A Parallel Fipa Architecture Based on GPU for Games and Real Time Simulations

Luiz Guilherme Oliveira dos Santos[1], Esteban Walter Gonzales Clua[1],
and Flávia Cristina Bernardini[2]

[1] Universidade Federal Fluminense — UFF
Instituto de Computação — IC
MediaLab
Niterói - RJ, Brasil
[2] Universidade Federal Fluminense — UFF
Instituto de Ciência e Tecnologia — ICT
LabIDeS — Laboratório de Inovação no Desenvolvimento de Sistemas
Rio das Ostras - RJ, Brasil

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 subproblems 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. The obtained results in our simulations were very promising and show that GPUs may be a choice for massively agents applications. We also present restrictions and cases where GPU based agents may not be a good choice.

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 [6] and human-agents massively used in game development, are generally solved with this approach.

The dynamic nature of agent distribution motivates research by groups working on the standardization of dynamic collaborative MAS. Mendez [5] describes each model of MAS proposed by these groups, and concludes that "The architecture models open environments composed of logically distributed areas where

M. Herrlich, R. Malaka, and M. Masuch (Eds.): ICEC 2012, LNCS 7522, pp. 306–317, 2012.

agents exist. The basic agents in this architecture are minimal agents, local area coordinators, yellow page servers, and cooperation domain servers". One of these models, used in this work, is Foundation for Intelligent Physical Agents (FIPA).

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 [17,12,18,15], 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 [16,2]. Others explore some of the problems related to the simulation itself just like collisions [7], Path-Planning [29], many behaviors [22] 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 [10] 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 [3,24]. 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 they achieve a good scalability [25].

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 [19,20] where it has been exposed a mapping process using a wide spread framework to program FIPA agents called JADE [1] to GPU Computing.

The purpose of this work is to present a novel and efficient architecture to autonomous agents using GPU Computing paradigm. We describe how a MAS is usually implemented following the FIPA model, and we develop a new paradigm to implement a MAS in GPU Computing based architectures. We also present a case study where we show how to map a typical MAS problem to GPU Computing based architecture.

This paper is organized as follows: Section 2 presents FIPA patterns and gives an overall about its usage and implications. Section 3 shows the architecture we are proposing, and gives a study case. Section 4 analyses the implementation and performance results. Finally, Section 5 concludes this work and describes future work.

2 FIPA

Agents can be based on two different architectures: logic- and reactive-based [5]. 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* [4] 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. Other patters such as MASIF(Mobile Agent System Interoperability Facility)[9], are used to specific applications, and are not so generic as FIPA.

During the evolution of FIPA, two main concepts has being developed: the FIPA-ACL (for communications purposes) and the Agent Management Framework. FIPA-ACL is the communication standard among agents based on the 90's internet patterns such as OMG, DCE W3C and GGF. The Agent Management Framework focus on how to create, operate and manipulate the agents. It defines the creation, registration, location, communication and operation process of the agents. In this paper, our efforts lies on the management of these agents.

FIPA stands that there is an abstract layer where all the agents have services provided to them and the programmer could develop application on the top layer of this architecture. On the other hand, all the agents are autonomous, they act like a peer-to-peer application and there must be a top application that controls their execution called Container. The container has an Agent Description of the agents inside it and has the authority to start the agents and control the agent's environment. This paradigm is called *Agent Oriented Programming* — AOP. Figure 1 shows at a top level the architecture of a system implemented according to AOP approach. There are many frameworks, platforms and applications based on AOP approach, such as JADE [1], FLUX [23] and JACK [28]. Nowadays, its use is as variable as possible, for instance: "robots" that search pieces of information in websites, media-oriented services[8] , evacuation and massive people simulations [26], and so on.

3 A GPU Based Architecture for Massive FIPA Based Agents

In this work we propose a novel FIPA based architecture using AOP paradigm for a massively concurrent agents application. As discussed in introduction, mapping an agent-based architecture to a SIMD paradigm, on which GPU is based, is not trivial and new structures must be proposed and developed. To test how works our general architecture, we implemented a simple agent system that behaviors based on a traditional A* path-finding algorithm, described in Subsection 3.3. Our objective in our experiments is analyze how scalable are the problems implemented using AOP paradigm on GPU architecture. So, due to lack of performance on GPU when processes execution demands high volume of communication, we avoided data communication among the agents. Results and tests using our proposed architecture to implement our MAS with the chosen agent behavior are presented and discussed in Section 4.

3.1 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.
Load all initial variables;
while ( Agent's conditions to stop are not valid) do
    Execute Agent's Behavior;
    Interact with the Environment;
    Change Agent's State;
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 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.

3.2 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

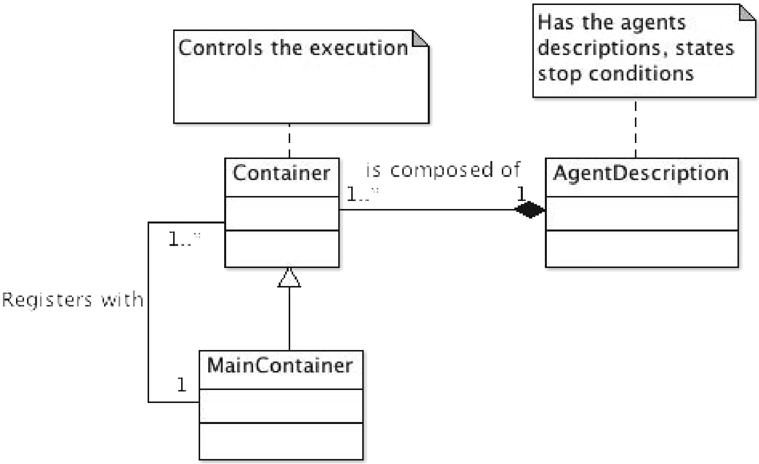

Fig. 1. Top Level Agent Oriented Programming Architecture

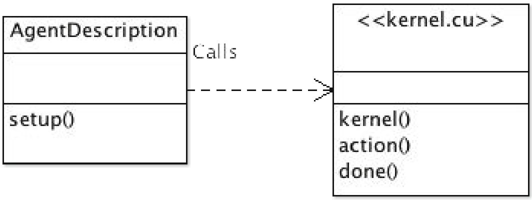

Fig. 2. Description of how a kernel is linked to the solution and called

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.

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

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

Algoritm 2. Kernel

```
Require: $World$: Agent's world.
Require: $State$: Initial Position of the Agent.
Require: $NumberofAgents$: Max number of agents used.
$UID = (BlockIndex \times BlockID) + ThreadID$;
if ($UID < NumberofAgents$) then
    while (Agent's not in final position) do
        $State =$ NextState($State$, $World$);
    end while
end if
```

$$N_B = \frac{N_T}{N_W} \tag{2}$$

$$N_{TpB} = \frac{N_T}{N_B} \tag{3}$$

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.

Using an heterogeneous programming paradigm, the Agent Descriptor has to configure and call a kernel one or more times, depending on the solution provided. When GPU is used, the Descriptor also makes the memory copies from CPU to GPU and vice versa. It has also to make the kernel configuration and the GPU deallocation. The agent behavior will be incorporated by the kernel itself and will be specific for each kind of actions. In our platform we develop a pathfinding agent, typically found in many crowd behavior games.

4 Test Case — Pathfinding Agents

The A* algorithm [13] is a search algorithm that uses a minimum cost heuristic and dynamic programming techniques. Different from other GPU implementations of this algorithm [3,27], we used the traditional heuristic of A*, defined

[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.

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 [11]. Algorithm 2 shows how is the kernel implementation. Not that if on line 3 restricts the number of agents inside the kernel, as explained in Section 3.2.

$$f(n) = g(n) + h(n) \tag{4}$$

A*'s Complexity: In best case, A*'s complexity is $O(N)$, where the algorithm finds the path directly to the objective node, where N is the number of nodes between the base node and the objective node. On worst case the algorithm is $O(b^N)$, where b is a partition factor. The complexity of A* depends directly of the heuristic function used, tending to be exponential if the function is too precise [21]. The average complexity is $O(N \text{ exp } (C\phi(N)))$, where $\phi(N) = log(N)^k$ [14].

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,

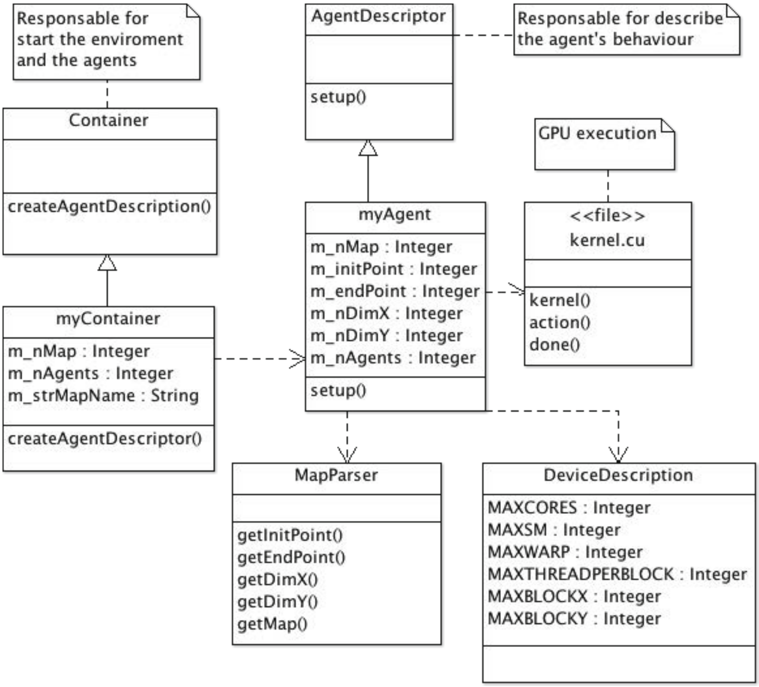

Fig. 3. Basic Class Diagram using the CPU

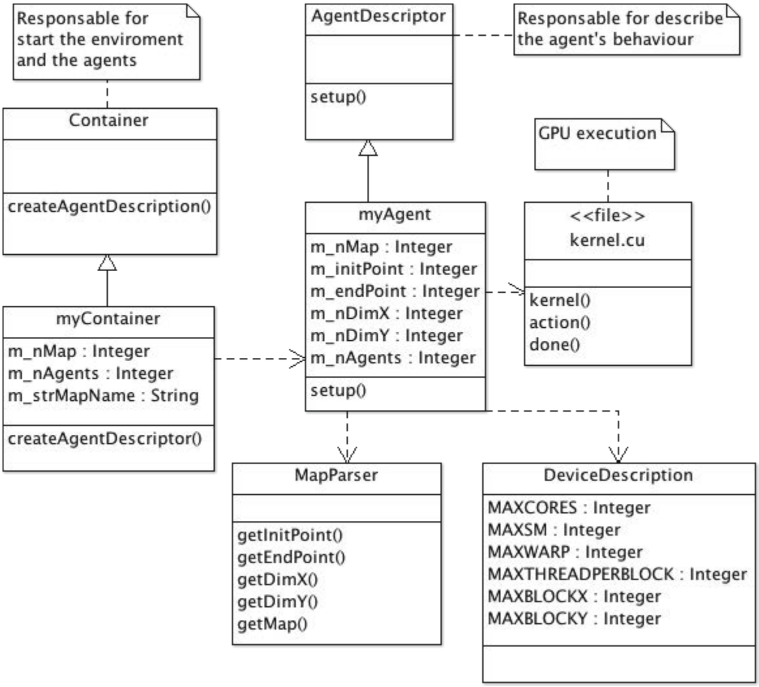

Fig. 4. Basic Class Diagram for the Agent Pathfiding on GPU

that is triggered by the setup function. Note that Figure 4 shows how our GPU FIPA Architecture is used to model a MAS.

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.

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

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

Table 1. Test Scenarios performed

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 2. Algorithms performance on both CPU and GPU

Test	CPU		GPU	
Number	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

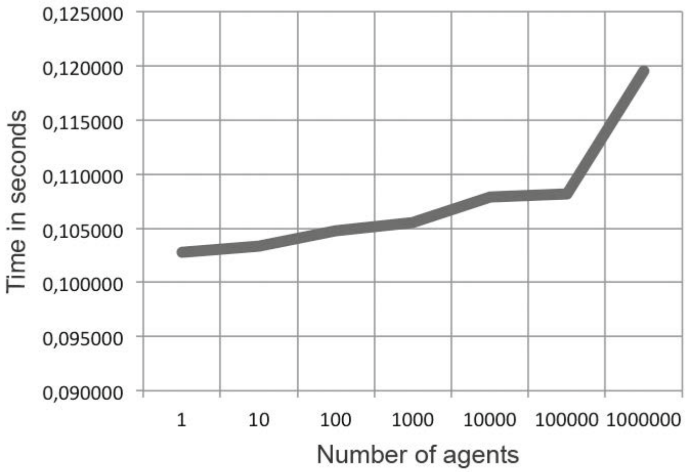

Fig. 5. Test Scenarios Results Chart

possible to see that there are perceptual changes when more warps per blocks are required, specially from 10^5 to 10^6 agents.

Restrictions of the Architecture: The proposed architecture follows some of the agent management patterns defined by FIPA and can be easily reproduced and adapted by some modifications. Though, there is still many restrictions implemented in our architecture regarding to implement heterogeneous agents with heterogeneous behaviors and agent communications. These restrictions exist due to the SIMD paradigm of the GPU, where is not recommended to create different branches inside one kernel just to solve these problems. One good solution could be multiple kernel calls, but nowadays we do not have control of how much of the GPU capacity will be allocated to each concurrent kernel. So it is hard to distribute the kernel calls correctly and smoothly. For this reason, our solution is focused for programmers that want to create FIPA agents with few communication and few heterogeneity among them. Another good point of it is to reproduce innumerable problems with the same pattern, only changing a few lines, increasing the productivity and the liability of the code, which is a big issue when dealing with game engines.

5 Conclusion and Future Work

Agent oriented paradigm has been largely used in game development, but almost all approaches implement them in a CPU architecture. While CPU's approaches allows generic and complex agent behavior solutions, it is shown that a huge amount of agents may be impracticable for interative frame rates. In this paper we present a novel and efficient multi-agent architecture for a GPU programming paradigm, allowing up to two orders of magnitudes of agents in interative frame rates.

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 will allow to different behaviors be treated in parallel.

References

1. Bellifemine, F., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. Wiley Series in Agent Technology (2007)
2. van den Berg, J., Patil, S., Sewall, J., Manocha, D., Lin, M.: Interactive navigation of multiple agents in crowded environments. In: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, I3D 2008, pp. 139–147. ACM (2008), http://doi.acm.org/10.1145/1342250.1342272
3. Bleiweiss, A.: Gpu accelerated pathfinding. In: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, pp. 65–74. NVidia (2008)
4. FIPA: Foundation for intelligent physical agents (2012), http://fipa.org/
5. Flores-Mendez, R.: Towards a standardization of multi-agent system frameworks. ACM Crossroads Magazine (1999), http://www.acm.org/crossroads/xrds5-4/multiagent.html
6. Franklin, S., Graesser, A.: Is it an Agent or Just a Program? a Taxonomy for Autonomous Agents. In: Jennings, N.R., Wooldridge, M.J., Müller, J.P. (eds.) ECAI-WS 1996 and ATAL 1996. LNCS (LNAI), vol. 1193, pp. 21–35. Springer, Heidelberg (1997)
7. Guy, S.J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., Dubey, P.: Clearpath: highly parallel collision avoidance for multi-agent simulation. In: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2009, pp. 177–187. ACM (2009), http://doi.acm.org/10.1145/1599470.1599494
8. Han, S.W., Kim, J.: Preparing experiments with media-oriented service composition for future internet. In: Proceedings of the 5th International Conference on Future Internet Technologies, CFI 2010, pp. 73–78. ACM, New York (2010), http://doi.acm.org/10.1145/1853079.1853099
9. Islam, N., Mallah, G.A., Shaikh, Z.A.: Fipa and masif standards: a comparative study and strategies for integration. In: Proceedings of the 2010 National Software Engineering Conference, NSEC 2010, pp. 7:1–7:6. ACM, New York (2010), http://doi.acm.org/10.1145/1890810.1890817
10. Lamarche, F., Donikian, S.: Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. Computer Graphics Forum 23, 509–518 (2004)
11. Lester, P.: A* for beginners (2004), http://www.policyalmanac.org/games/aStarTutorial.htm
12. Musse, S.R., Thalmann, D.: Hierarchical model for real time simulation of virtual human crowds. IEEE Transactions on Visualization and Computer Graphics 7, 152–164 (2001)
13. Nilson, N.J.: Problem-solving methods in Artificial Intelligence. McGraw-Hill (1971)
14. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley (1984)
15. Pelechano, N., Allbeck, J.M., Badler, N.I.: Controlling individual agents in high-density crowd simulation. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2007, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association (2007), http://dl.acm.org/citation.cfm?id=1272690.1272705

16. Reynolds, C.: Big fast crowds on ps3. In: Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames, Sandbox 2006, pp. 113–121. ACM (2006), http://doi.acm.org/10.1145/1183316.1183333
17. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. In: ACM SIGGRAPH 1987 Conference Proceedings, vol. 21, pp. 25–34 (1987)
18. Musse, S.R., Thalmann, D.: A model of human crowd behavior: Group interrelationship and collision detection analysis. In: Workshop Computer Animation and Simulation of Eurographics, pp. 39–52 (1997)
19. dos Santos, L.G.O., Bernardini, F.C., Clua, E.G., da Costa, L.C., Passos, E.: Mapping multi-agent systems based on fipa specification to gpu architectures. In: 3^a Conferencia Anual em Ciencia e Arte dos Videojogos, pp. 109–118. Instituto Superior Tecnico, Taguspark (2010)
20. dos Santos, L.G.O., Bernardini, F.C., Clua, E.G., da Costa, L.C., Passos, E.: Mapping a path-fiding multiagent system based on fipa specification to gpu architectures. In: X Simposio Brasileiro de Games e Entretenimento Digital (2011)
21. Stefik, M.: Introducing to Knowledge Systems. Morgan Kaufmann (1995)
22. Sung, M., Gleicher, M., Chenney, S.: Scalable behaviors for crowd simulation. In: Eurographics 2004 (2004)
23. Thielscher, M.: Flux: A logic programming method for reasoning agents. Theory and Practice of Logic Programming 5, 533–565 (2005)
24. Torchelsen, R.P., Scheidegger, L.F., Oliveira, G.N., Bastos, R., Comba, J.L.D.: Real-time multi-agent path planning on arbitrary surfaces. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D 2010, pp. 47–54. ACM (2010), http://doi.acm.org/10.1145/1730804.1730813
25. Turner, P.J., Jennings, N.R.: Improving the Scalability of Multi-agent Systems. In: Wagner, T.A., Rana, O.F. (eds.) AA-WS 2000. LNCS (LNAI), vol. 1887, pp. 246–262. Springer, Heidelberg (2001)
26. Valckenaers, P., Sauter, J., Sierra, C., Rodriguez-Aguilar, J.A.: Applications and environments for Multi-Agent Systems. Autonomous Agents and Multi-Agent Systems 14(1), 61–85 (2006)
27. Walsh, K., Banerjee, B.: Fast A* with iterative resolution for navigation. International Journal on Artificial Intelligence Tools 19, 101–119 (2010)
28. Winikoff, M.: Jack intelligent agents: An industrial strength platform. In: Multi-Agent Programming, pp. 175–193. Kluwer (2005)
29. Yersin, B., Morini, F., Thalmann, D.: Real-time crowd motion planning: Scalable avoidance and group behavior. Vis. Comput. 24(10), 859–870 (2008), http://dx.doi.org/10.1007/s00371-008-0286-0