# A GPU Massivelly Parallel FIPA Architecture

Luiz Guilherme Oliveira dos Santos
MediaLab - Universidade Federal Fluminense

Esteban Walter Gonzales Clua
MediaLab - Universidade Federal Fluminense

Flávia Cristina Bernardini
labIDes - Universidade Federal Fluminense

## Abstract

The dynamic nature and common use of agents and agent paradigm motives the investigation on standardization of multi-agent systems (MAS). The main property of a MAS is to allow the sub-problems related to a constraint satisfaction issues to be subcontracted to different problem solving agents with their own interests and goals, being FIPA one of the most commonly collection of standards used nowadays. When dealing with a huge set of agents for real time applications, such as games and virtual reality solutions, it is hard to compute a massive crowd of agents due the computational restrictions in CPU. With the advent of parallel GPU architectures and the possibility to run general algorithms inside it, it became possible to model such massive applications. In this work we propose a novel standardization of agent applications based on FIPA using GPU architectures, making possible the modelling of more complex crowd behaviours.

**Keywords::** Crowd Simulation, Artificial Intelligence, Multi Agent Systems, GPU Computing

**Author's Contact:**

{lguilherme,esteban}@ic.uff.br
flaviabernardini@vm.uff.br

## 1 Introduction

A multi-agent system — MAS — has the interesting property to allow modeling subdivisions of the constraint satisfaction problem to individual and different agents specifications, with their own interests and goals. Furthermore, domains with multiple agents of any type, including autonomous vehicles [Franklin and Graesser 1996] and human-agents massively used in game development, are generally solved with this approach.

The main concept of a MAS is to simulate real world environments and interactions, composed by many entities, *e.g.* a building full of people during an emergency evacuation, a bee community, biological interactions between cells or enzymes, and so on. In applications such as games and simulation, the creation of many individuals with different behaviors and/or objectives became widespread. There are several approaches that explore this dynamic property [Reynolds 1987; Musse and Thalmann 2001; S. R. Musse ; Pelechano et al. 2007], but when dealing with video-games and interactive applications, there are many computational restrictions that must be carefully analysed, since their computation can be expensive. Most of previous work explore the hardware limitations to create bigger crowds [Reynolds 2006; van den Berg et al. 2008]. Others explore some of the problems related to the simulation itself just like collisions [Guy et al. 2009], Path-Planning [Yersin et al. 2008], many behaviors [Sung et al. 2004] and so on. The biggest problem is that a crowd simulation leads to a huge amount of computing data, and it is hard to make it real time. Researches like [Lamarche and Donikian 2004] explore a different hardware models to improve its results and create many agents as possible.

Since 2006, the use of graphics processing units paradigm (GPUs) became not only a new research area, but it is being used inside many applications and operational systems to escape from performances bottlenecks. When GPUs became cheaper and fully programmable, many researchers are exploring this power in order to create more agents with improved behaviours, according to its limitations [Bleiweiss 2008; Torchelsen et al. 2010]. However, mapping agent behaviors to GPU architectures is not trivial, given the GPU restrictions and the complexity of Artificial Intelligence algorithms. Many heuristics of this field try to avoid $O(n^2)$ complexity using different and complex structures and decision trees. Although these AI algorithms gives good results to a single number of agents, hardly them achieve a good scalability [Turner and Jennings 2001].

Unlike other researches, we believe that a standardization of this process of creating agents in GPU architectures is necessary not only to improve the actual implementations, but also to make easier to game developers. This work is a continuation of previous work [dos Santos et al. 2010; dos Santos et al. 2011] where it has been exposed a mapping process using a wide spread framework to program FIPA agents called JADE [Bellifemine et al. 2007] to GPU Computing.

Agents can be based on two different architectures: logic- and reactive-based [Flores-Mendez 1999]. The former is based on knowledge systems, in which the programmer has to represent the complete environment and create rules to manipulate the agent according to reasoning mechanisms. The latter is generally based in a decision-making behavior. Unlike the logic-based method, the reactive doesn't need a reasoning system, but only the modeling of a communication with the environment data, in order to receive some sort of information, and acting according to the observed data.

FIPA stands for *Foundation For Intelligent Physical Agents* [FIPA 2012] and it is a non-profit organization, which develops patterns to create applications using agent-based approaches. This organization, founded in 1996, is composed from both academic and industry members since its creation. These agent-based approaches are widely used by academic and industrial computing solutions.

## 2 The GPU FIPA Architecture

Since FIPA is an Agent Oriented programming pattern, the main actor we have in our architecture is the agent. A simple agent life cycle in this architecture is illustrated in Algorithm 1.

**Algoritm 1** Life Cycle of a Generic Agent

**Require:** $Evironment$: Agent's world.
**Require:** $State$: Agent's Initial State.
1: Load all initial variables;
2: **while** ( Agent's conditions to stop are not valid) **do**
3: Execute Agent's Behavior;
4: Interact with the $Environment$;
5: Change Agent's $State$;
6: **end while**

The main controller of the agents is called a Container. It is possible to have one or more containers in an application and different types of agents within this container. One or more containers can also share the same environment. Generally, there is a Main Container to control all the other containers.

In the model we follow the agent is not directly implemented, but it has many descriptions that help the controller of these agents to know how to stop them and maintain its autonomy. Figure 1 shows how the role of both container and agent description during the execution and the relationship between them.

Each agent description dictates how the agent must be executed and what are the conditions to make the agent stop. This has to be implemented on the functions `action` and `done` respectively. In a

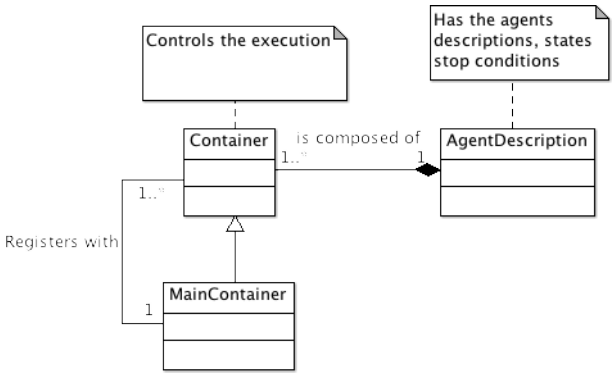


**Figure 1:** *Top Level Agent Oriented Programming Architecture*

CPU implementation these descriptions are inside the agents description, different from the GPU, that we need to put these in a separated file that goes there because of GPU compiler restrictions, as shown in Figure 2.

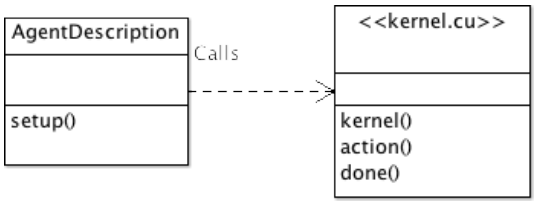


**Figure 2:** *Description of how a kernel is linked to the solution and called*

## 2.1 Kernel Structure

In a typical CPU approach the agent live along all the cycle, with the agent description for convenience. Another possible approach is to create a Behavior class that maps the agent's algorithm and improve the liability of the code. Since every agent has an autonomous execution and the GPU architecture follows a SIMD paradigm, it is necessary that each agent has the knowledge of his own code and data. Each agent is mapped to CUDA as threads, but not all agent's data will be processed inside the kernel, as shown in Figure 2, where method setup from AgentDescriptor class calls the kernel method in «kernel.cu». Methods action and done are called only inside the kernel («kernel.cu»).

Since the number of agents is variable, the kernel algorithm needs to be suitable to many different configurations. GPUs restrict the number of threads and blocks to be power of 2. For instance, if there are 1000 agents inside a simulation, there will be 1024 threads divided in blocks. In Algorithm 2 we see how this is done. If an agent has a Unique ID (UID) greater than the maximum number of agents, this agent will be idle. Since every agent is mapped into a thread, the UID can be easily achieved by a simple arithmetic using descriptors of the dimension of the block, ID of the block and ID of the thread.

**Algoritm 2** Kernel

**Require:** $World$: Agent's world.
**Require:** $State$: Initial Position of the Agent.
**Require:** $Numberof Agents$: Max number of agents used.
1: $UID = (BlockIndex \times BlockID) + ThreadID$;
2: **if** $(UID < Numberof Agents)$ **then**
3: **while** (Agent's not in final position) **do**
4: $State =$ NextState($State$, $World$);
5: **end while**
6: **end if**

To calculate the number of threads and blocks used, we use the following equations:

$$N_T = 2^{\left\lceil \frac{\log(N_A)}{\log 2} \right\rceil} \quad (1)$$

$$N_B = \frac{N_T}{N_W} \quad (2)$$

$$N_{TpB} = \frac{N_T}{N_B} \quad (3)$$

where:

$N_A$ is the number of agents in a simulation;

$N_T$ is the real number of threads that is going to be executed;

$N_B$ is the number of blocks created;

$N_W$ is the number of threads in a warp[1]. It is given by the GPU specification;

$N_{TpB}$ is the number of threads per block. The total amount of threads is given by $N_B \times N_TpB$.

These equations minimize the number of warps, improves the scalability and calculates the blocks and threads in power of 2. If the number of blocks or the number of threads per blocks is higher than the maximum of the GPU we relax the second equation and allows more warps per blocks.

## 2.2 Test Case — Pathfiding A*

The A* algorithm [Nilson 1971] is a search algorithm that uses a minimum cost heuristic and dynamic programming techniques. Different from other GPU implementations of this algorithm [Bleiweiss 2008; Walsh and Banerjee 2010], we used the traditional heuristic of A*, defined by Equation 4, where $g(n)$ evaluates the sum of costs from the beginning node to the node $n$, and $h(n)$ is the distance between the node $n$ and the objective node. We based our implementation on [Lester 2004]. Algorithm 2 shows how is the kernel implementation. Not that if on line 3 restricts the number of agents inside the kernel, as explained before.

$$f(n) = g(n) + h(n) \quad (4)$$

For simplicity, this test case has no obstacles and the agents are located into a two dimension map. The class diagram shown in Figure 3 shows the structure of an oriented object implementation of an agent description on the CPU. Figure 4 illustrates the class diagram of an agent description that uses a GPU to process part of its information. The kernel is placed in a separated file as a library, and executes the behavior of an agent. All the other settings are in myAgent class, that is triggered by the setup function. Note that Figure 4 shows how our GPU FIPA Architecture is used to model a MAS.

## 2.3 Performance Analysis

We want to verify the scalability of the application: the execution times maintain when we change the number of agents? We used an Intel Core i7 3.07GHz and 8GB DDR3 memory for the CPU, and a GeForce GTX 580 with 512 CUDA cores, 1544MHz for each core and 1536MB GDDR5 memory for the GPU tests. All the test case scenarios (Test Scn.) in Table 1 were performed 10 times in a CentOS 6 operational system. The map has a fixed 1000 × 1000 dimension[2]. Note that the columns $N_A$ (number of agents in a simulation), $N_B$ (number of blocks created) and $N_{TpB}$ (number of threads per block) are used for GPU configuration calculated by Equations 1, 2 and 3.

[1]Each thread is executed by a single core, and each block of threads in a Stream Multiprocessor(SM), which consist of array of cores. Warp is each subset of threads running in parallel in each block. The programmer does not have control of these warps swaps, being completely scheduled by the GPU itself.

[2]This size of map is considered a large one.

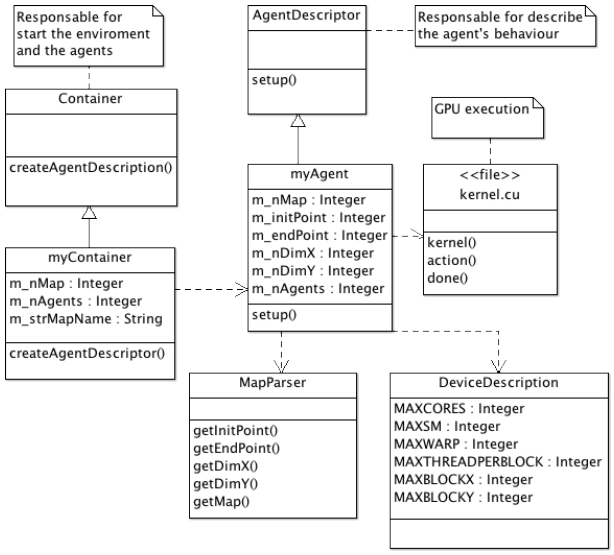


**Figure 3:** *Basic Class Diagram for the Agent Pathfiding on GPU*

| Test Number | $N_A$ | $N_B$ | $N_{TpB}$ |
|---|---|---|---|
| $T_0$ | $10^0$ | 1 | 1 |
| $T_1$ | $10^1$ | 1 | 16 |
| $T_2$ | $10^2$ | 4 | 32 |
| $T_3$ | $10^3$ | 32 | 32 |
| $T_4$ | $10^4$ | 512 | 32 |
| $T_5$ | $10^5$ | 1024 | 128 |
| $T_6$ | $10^6$ | 1024 | 1024 |

**Table 1:** *Test Scenarios performed*

Table 2 shows the mean and standard deviation of the 10 times measured on executions. We can observe that, as the number of agents grows ($T_1$ to $T_6$), the GPU maintains the scalability, loosing time only when the number of warps grows. However, these still are good times for a realtime simulation. On the other hand, the CPU times increase linearly as the number of agents grows. To a better perception of the time increase on GPU implementation, Figure 5 shows the evolution of the time execution when increasing the number of agents. It is possible to see that there are perceptual changes when more warps per blocks are required, specially from $10^5$ to $10^6$ agents.

| Test Number | CPU | | GPU | |
|---|---|---|---|---|
| | Mean Time(s) | Standard Deviation(s) | Mean Time(s) | Standard Deviation(s) |
| $T_0$ | 0,000027 | 0,000004 | 0,102781 | 0,001090 |
| $T_1$ | 0,000240 | 0,000005 | 0,103334 | 0,000914 |
| $T_2$ | 0,002377 | 0,000006 | 0,104765 | 0,003214 |
| $T_3$ | 0,024976 | 0,000911 | 0,105510 | 0,004830 |
| $T_4$ | 0,235692 | 0,000077 | 0,107930 | 0,008282 |
| $T_5$ | 2,360808 | 0,003434 | 0,108138 | 0,005976 |
| $T_6$ | 23,562007 | 0,000502 | 0,119523 | 0,001940 |

**Table 2:** *Algorithms performance on both CPU and GPU*

## 3 Conclusion

In the future, this works aims to determinate how to use GPU computing inside AOP paradigm. With the evolution of this work, we intend to create an abstraction layer to turn possible to create agents directly in GPU. This aims to facilitate the developer to not have to learn GPU's architecture, increasing the productivity process in applications that require massive use of agents.

We believe that along with the development of the GPU Computing, the restrictions in creating agents that we've shown will sightly decrease during the time. However, its scalability and massiveness nature will be maintained. For future work we intend to evolve the standardization of this architecture solving some of the restrictions we've found, such as communications and execution of heterogeneous agents in a more complex environment. We also intend to accept in our architecture more behaviors to agents, with the possibility to explore kernel capabilities of the GPU. This functionality

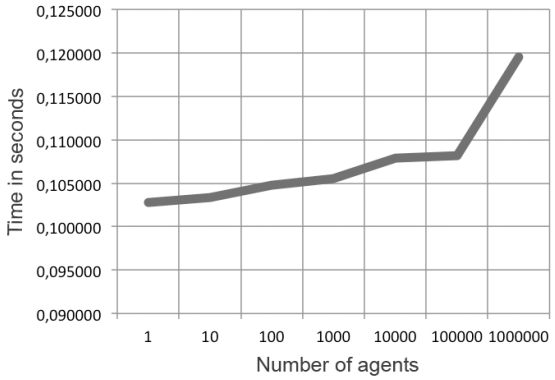


**Figure 4:** *Test Scenarios Results Chart*

will allow to different behaviors be treated in parallel.

## References


BELLIFEMINE, F., CAIRE, G., AND GREENWOOD, D. 2007. *Developing multi-agent systems with JADE*. Wiley Series in Agent Technology.

BLEIWEISS, A. 2008. Gpu accelerated pathfinding. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, NVidia, 65–74.

DOS SANTOS, L. G. O., BERNARDINI, F. C., CLUA, E. G., DA COSTA, L. C., AND PASSOS, E. 2010. Mapping multi-agent systems based on fipa specification to gpu architectures. In $3^a$ *Conferencia Anual em Ciencia e Arte dos Videojogos*, 109–118.

DOS SANTOS, L. G. O., BERNARDINI, F. C., CLUA, E. G., DA COSTA, L. C., AND PASSOS, E. 2011. Mapping a pathfiding multiagent system based on fipa specification to gpu architectures. In *X Simposio Brasileiro de Games e Entretenimento Digital*.

FIPA, 2012. Foundation for intelligent physical agents. `http://fipa.org/`.

FLORES-MENDEZ, R. 1999. Towards a standardization of multi-agent system frameworks. *ACM Crossroads Magazine*. `http://www.acm.org/crossroads/xrds5-4/multiagent.html`.

FRANKLIN, S., AND GRAESSER, A. 1996. Is it an agent or just a program? a taxonomy for autonomous agents. In *Intelligent Agents III. Agent Theories, Architectures, and Languages. LNAI*, J. Muller, M. Wooldridge, and N. Jennings, Eds., vol. 1193, 21–35.

GUY, S. J., CHHUGANI, J., KIM, C., SATISH, N., LIN, M., MANOCHA, D., AND DUBEY, P. 2009. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, SCA '09, 177–187.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23*, 509–518.

LESTER, P., 2004. A* for beginners. `http://www.policyalmanac.org/games/aStarTutorial.htm`.

MUSSE, S. R., AND THALMANN, D. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics 7*, 152–164.

NILSON, N. J. 1971. *Problem-solving methods in Artificial Intelligence*. McGraw-Hill.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2007. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '07, 99–108.

REYNOLDS, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. In *ACM SIGGRAPH '87 Conference Proceedings*, vol. 21, 25–34.

REYNOLDS, C. 2006. Big fast crowds on ps3. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM, Sandbox '06, 113–121.

S. R. MUSSE, D. T. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Workshop Computer Animation and Simulation of Eurographics*, 39–52.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. Eurographics '04.

TORCHELSEN, R. P., SCHEIDEGGER, L. F., OLIVEIRA, G. N., BASTOS, R., AND COMBA, J. L. D. 2010. Real-time multi-agent path planning on arbitrary surfaces. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, I3D '10, 47–54.

TURNER, P., AND JENNINGS, N. 2001. Improving the scalability of multi-agent systems. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, vol. 1887 of *Lecture Notes in Computer Science*. Springer Berlin, 246–262.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, I3D '08, 139–147.

WALSH, K., AND BANERJEE, B. 2010. Fast A$^*$ with iterative resolution for navigation. *International Journal on Artificial Intelligence Tools 19* (February), 101–119.

YERSIN, B., J, M., MORINI, F., AND THALMANN, D. 2008. Real-time crowd motion planning: Scalable avoidance and group behavior. *Vis. Comput. 24*, 10 (Sept.), 859–870.