TerraME HPA: Parallel Simulation of Multi-Agent Systems over SMPs

Work in Progress Paper

Saulo H. C. Silva, Tiago G. de S. Carneiro, Joubert de C. Lima and Rodrigo R. Pereira
Federal University of Ouro Preto
{saulo.henrique.cabral, tiagogsc, joubertlima, rreisp}@gmail.com

ABSTRACT

Construction of prognoses about environmental changes demands simulations of massive multi-agent models. This work evaluates the hypothesis that the combined use of techniques such as annotation and bag of tasks can result in flexible and scalable platforms for multi-agent simulation. Although these are well known techniques, most environmental modeling platforms use other approaches to provide high performance computing. In general, the approach used is dependent of the modeling paradigm theses platforms implement. We are looking for approaches that can cope with multiple modeling paradigms. To evaluate our hypothesis, the TerraME modeling platform was extended to run over SMPs (Symmetric Multiprocessors) architectures and used in real case studies. While annotation allows modelers to implement different parallelization strategies without prevent models to run over sequential architectures, the bag of tasks provides load balancing over multiprocessors. The results demonstrated that 35% of linear speedup can be obtained for models with high dependence among tasks, when 8 processors are used. Moreover, for models that have low data or control dependencies, around 90% of linear speedup can be obtained.

Categories and Subject Descriptors

I.6.7 [**Simulation Support Systems**]: Performance and scalability of multi-agent simulation environment – Design, studies and measurement.

Keywords

Spatio-temporal, Simulation, Parallel, Multi-Agent, TerraME.

1. INTRODUCTION

The construction of prognoses about environmental changes demands simulation of massive multi-agent models. In this context, MASON [6,9,10] and Repast [4,5] platforms for agent based modeling stand out for offering parallel modeling as an alternative solution to both data intensive and high performance computing.

Control and data dependencies, imposed by the application domain, turn the automated model parallelization a challenge. Moreover, the scalability of a parallel model is determined by many variables, including the partitioning strategy that, in general, depends on the modeling paradigm used, among them: Systems Theory, Agents Theory, and Cellular Automata Theory. This way, there is a big counterpart between model performance and parallelization transparency when we develop parallel models. The less the modeler needs to deal with problems like partitioning, concurrency control, or processes synchronization and communication, in general, less will be the performance gain. Balancing performance requirements and programming simplicity is a relevant challenge, since modelers are, generally, experts in the application domain with low programming skills. In general, modelers are environmentalists, geologists, anthropologists, etc.

This work evaluates the hypothesis that the use of annotation technique for parallel model programming together with a bag-of-task technique for load balancing results on a multi-agent modeling platform with high flexibility and scalability. For this, the environmental modeling platform TerraME was extended to be efficiently executed over symmetric multiprocessors systems – SMPs, i.e., hardware architectures in which several processors share a single memory system and a single Operating System instance. We conduct experiments to analyze TerraME HPA *(High Performance Architecture)* platform performance. Results demonstrated that TerraME HPA is scalable and provides a simple application programming interface (API), enabling TerraME models portability from sequential to SMP architectures. TerraME HPA platform obtained 35% of linear *speedup* when simulates TROLL [3], an individual based model that has large sequential pieces and simulates tropical rainforest growth in 3D spaces. We also test TerraME HPA platform with low dependency models. It obtained 90.7% of linear *speedup* simulating a spatial version of prey-predator model. In TerraME HPA the modeler easily defines segments of code that have to be executed in parallel, synchronization barriers, and critical sections in which shared resources can be correctly accessed.

The remainder of this article is organized as follows: TerraME environmental modeling platform is presented in section 2. Section 3 qualitatively compares TerraME HPA approach with MASON and Repast, two promising multi-agent platforms for spatiotemporal simulation. In section 4, both TerraME HPA architecture and experiments are detailed. Section 5 presents the results of experiments. Section 6 presents the final remarks.

2. TerraME

TerraME [2] is a platform to develop dynamic spatially-explicit models which integrates multiple modeling paradigms, including agent based. In TerraME, environmental phenomena are represented by *Environment* objects. Environments can be considered micro-worlds. The landscape of an environment is defined by *CellularSpace* objects, that is, irregular grid of cells

SIGSIM-PADS'13, May 19–22, 2013, Montréal, Québec, Canada.

where cells may have several attributes: Land use, altimetry, temperature, etc. Actors and change processes are represented by *Agent* and *Automaton* objects embedded in the *Environment*. An *Environment* can have many *Timer* objects too. Each *Timer* is a discrete event scheduler that determines the instant in which the *Agent* and *Automaton* objects will be executed. At each event, Agent and Automaton objects percept changes in the *Environment* and react on it, changing it and themselves. A complex phenomenon can be represented by an hierarchy of *Environment* objects, in which lower level modules implements some specific properties of the phenomenon and are coordinated by higher level modules. These ideas are illustrated in Figure 1.

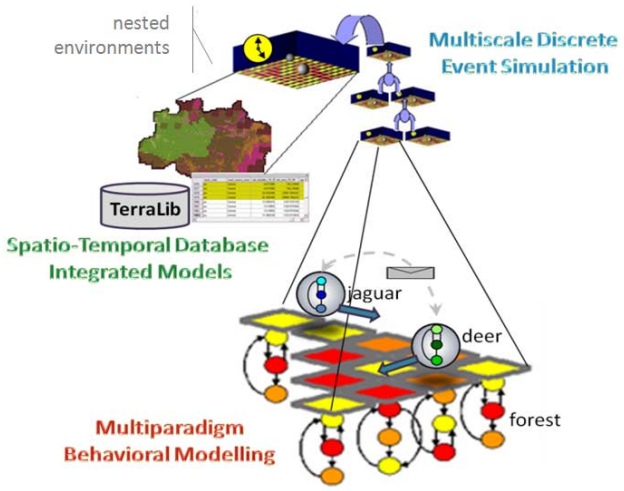

Figure 1: TerraME hierarchical concepts

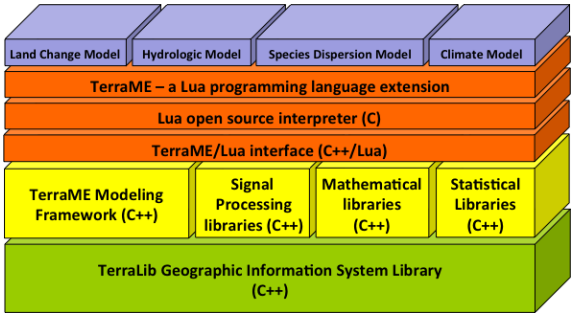

Figure 2. TerraME architecture [2]

Figure 2 illustrates TerraME tiered architecture. The user models are located on the first tier. They are written in a high level programming language named TerraML (TerraME Modeling Language), which extends Lua script language [8] with new data types. Each TerraML model instruction is interpreted by TerraME interpreter, located at second tier. It calls services of TerraME simulation engine located at the third tier. TerraME interpreter and simulation components are implemented in C++. They use TerraLib C++ library [1] to access geographic databases.

3. RELATED WORK

Although there are many agent based modeling platforms, including the pioneer Swarm [12] and the highest-level Netlogo [14], MASON [9,10] and Repast [4,5] are the most prominent high performance agent based modeling platforms used for environmental modeling. JDevs [7] is the only modeling framework that allows a wide range of modeling paradigms be expressed in the DEVS formalism. However, it does not provide a implementation for high performance computing.

The sequential version of the Repast toolkit can be programmed in Java. However, the Repast HPC is complete novel version of Repast specially designed for taking advantage of distributed memory hardware architectures. It was implemented in C++ over the MPI (Message Passing Interface) middleware, obligating modelers to explicit write senders, receivers and messages. There is no specific version for shared memory hardware architecture. This approach fits well into multicomputer architectures. However, in multicore machines and sometimes also over SMPs, MPIs implementations impose an overhead. The overhead is generated by the marshaling of message parameters and by the transmission of messages through network protocol stacks [13]. Fortunately, messages are not necessary in a shared memory system, so the use of MPI over SMPs can be unnecessary. In general, message passing based applications are more complex to be coded than shared address based applications.

Repast uses a master discrete event scheduler to iterate the simulation forward. It coordinates local schedulers spread among MPI processes. Each process is responsible for the execution of local agents. Local schedulers only execute events according to the lowest clock tick (interval between consecutive events) in the simulation. Copies of non-local agents may reside on a process, allowing agents to be shared across processes. Local agents can then interact with these copies. Processes are automatically synchronized when they require copies of agents from another process. However, in order to allow automatic synchronization, the modeler must provide the agent serialization and deserialization code.

MASON platform is developed in Java and provides a hybrid solution that can take advantage of distributed and shared memory hardware architectures. In MASON, a master process coordinates several worker processes located on different processors. The master divides the space inhabited by agents in several regions and each one is assigned to a worker. Each worker process has a discrete event scheduler that simultaneously runs agents that inhabit a particular region. In order to assure coherent computations, all communications with agents in neighbor regions (processors) are simulated before the execution of agents inside a region and the simulation is automatically synchronized at each time step. Thus, modelers have no worries about the coherency of computations. However, if several agents move to a single region, MASON may suffer from the imbalance of workload in different processors. Moreover, the number of worker processes in the simulation needs to be manually defined by the modeler. This fact can be a drawback because if the modeler defines more workers than the number of processors available in the hardware, the whole simulation performance may decrease due to workload imbalance or due to processes management overhead.

4. TerraME HPA approach

In order to allow the parallelization of models that use multiples modeling paradigm, TerraME HPA has been designed as a TerraML middleware, which is not dependent on TerraME specific data types and is capable of parallelize any TerraML (Lua) function. Decisions about problem partitioning, processes synchronization and concurrency control are left to the modeler. This approach can make model parallelization harder, but allows modelers fit parallelization strategies in order to achieve maximum performance.

TerraME HPA provide an API that allows modelers to define which functions should be run in parallel, barriers of synchronization, and critical sections. This API is provided in the form of annotations, i.e., structured comments that are inserted in the model source code. Thus, the annotated model source code can also run in a sequential version of TerraME, facilitating model verification. Before interpreting and executing a model, TerraME HPA automatically translates the annotated version of the model source code into a parallel version with explicit calls to TerraME HPA parallelization services. Then, the model is interpreted by the parallel version of TerraML (Lua) interpreter developed in this work.

Once parallel functions are called inside a model, they are instantiated and go to a bag of tasks waiting to be executed, and the model resumes its execution. This way, parallel functions are executed asynchronously by TerraME HPA threads named *workers*. In order to balance the workload among the hardware processors, TerraME HPA keeps a worker thread by processor, which continuously consumes parallel function from the shared bag of tasks. Thus, TerraME HPA attempts to ensure maximum utilization of available processors.

For while, TerraME HPA only provides services for parallel execution of functions over shared memory hardware architectures. A version for distributed memory hardware architectures is still under development.

4.1 TerraME HPA programming interface

The following methods are implemented in the TerraME HPA parallel application programming interface:

- HPA PARALLEL: Tells the interpreter that the next function found in the source code should be executed in parallel.
- HPA JOIN *Fx*: Defines a synchronization barrier, i.e., waits until all instances of the function *Fx* finish running.
- HPA JOINALL: Defines a synchronization barrier, i.e., waits all instances of parallel functions finish running.
- HPA ACQUIRE *Gl*: Sets the entry point of a critical section named *Gl* in order to ensure mutual exclusion inside the critical section. Just one parallel function get inside *Gl* until a release call is found.
- HPA RELEASE *Gl*: Indicates the end of a critical section, releases the access to the critical section *Gl*

4.2 Annotations: A flexible strategy

Annotations are structured comments in which methods of the TerraME HPA parallel API are called. Figure 3 illustrates an agent based model simulating preys and predators embedded in a *closed environment*. They are located in a unique *Cell* of a CellularSpace and can move to another during the simulation. Each predator can attack a prey using the *ag.attack(ag1)* method call, in which variables *ag* and *ag1* are predator and prey agents, respectively. During an attack, predators seek for preys in neighbor cells. If they find a prey they kill it and move to its Cell. Otherwise, they stay in the same place. Preys move randomly through empty cells.

In Figure 3, functions *execAgentPred* and *execAgentPrey* responsible for executing an given agent. Then, the model iterates over a set of agents and execute them concurrently. There are two synchronization barriers to wait all agents to finish one simulation step. Predators are always simulated before preys. The *forEachNeighbor* function iterates over cells in neighborhood of the *Cell* received as its first parameter. At each neighbor cell, it applies the function received as its second parameter. The *forEachAgent* function iterates over agents embedded in the *Cell* received as its first parameter applying the function received as its second parameter. The ag:*getLocation*() function returns the *Cell* where the agent *ag* is inserted. The command *cs = CellularSpace{ xdim = DIMENSION }* creates a squared grid of DIMENSION x DIMENSION cells, where DIMENSION is a integer constant.

```
predator = function()
    ag = {energy = 20, type = "predator"}
    ag.execute = function(self, ag1)
        attack();
    end
    ag.attack = function(self, ag1)
        forEachNeighbor(self:getLocation(), function(cell, neigh)
            forEachAgent(neigh, function(ag1)
                if ag1.type == "prey" then
                    ..DO SOMETHING TO KILL AND MOVE..
                end
            end)
        end)
    end
    ag.class = "predatores";
    ag.state = State{ id = "hunting" }
    return Agent(ag)
end
cs = CellularSpace{xdim = DIMENSION}
predatores = {}
for i = 1, POPUL_PREDATOR do
    ag = predator();  cell = cs:sample()
    ag:enter(cell);  table.insert(predatores, ag)
end

function execAgentPred(i)
    predatores[i]:execute()
end
function execAgentPrey(i)
    preys[i]:execute()
end

for i = 1, #predatores do
    --HPA PARALLEL
    execAgentPred(i)
end
--HPA JOINALL
for i = 1, #preys do
    --HPA PARALLEL
    execAgentPrey(i)
end
--HPA JOINALL
```

Figure 3: Prey-predator agent based model in TerraML.

4.3 Bag of tasks: As load balance strategy

Most modeling platform are based on a unique modeling paradigm, allowing simulation engine architects to make several design decision about parallelization, communication and synchronization strategies. This way, most tasks involved in the development of parallel models are treated transparently by the simulation engine. For instance, in agent based modeling platform, agent computations usually depends on properties of the area where they are located. So for problem partitioning purpose, it is common to divide the data structure representing the space into several regions and assign each region to a different processor. Consequently, a processor is responsible for simulating all agents inserted into its region. And therefore, the concentration of agents in a certain region may cause imbalance in the system workload.

As stated before, TerraME aims to support the simultaneous use of several modeling paradigm for the development of a unique model. This is a important requirement for integrated environmental modeling, once no modeling paradigm alone is able to represent all the complexity found in Terrestrial phenomena: Land use and cover change, global change, disease spreading, fire propagation, etc. So, a higher level of abstraction is necessary to conceive a simulation engine capable to concurrently run models based on multiple paradigms. For this reason the TerraME HPA has been developed as a TerraML (Lua) parallel middleware, i.e., a TerraML interpreter that is capable to run functions in parallel and control concurrent access to shared variables. This new interpreter has been implemented in C programming language and replaces the Lua interpreter in the TerraME tiered architecture, Figure 2.

In opposition to Lua interpreter that keeps only one execution stack in which all computation are performed, TerraME HPA keeps track of several execution stacks. The model has a main execution stack where shared variable are stored and each parallel function has its own execution stack where local variable are stored. Concurrent accesses to the main stack are transparently controlled by TerraME HPA interpreter that ensures mutual exclusion and atomicity to the operations. This way, TerraME HPA assures the coherency of computations over shared variables.

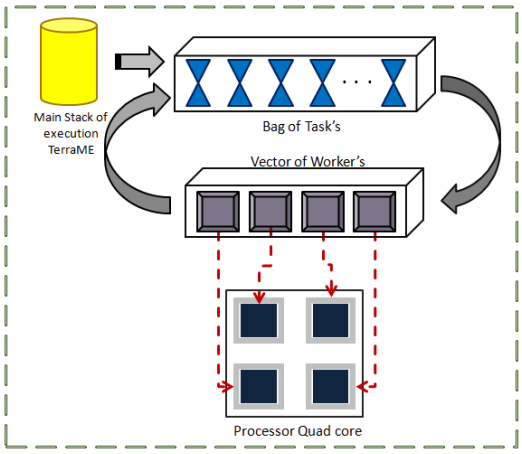

Figure 4: TerraME HPA parallelization mechanism.

When TerraME HPA interpreter starts, it creates a global data structure named bag-of-task in which parallel functions are temporally stored waiting to be executed. Also, it automatically identifies the number of available processors and starts one C++ thread by processor, avoiding a profusion of threads in the system whose exceeding context swap might slowdown model runtime. These threads are named *workers* and are responsible for continuously consume parallel functions from the bag-of-tasks, executing them one after the other, in a attempt to maintain the processors utilization near to 100% and keep the workload balanced. Figure 4 illustrate the TerraME HPA parallel architecture.

When a parallel function is called, it is instantiated and is inserted in the bag-of-tasks data structure. The interpreter collects its name, creates its local execution stack, pushes the arguments in the stack and saves the main stack address for the function returning value. Then, the model immediately resumes its execution. This way, TerraME HPA implements a asynchronous model of concurrency. In order to gather the correct result from the calling of a parallel function, the modeler must implement a synchronization barrier for waiting the end of its computation.

4.4 Agent communication

Although agents can communicate through the exchange of logical messages, the communication is implemented through shared variable. This mechanism is simple to the modeler and avoids the marshaling of message parameters and the transmission of messages through inter-process communication channels or networks protocols.

4.5 Critical Sections

Critical sections can be coded by the modelers at any granularity. A single variable or a collection of variables can be locked. Figures 5 and 6 show some snippets of the TROLL forest growth model. They illustrate how to implement coarse and fine grained critical sections in TerraME HPA. Figure 5 indicates that the interpreter will acquire the entire *summaryTable* shared variable when HPA ACQUIRE annotation will be executed. The variable *summaryTable* is a two-dimensional table that stores a integer number at each position. In contrast, Figure 6 illustrates a more efficient *summaryTable* access. Variable *lockOnePosition* indicates the finest lock granularity. Only the element in the position [cell.specie][MORTALITY_SUM_IDX] will be locked.

For each HPA ACQUIRE annotation, TerraME HPA interpreter inserts a new lock in a map of model variables. Each worker thread points to a shared map of variables. If a shared variable is acquired, the current worker waits until a notification occurs, indicating the global variable is released. This wait/notify mechanism avoids busy-waiting in TerraME HPA.

```
--example where block all positions of variable summaryTable
local lockAllPositions = "summaryTable"

--HPA ACQUIRE lockAllPositions
summaryTable[cell.specie][MORTALITY_SUM_IDX] =
summaryTable[cell.specie][MORTALITY_SUM_IDX] + 1;
--HPA RELEASE lockAllPositions
```

Figure 5: Coarse lock

```
--example where block just one position of variable summaryTable
local lockOnePosition = "summaryTable"..cell.specie..MORTALITY_SUM_IDX

--HPA ACQUIRE lockOnePosition
summaryTable[cell.specie][MORTALITY_SUM_IDX] =
summaryTable[cell.specie][MORTALITY_SUM_IDX] + 1;
--HPA RELEASE lockOnePosition
```

Figure 6: Fine lock

5. EXPERIMENTS

Two multi-agent models were used in the performance analysis experiments. In first experiment, a model with no task dependencies simulates an *Environment* formed by isolated cells in which prey-predator populations evolve with no influences of neighbor populations. This way, the modeler computes all predators in parallel firstly and then all preys in parallel. At each simulation step, there are two synchronization barriers: one to wait predators computations and a second to wait preys computations. The prey-predator implementation avoids simultaneous prey-predator executions, so it avoids costly synchronizations during a unique simulation step. Figure 3 illustrates the main part of prey-predator model in TerraML.

In the second experiment, a complex model with several synchronization points and a global variable is simulated. A *CellularSpace* is shared by *Agent* objects, which simulates the forest growth process in three dimensions. The TROLL model considers that in the interior of tropical rain forests individual trees compete for the available light shadowing each other. Therefore, the computations on a 3D cell depend on the neighbor trees [3]. In both experiments, the shared *CellularSpace* was split into several logical sub-spaces embedded into nested *Environment* objects. *Environments* were simulated concurrently.

Figure 7 illustrates the costly synchronization point of TROLL model. The function *calcLeafAreaIndex* is responsible for calculating the index of leaf area in a cell. Unfortunately, the computation of this index depends on the value of leaf area index of the neighbor cells. A partition strategy to avoid too many synchronizations can be based on height-layers. The computation is concurrently executed from the top of the forest to the bottom, the layer L1 (highest) is computed, then TerraME HPA synchronizes, then a second layer L2 is processed, then a second synchronization occurs, and so on.

```
function calcLeafAreaIndex_( p, vertical )
    --vertical parameter indicates if the forest this tilted

    if ( vertical ) then

        --[[the upper layers depend on the values of the lower
        layers is thus necessary to parallelize by layers]]
        for h, layer in ipairs(csTable) do

            --calculates to all cells in layer
            for idCell = 1, #layer.cells do
                --HPA PARALLEL
                calcLeafAreaIndex_1(p,idCell,h)
            end
            --HPA JOIN calcLeafAreaIndex_1
        end
    else
        for h, layer in ipairs(csTable) do

            for idCell = 1, #layer.cells do
                --HPA PARALLEL
                calcLeafAreaIndex_2(p,idCell,h)
            end
            --HPA JOIN calcLeafAreaIndex_2
        end
    end
end
```

Figure 7. Function *calcLeafAreaIndex* of model TROLL

The main reason to implement two models in experiments, one with low dependencies and other with high dependencies, is to illustrate that TerraME scales according to the modeler implementation. Efficient partition strategies, with few synchronizations and few serial code are always desired, but not often obtained.

Experiments were performed on a machine with OS Windows Server 2003 64-bit with 16 GB RAM DDR2 667 MHz and 2 Intel Xeon processors, summing 8 cores. Each experiment was executed 5-10 times, depending on its size. The runtime variance is calculated and then the speedup of each model experiment is calculated. The results are presented in the next section.

6. RESULTS

Figure 8 presents the reduction of simulation time for the prey-predator model. The use of two cores reduces the time by half. With 8 cores, the simulation time almost reaches the asymptotic lower bound. The parallel model scales very well, presenting 90.7% of linear speedup compared with the sequential model version.

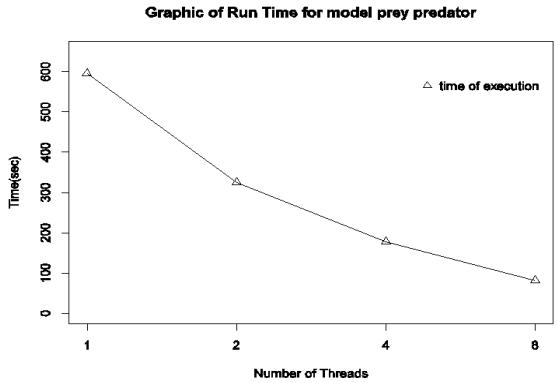

Figure 8. Prey-Predator simulation runtime

Figure 9 presents the reduction of simulation time for TROLL model. The use of two cores reduces the time by 35%. However, as the number of threads increases the concurrence also increases and the synchronization points become the bottleneck of the solution. Simulations with eight processors achieve speedup of only three. Nonetheless, it is out of scope of this paper to discuss new solutions for parallel implementations of TROLL.

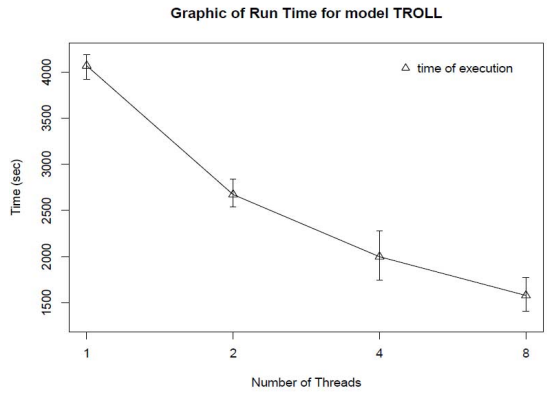

Figure 9. TROLL simulation runtime

Figure 10 shows that TerraME HPA can deliver scalability to multi-agent models, with low overhead in concurrency management. While the runtime of the sequential prey-predator model presents an exponential growth as the number of agents increases, the runtime of the parallel model presents a very slow growth.

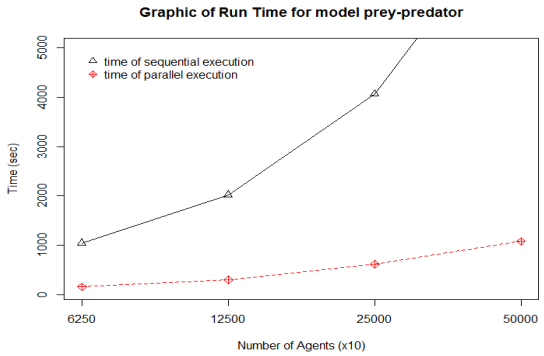

Figure 10. Prey-Predator scalability

Figure 11 shows the increasing of memory consumption as the number of prey and predator agents grows. TerraME HPA imposes a small overhead in terms of memory consumption when compared with the sequential version of TerraME. In this case, the parallel simulation was conducted with 8 worker threads. During the simulation, the average utilization of 8 available processors was 95.667%.

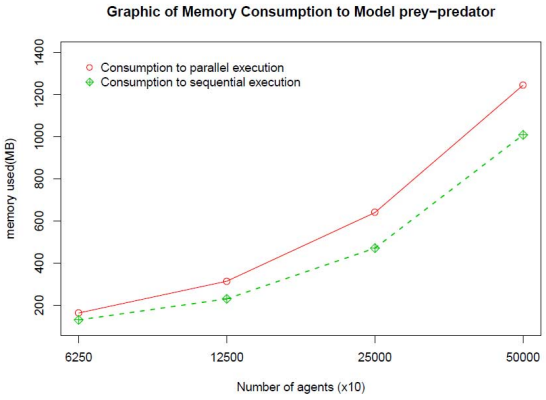

Figure 11. Prey-Predator memory consumption

7. FINAL REMARKS

The results demonstrated that the annotation technique allows modelers define model parallelization granularity and strategy without preventing that the model source code could be executed by sequential machines. Verification, debugging and validation of sequential models on sequential machines is easier than on parallel ones. Modeler has freedom to take most advantage of SMP architectures, i.e., a simple shared address space. However, the approach provides no transparency on parallelization problems and decisions

Multi modeling paradigms are addressed by TerraME HPA flexible design. Our experiments demonstrate that TerraME scales well when the modeler designs efficient parallel solutions.

The bag-of-task technique is useful for load balance among multiple processors. There is no saturation in the number of workers and they keep working task after task. Unfortunately, one task can be harder to compute then others, but for TerraME HPA they have equal priorities, so their balanced CPU usage depends only on the model design.

The results obtained by the proposed approach in multi-agent experiments can be extended to models based on other modeling paradigms, as General System Theory and Cellular Automata Theory. TROLL experiment demonstrates that TerraME is useful to real case studies. TROLL experiment also showed that very often real problems are hard to be partitioned. The first parallel and open source TROLL version can be tested against new parallel alternatives. Huge instances of TROLL can now be simulated using TerraME, opening new perspectives for better understanding tropical forests growth.

Future works include the development of solutions for distributed memory hardware architectures and GPU devices. Tests with some Operating Systems that offer SMP capabilities over multicomputer architectures are part of our actual work. We just started tests with *kerrighed* [11]. It is also necessary to develop new experiments to compare the performance and scalability of TerraME HPA with the performance and scalability of other modeling platforms.

8. ACKNOWLEDGMENTS

This work was partially funded by CAPES, CNPq (CTINFO 09/2010) and UFOP.

9. REFERENCES

[1] Câmara G., Vinhas L., et.al. TerraLib: An Open Source GIS Library for Large-Scale Environmental and Socio-Economic Applications, Open Source Approaches in Spatial Data Handling, pages 247–270. 2008.

[2] Carneiro, T. G. S. *Nested-CA: A foundation for multiscale modeling of land use and land change.* phd thesis in computer science, The National Institute for Space Research, Brazil, 2006.

[3] Chave, J. *Study of structural, sucessional and spatial patterns in tropical rain forest using troll, a spatially explicit forest model.* Ecological Modeling, 1999.

[4] Collier N. *Repast HPC Manual.* Argonne National Laboratory, Argone, IL, USA, 2012.

[5] Collier N. and North M., *Repast HPC: A platform for Large-Scale Agent-Based Modeling.* Computing Techniques for Complex System Simulation, November 29, 2011. ISBN-13: 978-0470592441.

[6] Cordasco G., Chiara D. R., et al. *A Framework for distributing Agent-based simulations.* EURO PAR 2011: PARALLEL PROCESSING WORKSHOPS, Lecture Notes in Computer Science, 2012, Volume: 7155/2012, 460-470, DOI: 10.1007/978-3-642-29737-3_51.

[7] Filippi, J., and Bisgambiglia, P.. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. Environmental Modelling and Software 19(3) 261-274. 2004.

[8] Ierusalimschy, R, *Programming in Lua*, Lua.Org, ISBN 85-903798-1-7. 2003.

[9] Luke S. *Multiagent Simulation And the MASON Library*. George Mason University, First Edition, August, 2011.

[10] Luke S., Cioffi-Revilla C., et al., *MASON: A New Multi-Agent Simulation Toolkit.* Proceedings of the 2004 Swarmfest Workshop. 2004.

[11] Lottiaux, R. ; Vallee, G. ; et al., Kerrighed and data parallelism: cluster computing on single system image operating systems. Proceedings of the International Conference on Cluster Computing, 2004.

[12] Minar, N., Burkhart, R., Langton, C., Askenazi , M.,. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulation. SFI Working Paper 96-06-042. 1996.

[13] Shahid Hussain and Hassan Shabbir. *Directory scalability in multi-agent based systems*. Master's thesis in Science in Software Engineering, School of Engineering at Blekinge Institute of Technology.2008

[14] Tisue, S., and Wilensky, U.. *NetLogo: A Simple Environment for Modeling Complexity*, International Conference on Complex Systems: Boston. 2004.