ParCeL-5/ParSSAP: A Parallel Programming Model and Library for Easy Development and Fast Execution of Simulations of Situated Multi-Agent Systems

Stéphane Vialle
SUPELEC
2 rue Edouard Belin
57070 Metz, France

Stephane.Vialle@supelec.fr

Eugen Dedu Claude Timsit
Laboratoire PRiSM
Université de Versailles St-Quentin-en-Yvelines
45 Avenue des Etats-Unis
78035 Versailles Cedex, France

Eugen.Dedu@prism.uvsq.fr
Claude.Timsit@prism.uvsq.fr

Abstract

This paper introduces a new parallel programming model for situated multi-agent systems simulations and its parallel library implementation on shared memory MIMD parallel computers. The first goal is to allow users to easily implement situated multi-agent systems, following their natural paradigm: concurrent agent behavior definition and environment update programming. The second goal is to obtain efficient parallel executions on shared memory MIMD parallel computers, without dealing with parallel programming difficulties (such as load balancing and processes synchronization).

We present our parallel programming model (ParCeL-5/ParSSAP), and the parallel algorithms we have developed for the simulation two agent percepts: direct vision and potential field detection. Finally, we give an example of complete multi-agent system programmation and its performance measurements on a 64-processors SGI-Origin2000, exhibiting easy development and good speed up.

1. Motivations

Situated multi-agent systems can be used to model and study many real systems, composed of entities acting simultaneously in an environment [2], such as traffic-jams, ant societies, or robot teams [5].

But precise simulations of simultaneity of agent actions, or simulations of large multi-agent systems can be very time-consuming [1]. Multi-Agent Systems generate a lot of computations but also a lot of interactions between agents and their environment which introduce an overhead. Moreover, simultaneity of agent actions lead to spatial conflicts that need frequently to be solved fairly for all agents, hence generating extra-computations. Then parallel execution is needed to maintain acceptable execution times.

From the programmer point of view, multi-agent system implementation is not obvious if we have to implement simultaneity of agent actions, environment update and agent perceptions (such as visibility area detection and odor propagation). From the performance race point of view, intrinsic parallelism of multi-agent systems (MAS) is not always compatible with modern parallel computer one. Modern and general purpose parallel computers are MIMD ones and have frequently a limited number of powerful processors, while situated MAS can have many agents, each one making small computations. Moreover, MAS environment update is not simple to parallelize. For these reasons, efficient implementation of situated MAS on MIMD computers is not straightforward. Adapted programming models and efficient parallel development tools are needed to limit development times and obtain efficient runs.

Many parallel programming languages for MIMD architectures exist [3], but none is particularly adapted to MAS paradigm. The different development environment existing for MAS are not really efficient on MIMD parallel computers. The lack of tools has lead us to design some programming models adapted to situated multi-agent systems, and to implement efficient parallel libraries dedicated to MIMD architectures. Our last model and library is ParCeL-5/ParSSAP, and it provides optimized parallelization of agent percepts and environment update.

2. Previous works

We have already developed a line of parallel programming tools (languages and libraries) since 1989, the ParCeL line (*Parallel Cellular Languages*) dedicated to distributed computing systems as situated multi-agent systems. They are based on *cell* network programming.

ParCeL-1 was first destined to Transputer-based parallel architectures (old distributed memory architectures) and some Artificial Intelligence applications with intrinsic parallel computing, such as neural networks and semantic networks [7]. It was efficient on Transputer-based architectures, but was too close to these architectures: it was easy to understand, but lacked expressiveness and dit not exploit features of more recent parallel computers. ParCeL-3 has been designed for shared memory and distributed shared memory architectures, such as SGI-Origin2000, and for a large wide of applications, including simulations of multi-agent systems [6]. It was a very generic development tool, highly expressive, but very complex to use. Even with the PIOMASS simulator [1] built on ParCeL-3, MAS simulator implementations remained too complex.

So, we designed ParCeL-5/ParSSAP, with a programming model specifically intended for situated multi-agent systems, but easy to use and with an efficient implementation on modern distributed shared memory machines. Other ParCeL projects are neural network parallel development tools, out of scope of this article.

3. Parallel programming model of ParCeL-5/ParSSAP

3.1. Basic components of the model

ParCeL-5/ParSSAP model is based on four basic components : the *environment*, the *resources*, the *agents* and the *arbitrator*.

Agents evolve inside an *environment*, spatially discretized as two dimensional square mesh or torus. Each box has a permanent *type*: empty, obstacle or resource, and a dynamic *state*: occupied or unoccupied (by an agent). Square boxes have 4-connectivity or 8-connectivity, communicating only by their edges or by their corners.

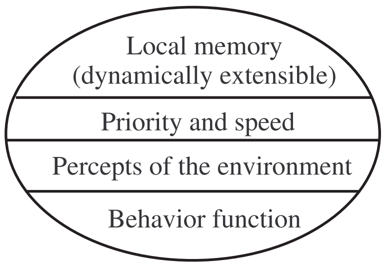


Figure 1: General view of an agent/cell of ParCeL-5/ParSSAP programming model.

Each box supports only one agent at a time, and an obstacle entirely fills a box.

Resources are entities that entirely fill a box, but still accept one agent inside. Resources store objects (wealths), that agents can take, move and drop, and can be perceived by agents from near or far boxes (depending of percept configurations). Each resource is visible by agents if it is not hidden by obstacles, and can propagate a decreasing potential. A user can define as many resources and as many types of resources as necessary. The type of a resource defines the type of the objects stored and the type of potential propagated.

Cells or *Agents* of ParCeL-5/ParSSAP are mobil agents, and have local memories that can be increased dynamically. Agents can be defined before the beginning of the simulation, or can be dynamically created after the simulation has started, from the behaviors of already existing agents. In both cases, agents can be destroyed dynamically. Each agent has a speed, corresponding to the frequency of its activation, and a priority used by the arbitrator during the conflict resolutions. Figure 1 summarizes agent features of ParCeL-5/ParSSAP model.

The *arbitrator* consists in a set of rules modelling the physical laws of the environment, automatically applied when agents attempt to act and when environment is updated. Arbitrator manages all conflicts caused by agent actions, such as several agents attempt to enter the same box at the same time. Then, a winner agent is randomly and fairly chosen by the arbitrator from agent with highest priority engaged in the conflict.

3.2. Main features of agent behaviors

ParCeL-5/ParSSAP model defines some percepts and actions of agents, that are useful to define agent behaviors, such as dynamic agent creation and destruction. But our model does not limit agent behavior design.

The only constraint of our model on agent percepts is their limitation to a local perception, in the spirit of agent concept. On the other hand, the number of agent percepts of our model is not limited. Today we have parallelized and implemented two classic percepts: *direct vision* and *potential field detection* [2], limited by a vision radius and a potential radius. Direct vision allows agents to know instantaneously the type and the state of a distant box, but it is stopped by obstacles. Resources and agents do not stop vision, but obstacles do. At the opposite, potential field detection allows agents to always reach the resources, since potential propagation is a wave propagation that bypasses the obstacles. But agents detect only the potential of their neighbor boxes (and look for increasing potentials), which provide them the direction to follow, and not the type and state of the boxes they are going to cross.

Agents can make different actions on their environment during the activation of their behaviors. Some actions can be cumulated in the same activation, others are exclusive. Sometimes they lead to conflicts among agents, resolved by the arbitrator. Currently, during its activation a ParCeL-5/ParSSAP agent can *move* in a neighbor box without obstacles or other agents, *take* or *drop* objects inside resources if they are not empty, respectively full, and *create* or *destroy* other agents. *Move*, *drop* and *take* actions are exclusive (either one movement, or one object manipulation per activation), but agent *creations* or *destructions* can be always cumulated indefinitely.

3.3. Timing model and cyclic running of MAS simulations

ParCeL-5/ParSSAP time model is a discret one, with a cyclic running of the MAS simulations. A simulation begins with a call to an initialisation routine of the environment and the resources and the agent percepts. Afterward, the simulation loop begins, composed of five steps per cycle (see figure 2).

Inside each simulation cycle, all agents are activated in parallel at the beginning of the step 1, but no action is executed, agents just compute and declare their action intentions. Afterward the arbitrator solves the conflicts, pointing out some winners at step 2, and actions of winners and non-conflicting agents are executed at step 3. Then, step 4 is dedicated to the execution of a function defined by the user. It is usual to save some simulation results on disks, or send data on simulation evolution to a visualisation system. This step can be sequential or parallel, depending on the user design and implementation, no constraint is added by ParCeL-5/ParSSAP model. Finally, step 5 updates the environment and percept data structures, as potential repropagation when potential resources has evolved.

3.4. Step of simulation analysis

The analyse of a multi-agent system is a complex problem, because it is composed of numerous autonomous entities (the agents), evolving in parallel and difficult to track. Various functionalities are needed to access to global system results (as total number of agents), to make a statistical analysis (as percentage of agent bringing object), and to save simulation evolution. So, ParCeL-5/ParSSAP model includes the concept of simulation analysis functionalities. It supplies some basic analyse and save functions, and includes a special step in its simulation loop to call these functions: the *user end-cycle function step* (see figure 2).

3.5. Parallel implementation of the model

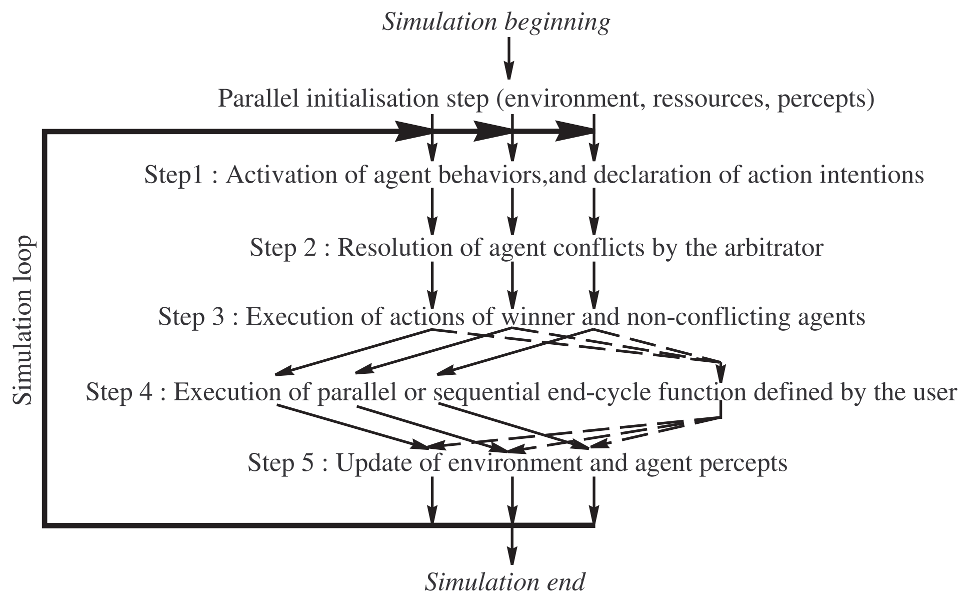
ParCeL-5/ParSSAP has been implemented on shared memory parallel computers using the multithreading paradigm: Distributed Shared Memory SGI-Origin2000, with up to 64 processors, using native Irix-Thread library, and classic shared memory SUN-Enterprise450, with up to 4 processors, using Posix-Thread library [4].

ParCeL-5/ParSSAP parallel implementation follows the ParCeL-5/ParSSAP model. Each processor concurrently runs a loop simulation (see figure 2), a synchronization barrier separates each step, and an agent can be easily implemented as a *cell*. But in order to be efficient on MIMD parallel computers, each processor manages several cells, and a cell is not implemented as a process (too few computations to be a process). It is implemented as a generic and minimal data structure easy to manage, pointing on specific data structures and a behavior function (see figure 3).

Moreover, time consuming algorithms engaged in agent percept simulations have been parallelized and optimized: visibility field computation and wave potential field propagation. They are described in the next sections.

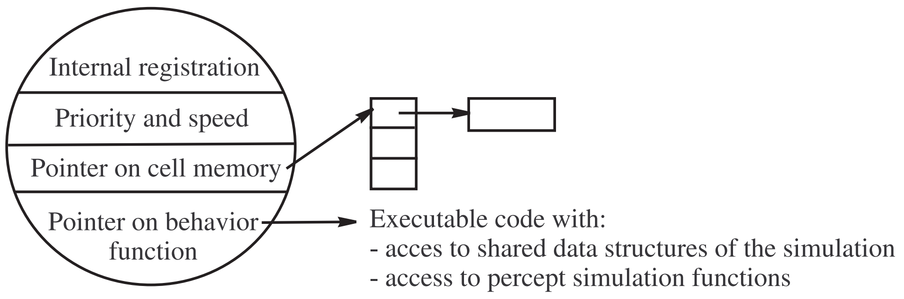
4. Parallelization of visibility field computation

Vision percept allows an agent to get informations on a distant box if there is no obstacle be-



Simulation beginning
Parallel initialisation step (environment, ressources, percepts)
Step1 : Activation of agent behaviors,and declaration of action intentions
Step 2 : Resolution of agent conflicts by the arbitrator
Step 3 : Execution of actions of winner and non-conflicting agents
Step 4 : Execution of parallel or sequential end-cycle function defined by the user
Step 5 : Update of environment and agent percepts
Simulation loop
Simulation end

Figure 2: Cyclic running of a ParCeL-5/ParSSAP simulation of MAS.



Internal registration
Priority and speed
Pointer on cell memory
Pointer on behavior function
Executable code with:
- acces to shared data structures of the simulation
- access to percept simulation functions

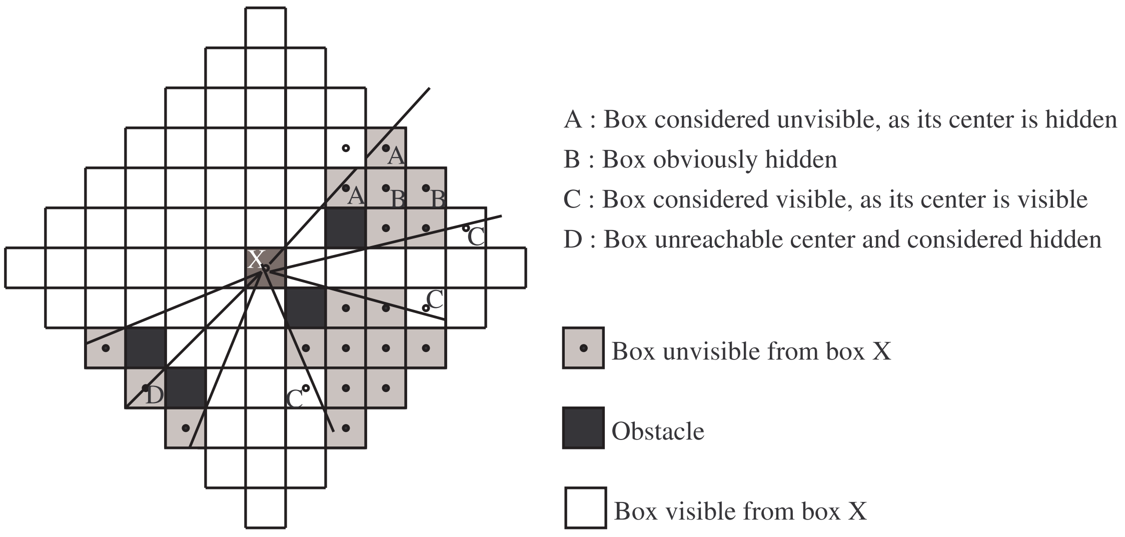
Figure 3: Generic data structure used for any agent/cell in our parallel implementation of ParCeL-5/ ParSSAP model.

tween the center of its box and the center of the distant box (see figure 4 for an example). Other agents and resources do not stop its vision. As obstacles are fixed, the visibility field from any box can be definitely computed at the initialization of the simulation. This initialization takes a long time but can be efficiently parallelized: visibility fields from different boxes are independent computations.

We use a ray-tracing algorithm to draw lines between box centers and to test if they encounter some obstacles. In order to avoid some round off problems during ray-tracing on our discretized two dimensional grid, we use a *supercover* line tracing algorithm, that draw thick lines and does not forget an obstacle corner, for example.

Our present parallelization is based on domain partitioning: each processor computes visibility fields of all boxes in a slice of the environment, using supercover line tracing algorithm. This parallel computation seems straightforward, but two problems remain: load unbalance if obstacles are not homogeneously spread on the environment (as they influence computations), and parallel memory allocations. Environment data structures and visibility fields are dynamically allocated, to adapt to simulation parameters introduced by users (as environment size and vision radius). As visibility field of one box can be used only by an agent situated on it, visibility field data structures can be accessed only by the processor in charge of this box. So, we decide to make dynamic allocation of this data structure on the processor managing this box, in order to help the OS to allocate a memory area near of this processor, not a big and far shared memory area. But parallel visibility field allocation has not scaled efficiently with standard memory allocation functions of the SGI-Origin2000, even when making only one allocation for all visibility fields of a processor. Dynamic memory allocation remains a sensitive issue in parallel programming.

Finally, we obtain the very good but not lin-



X
A
B
C
D
A : Box considered unvisible, as its center is hidden
B : Box obviously hidden
C : Box considered visible, as its center is visible
D : Box unreachable center and considered hidden
Box unvisible from box X
Obstacle
Box visible from box X

Figure 4: Visibility field from box X, with a 4-connectivity and a vision radius of 6 box length.

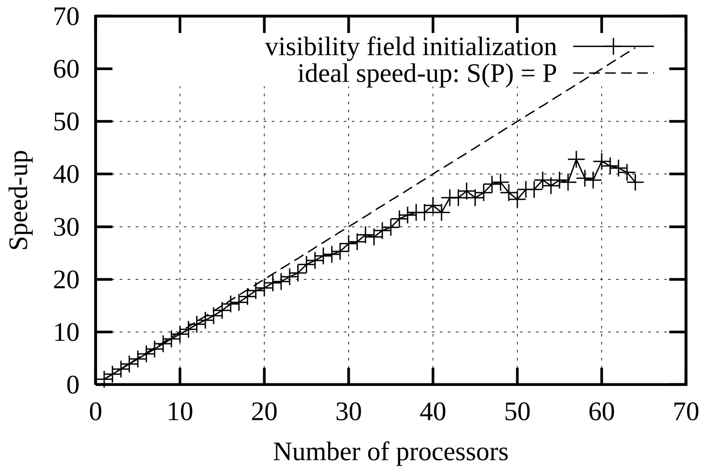


Figure 5: Example of speed up of parallel visibility field allocation and initialization on a SGI-Origin2000.

ear speed up of the figure 5, for an environment of 512 × 512 boxes with no obstacles and a 4-connectivity and a vision radius of 8 box length, on a SGI-Origin2000. Fortunately, usage of these visibility-fields is automatically donne in parallel from parallel run of agent behavior functions, no more algorithmic efforts are needed.

5. Parallelization of wave potential field propagation

Resources can emit some potentials that spread and decrease, and bypass obstacles. Agents detecting simple potential gradients on neighbor boxes follow increasing potential directions and finally reach resources quickly, avoiding obstacles and moving on shortest paths (see figure 6). This percept models a kind of odor propagation and detection. When different potential fields meet, a *max()* operator is applied: each box reached by several potentials takes the maximum of these values (see the potential field frontier on figure 6). But different resource types emit different kinds of potential that do not mix but overlap. So, resources of different types coexist without any damage.

A resource potential usually matches a resource feature, such as the number of objects it still contains, and evolves during the simulation. So, a regular potential field update is needed, and a parallelization is welcome as it is a time consuming computation. At the opposite, potential field exploitation from each agent behavior is obviously parallel, and need no particular algorithmic effort.

Two main parallelization strategies have been studied: environment partitioning among pro-

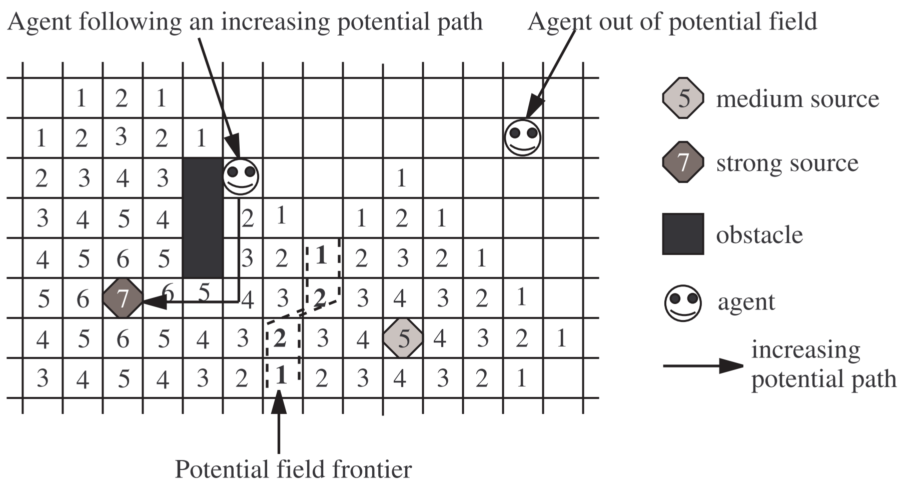

Agent following an increasing potential path
Agent out of potential field
medium source
strong source
obstacle
agent
increasing potential path
Potential field frontier

Figure 6: Example of wave potential fields, skirting obstacles and attracting agents along shortest paths.

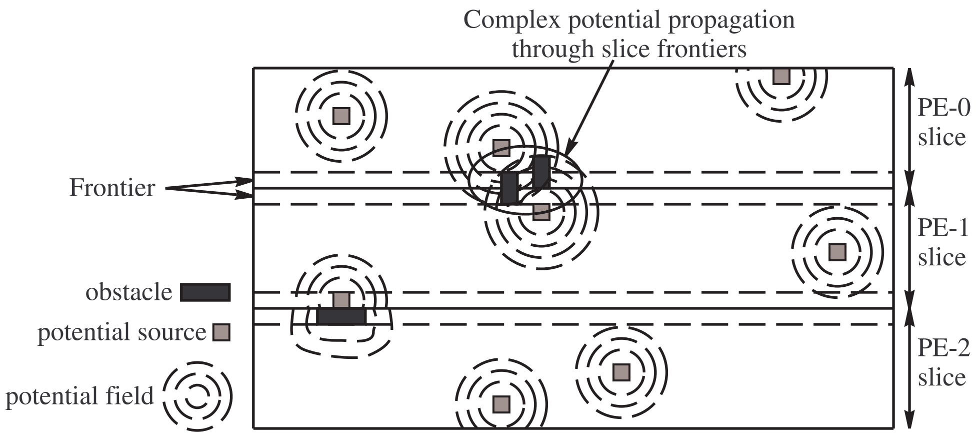


Figure 7: Potential fields parallel propagation based on environment partitioning.

cessors, and environment duplication on each processor. The last strategy leads each processor to compute the entire propagation field of a subset of resources, and finally all fields are combined into one final field, each processor computing a sub-part of this final field. This parallelization seems easy to implement, but the total amount of needed memory increase with the environment size and the number of processors: $\alpha \times N \times P$. We have considered excessive this amount of needed memory, and we have preferred the environment partitioning strategy. Each processor manages a slice of the environment grid, propagates potentials of the included resources until the slice frontiers, and then takes into account the potential of boxes on the other side of the frontiers and propagate again their potentials through its slice. We name this last step a repropagation. Processors make repropagations until no potential box is updated in the entire environment (many re-propagation steps can be necessary). Figure 7 illustrates this mechanism.

Several sequential algorithms have been tested, looking for an efficient run of sequential parts on each processor. Some recursive algorithms proceed the minimal number of boxes, but need recursive implementations with numerous function calls, and iterative algorithms make numerous loops processing all boxes of the environment (from up and left to bottom and right corner) but support efficient implementations taking advantage of cache memory (contiguous array accesses) and processor pipeline (possible loop unrolling). Breadth first recursive method has appeared to be the most efficient one when obstacles are numerous, and iterative method has been the most efficient one when potential sources are numerous. Moreover, most efficient sequential method can be different for initial propagation inside environment slice and for frontier repropagations.

We have experimented several combinations of sequential methods with environment partition-

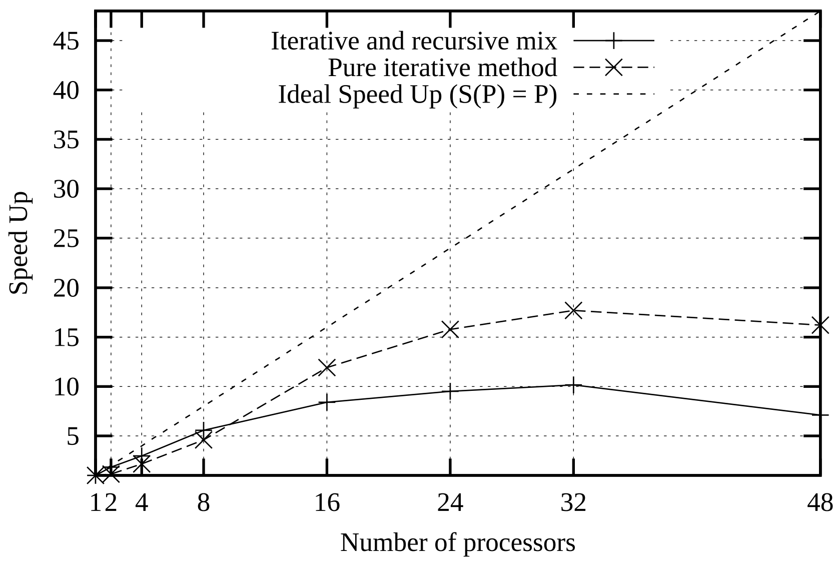


Figure 8: Speed up of parallel potential field propagation, based on environment partitioning, with mixed iterative and recursive sequential propagation inside environment slices, and with pure iterative propagation. Parallel run on a SGI-Origin2000, and a 1024×1024 environment with no obstacles and 1% of resources randomly spread.

ing parallelization strategy, on a basic environment of 1024×1024 boxes, without obstacles, and with 10000 resources (1% of the environment boxes) randomly spread (homogeneous environment). Compared to the same sequential execution time, we obtained better speed up on small number of processors with a mix of iterative method for initial propagation inside environment slices and recursive method for frontier repropagation. But we have obtained better speed up on larger number of processors with always iterative method, see figure 8. Finally, we have reached a speed up close to 18 on 32 processors for potential propagation routine on homogeneous large environments (efficiency close to 55%).

So, optimization of parallel potential propagation is not obvious and seems to need a heuristic to dynamically choose the best algorithm composition, function of number of processors, environment size, environment constitution, wave potential propagation radius, ...

6. Global performance and conclusion

In order to test the genericity of our programming model, we have successfully implemented a *Conway's game of life* with still agents, and an elementary simulator of *people doing shopping in a city center*. However, we present the performance of a larger simulator of *carrier robots* implemented in ParCeL-5/ParSSAP, used to study global population behavior emerging from individual ones, and typically needing big and intensive simulations.

These simulations run 10000 to 100000 robots (agents or cells) evolving in a 1024×1024 grid world (environnement), containing mines and factories (resources). Robots have to catch some ore (objects), to carry it to factories and then to drop it, and they repeat this activity until all the ore has been carried. Some randomly put obstacles disturb robot moves, and potential fields emitted by resources help robots to reach mines and factories. Potential emitted by a mine decreases when its ore stock decreases, and factory potentials remain constant, as it is decided they have no storage limit. So, regular potential field update is needed to take into account the mine stock decrease. Statistic information about the system evolution is regularly saved on disk, to allow a postmortem analyse of the efficiency of the robot behaviors.

ParCeL-5/ParSSAP programming model and its current implementation allows the user to easily model the robots and their world, by just implementing a few agent behaviors and specifying a few environment features. As such, user can run parallel simulations without parallel algorithmic nor parallel implementation nor simultaneity simulation efforts. The first goal of the model, to be adapted to the simulations of situated multi-agent systems, is therefore reached on this application.

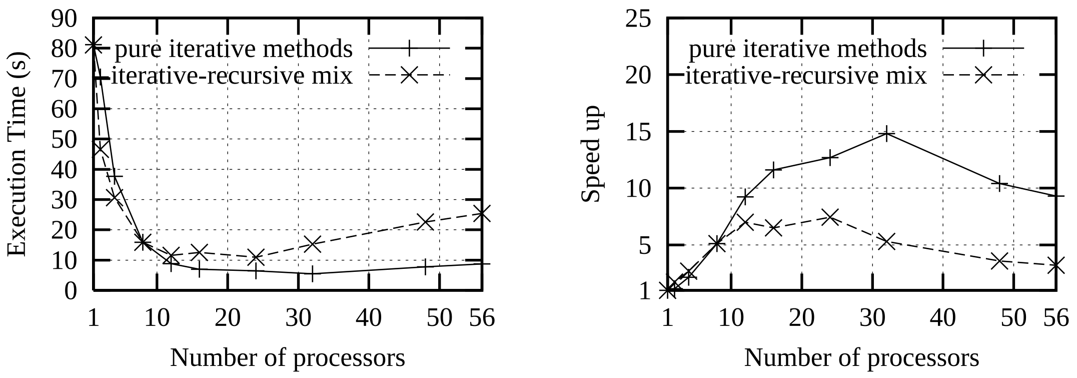


Figure 9: Execution time and Speed up of the *carrier robots* simulator, implemented in ParCeL-5/ParSSAP, and run on a SGI-Origin2000.

We have run our application on a SGI-Origin2000, with two different potential propagation methods, and we have obtained the execution time and speed up curves of the figure 9. Execution times are *Wall-Clock* times of the *entire* application runs (including disk IO). We have reached a speed up of 15 on 32 processors of a SGI-Origin2000 with an entirely iterative propagation method. On smaller parallel computers, such as a 4-processors SUN-Enterprise450 or on just 4 processors of the SGI-Origin2000, we have obtained the best speed up with a mixed iterative and recursive method, and we have reached speed ups of 2.7 (SGI-Origin2000) and 3.5 (SUN-Enterprise450).

So, these first results on an application with a homogeneous environment are satisfying, therefore the second goal of efficient parallel run is partially reached on this example.

But two problems remain. First, most efficient potential propagation algorithm depends on the number of processors used, and on the environment configuration (number of obstacles, number of resources, ...). We need to find and implement a heuristic that dynamically and automatically makes this choice. Second, performance has decreased when we have modelled more complex worlds, with some resources concentrations that lead to agent concentrations and to load unbalance among processors. In the future we plan to optimize load balancing and to design a heuristic to point out the most efficient potential propagation method, before adding new functionalities to our simulation model of situated multi-agent systems.

Acknowledgements

Authors thank the *Region Lorraine* (France), who has supported a part of this research, and the *Charles Hermite Center* (France) for accesses to its SGI-Origin2000.

References

[1] M. Bouzid, V. Chevrier, S. Vialle, and F. Charpillet. Parallel simulation of a stochastic agent/environment interaction. *Integrated Computer-Aided Engineering*, 8(3), 2001.

[2] J. Ferber. *Multi-Agent Systems: Towards a Collective Intelligence.* Addison-Wesley, 1998.

[3] L. Kale. Programming languages for CSE: The state of the art. *IEEE Computational Science & Engineering*, 5(2):18–26, Apr.-June 1998.

[4] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrel. *P-Threads Programming.* O'Reilly & Associates, Inc., first edition, September 1996.

[5] M. Tabme, J. Adibi, Y. Al-Onaizan, A. Erdem, G.A. Kaminka, S.C. Marsella, and I. Muslea. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110(2):215–239, June 1999.

[6] S. Vialle, M. Bouzid, V. Chevrier, and F. Charpillet. ParCeL-3: a parallel programming language based on concurrent cells and multiple clocks. *SNPD00*, 2000. Reims, France.

[7] S. Vialle, Y. Lallement, and T. Cornu. Design and implementation of a parallel cellular language for MIMD architectures. *Computer languages*, December 1998.