors if necessary.
- **Partition Objects()**: Call the ParMETIS partition to redistribute the graph in an optimal way over the processors.
- **Distribute Objects()**: Broadcast graph data from processor which has rank 0, and call rebuild_local_list() on each processor.
- **gather()**: Bring the entire graph on processor which has rank 0 up to date, copying information from remote processors as necessary.

The model implementation needs to communicate in the functions is as follows:

- **EcoLab_grid::move(size_t nmapxy)**: Once all processors change their own grid cells proc member, the manner in which it acknowledges the changes along the grid is using gather() and distribute_object(). Once all grid is updated, in order to be balanced it calls to Partition_objects() and then calls rebuild_local_list(); for successfully acknowledge the balance.
- **EcoLab_grid::step(size_t nmapxy)**: Before to call to agents it calls to partition_objects to ensure that all objects are brought up to date.

5.3 Execution

In order to execute the implemented model it is necessary to create a TCL script. This script acts as the main file so is necessary to specify the initial parameters that are attached to EcoLabDemoModel member variables and the logical sequential calls to the model C++ functions in order to a correct behaviour.

The script should be alike to the one as follow:

```tcl
#!Demo_03_Model
proc simulate {} {
  uplevel #0 {
    set x 0
    set Total_time 0
    parallel use_namespace DemoModel
    set stopAt [lindex $argv(1)]
    parallel countOfAgents [lindex $argv(2)]
    parallel seed [lindex $argv(3)]
    parallel nmapx [lindex $argv(4)]
    parallel nmapy [lindex $argv(5)]
    parallel nmapxy [lindex $argv(6)]

    parallel init
    parallel addAgent
    while {$x < $stopAt} {
      puts "NewIteration"
      set Total [ time {parallel  doSomething} ]
      incr x
      puts $Total
    }
  }
}

#puts " [lindex $argv(1)] "
if {$argc==7} {simulate} \
else {puts "   USAGE IS: mpirun -np X ./
    Demo_03.tcl <stopAt> <countOfAgents> <seed> <
    nmapx> <nmapy> <nmapx*nmapy> "}
```

Listing 2: MainScript.tcl

In which: /beginitemize /item #!Demo_03_Model: is the executable file that results from the compilation of the model. /item use_namespace: allows to access into

Demo_03_Model variables, methods and functions. /item parallel instruction allows to all processors to perform a action simultaneously. /enditemize

6 Results

This section shows some results obtained from the EcoLab framework using the benchmark implementation described in section 5. The experiments have been executed in a machine with 72 cores (6 cores/ node). Although only four nodes were available.

First, I have conducted three different set of experiments in order to evaluate the scalability of the EcoLab framework.

The first set of experiments have the same number of agents(10000), an extra size of 256 B for each interaction ,a grid size of 300x300 units, and an extra amount of work for each agent corresponding to the computation of a FFT on a table of 16KB.

Figure 3 shows that EcoLab framework communications are low enough to allow an increase in performance.

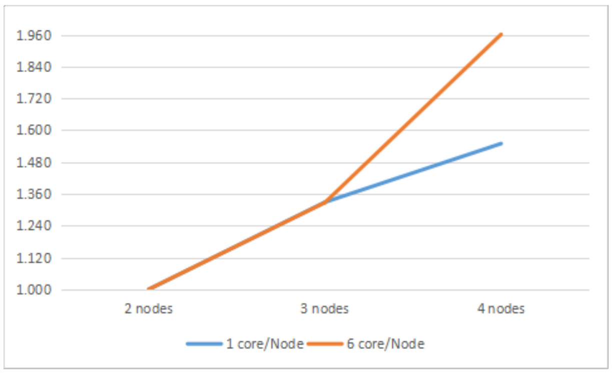

Fig. 3: Speed UP results from 4 to 24 nodes/core and from 2 to 4 nodes/core

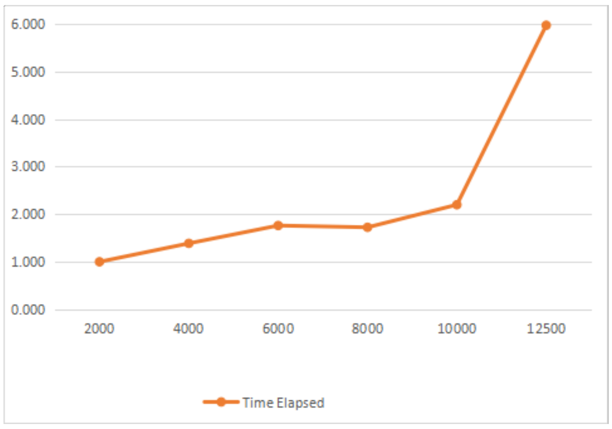

Fig. 4: Time elapsed results with 24 nodes/core varying the amount of agents

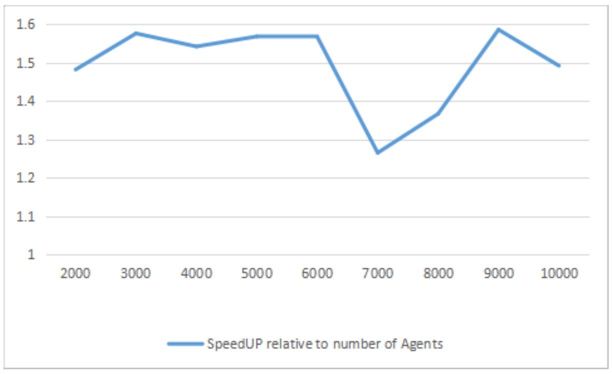

Fig. 5: Speedup of interactions of 16B relative to 256B

For the second set of experiments, the number of nodes/-core has been fixed to 24 and the number of agents has been varied from 2000 to 12500 rest is the same than previous set of experiments. Figure 4 shows that increasing the number of agents has a noticeable impact which is due by an increased amount of interactions and an increased amount in compute. It is particularly noteworthy that increase the number of agents adobe 10000 has an extra cost probably due to an non-sequential increment in the interactions costs.

For the third set of experiments, the experiments are alike the second set of experiments but now with the extra size of 256B were changed into 16B for each interaction. Figure 5 shows that in fact decreasing the size of interactions has a positive impact into the performance and also allows to have more agents in the same grid space of size 300x300units.

Lastly, I used a tracing instrumentation tool called TAU[18] in order to records a detailed log of events that allows to the user to see when and where routine transitions and communications take places. With this tool I can see an overview of the agents iteractions that takes place in the model developed.

In order to get a clear vision of the interactions between agents, I performed an experiment with 4 cores in which each core is allocated in a different node, with 10000 agents allocated and a extra size of 256B for each interaction a grid size of 300x300 units, and an extra amount of work for each agent corresponding to the computation of a FFT on a table of 16KB. Figure 6 shows a communication overview in which each yellow line represents a bunch of messages sent between processors. Figure 7 shows the initialization of the grid in which each process creates it own grid space. Figure 8 and 9 shows an DoSomething interaction, in these views we can see that the amount of work allocated in each processors changes in dynamically way. However it isn't balanced and according to the Partition Objects() function which uses PARMETIS it should be.

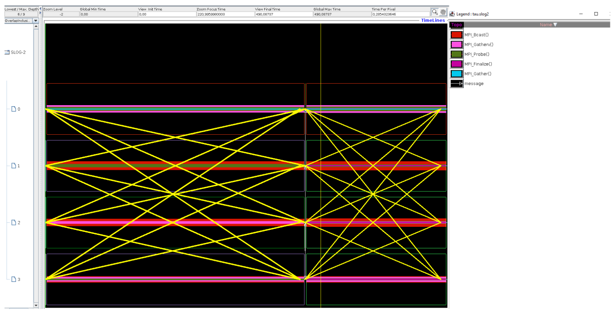

Fig. 6: A Communication overview

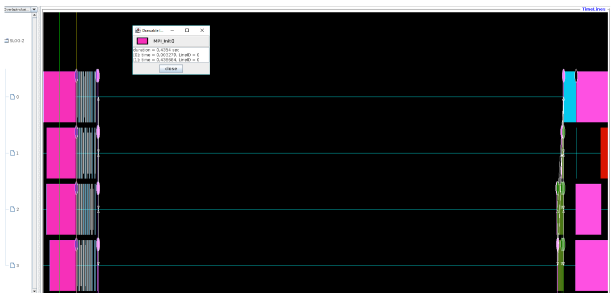

Fig. 7: Initialization of the grid and agents

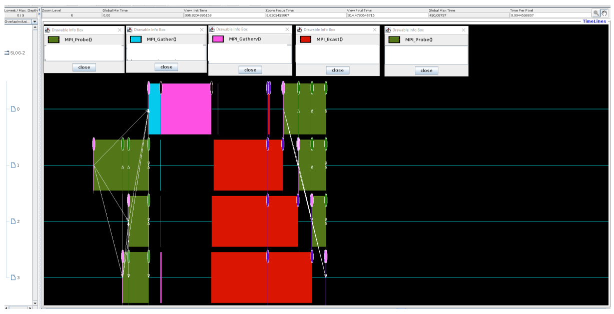

Fig. 8: DoSomething Communication Iter N

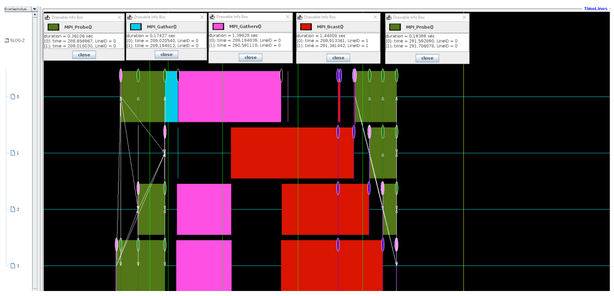

Fig. 9: DoSomething Communication Iter N+1

7 Conclusions

Although one could think that the installation is an trivial task it wasn't. I see there a somewhat big downside with concerning EcoLab platform because it has zero information about how to install it locally in a Linux distribution so one can go through a lot of problems that isn't documented. Over the course of the installation process I had to install one to one dependencies that figured in EcoLab documentation the problem resides in the ones no listed caused by the Linux distribution that is explained is only ubuntu while the implementation is allocated in a CentOS cluster. Furthermore, in a local installation, one must edit manually every old-fashioned makefile of the EcoLab platform in order to correctly set path and even compilation flags that doesn't correctly set in the parameters options.

The others problems are related to the execution and usage of the compiled models. The models and his agents has to been written in one file, the TCL script has to have the same name as the binary associated to the model and must be called in the first sentence of the script. This doesn't need to be a problem but I wasn't be able to locate these little details in the documentation.

Implement your own model in the EcoLab platform is hard to do in a first instance, mostly by the leak of explained examples. However as you acquire experience with it becomes easier. This is so nice because I got the felling that once you implement one model, if the platform possibility convince you, you can implement the next one much more easier.

Another reason about this difficult is the little few users of the application. thus,cause an important leak on example implementations and documentations apart from the author ones. Also, this has a negative impact into the EcoLab platform because it potential can be easily hidden.

I must say that , as well as the author, that I'm agree with him about the flexibility of the platform. This thought is because you can handle the communication using an high-level library through the use of Graphcode, which is capable of handle a high ratio of possibles HPC models, but it also use a lower-level library through the use of ClassdescMP which is similar to MPI and mostly extends it serialization competence.

As future work this could be expanded in others HPC environments and compare its performance with different hardware configurations also it would be nice to compare the EcoLab framework with others ABM like RepastHPC and Flame.

8 Acknowledgements

This project would have not been possible without DASCO cluster. Special thanks to my tutor, Anna Sikora and Eduardo César from the Autonomous University of Barcelona (UAB).

References

[1] Anna Sikora Andreu Moreno and Eduardo César. "Unpublished: Designing a Benchmark for Assessing the Performance of Parallel Agent-based Simulation Applications". 2018.

[2] *Berkely DB*. `http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html`. Accessed:21-04-2018.

[3] Eric Bonabeau. "Agent-based modeling: Methods and techniques for simulating human systems". In: *Proceedings of the National Academy of Sciences* 99.suppl 3 (2002), pp. 7280–7287.

[4] *Cairo library*. `http://cairolibrary.org/`. Accessed:21-04-2018.

[5] AL Chin et al. "Flame: An approach to the parallelisation of agent-based applications". In: *Work* 501 (2012), p. 63259.

[6] *Cluster definition*. `https://web.archive.org/web/20111027215420/http://www.clusters.nom.es/`. Accessed:21-04-2018.

[7] GNU GCC. *GCC C++ Standard Library*.

[8] Ananth Grama et al. *Introduction to parallel computing*. Pearson Education, 2003.

[9] Rolf Hempel. “The MPI standard for message passing”. In: *International Conference on High-Performance Computing and Networking*. Springer. 1994, pp. 247–252.

[10] George Karypis. “METIS and ParMETIS”. In: *Encyclopedia of parallel computing*. Springer, 2011, pp. 1117–1124.

[11] Paulo Leitão, Udo Inden, and Claus-Peter Rückemann. *Parallelising multi-agent systems for high performance computing*. Vol. 6. 2013, p. 1.

[12] Duraid Madina and Russell K Standish. “A system for reflection in C++”. In: *Proceedings of AUUG2001: Always on and Everywhere* (2001), p. 207.

[13] Duraid Madina and Russell K Standish. “A system for reflection in C++”. In: *Proceedings of AUUG2001: Always on and Everywhere* (2001), p. 207.

[14] Duraid Madina and Russell K Standish. “A system for reflection in C++”. In: *Proceedings of AUUG2001: Always on and Everywhere* (2001), p. 207.

[15] John K Ousterhout and Ken Jones. *Tcl and the Tk toolkit*. Pearson Education, 2009.

[16] *PNRG library*. `http://statmath.wu-wien.ac.at/software/prng/index.html`. Accessed:21-04-2018.

[17] *Readline library*. `https://launchpad.net/ubuntu/+source/readline`. Accessed:21-04-2018.

[18] Sameer Shende. “Profiling and tracing in linux”. In: *Proceedings of the Extreme Linux Workshop*. Vol. 2. Citeseer. 1999.

[19] Russell K Standish and Richard Leow. “EcoLab: Agent based modeling for C++ programmers”. In: *arXiv preprint cs/0401026* (2004).

[20] Russell K Standish and Duraid Madina. “Classdesc and Graphcode: support for scientific programming in C++”. In: *arXiv preprint cs/0610120* (2006).

[21] *Unuran library*. `http://statmath.wu-wien.ac.at/unuran/`. Accessed:21-04-2018.

[22] Brent B Welch, Ken Jones, and Jeffrey Hobbs. *Practical programming in Tcl and Tk*. Prentice Hall Professional, 2003.

[23] *Zlib library*. `https://zlib.net/`. Accessed:21-04-2018.