État de l'art des simulations multi-agents sur GPU

Emmanuel Hermellin, Fabien Michel, Jacques Ferber

État de l'art des simulations multi-agents sur GPU

Emmanuel Hermellin
emmanuel.hermellin@lirmm.fr

Fabien Michel
fmichel@lirmm.fr

Jacques Ferber
ferber@lirmm.fr

LIRMM - Laboratoire Informatique Robotique et Micro électronique de Montpellier
Université Montpellier 2 - CNRS - 161 Rue Ada, 34090 Montpellier

Résumé

De nombreux outils et plates-formes permettent le développement de simulations multi-agents sur CPU. Cependant, les temps de calculs et les besoins en ressources deviennent de plus en plus importants au vu du nombre d'entités simulés. Une solution est de se tourner vers le calcul sur carte graphique : le GPGPU. Apportant un rapport performance / prix imbattable, cette technique souffre cependant d'une programmation très spécifique qui limite son adoption par la communauté. Nous nous proposons donc de faire un état des lieux des simulations multi-agents sur GPU.

Mots Clef

Simulation multi-agents, GPGPU, CUDA, OpenCL.

Abstract

There are many tools for doing multi-agent based simulations (MABS) on CPU. However, computation time and resource increases so much with the number of entities that the CPU cannot support this inflation and an appealing solution is to use GPGPU. Inexpensive and powerful, this technology has one big issue : It's programming difficulty which limits the accessibility and the reusability. In this paper, we make a survey on multi-agents simulations on GPU.

Keywords

Multi-Agent Systems, GPGPU, CUDA, OpenCL.

1 Introduction

Les systèmes multi-agents (SMA) sont utilisés dans de nombreux domaines pour modéliser des systèmes très variés, pourvus que ces derniers puissent être représentés par un ensemble d'entités autonomes en interaction appelées agents. Dans certains cas, la complexité associée aux comportements des agents, ou plus simplement leur nombre, nécessite une très importante puissance de calcul. De fait, la puissance des processeurs classiques représente souvent un verrou majeur qui limite fortement le cadre dans lequel un modèle peut être conçu et expérimenté.
Avec la démocratisation du *Calcul Haute Performance* (HPC) et du *GPGPU*, des travaux ont été réalisés pour simuler des SMA sur GPU. Ces contributions sont nombreuses et démontrent les possibilités de gains importants offertes par le GPGPU (jusqu'à 40 fois plus rapide que sur CPU avec des plate-formes spécialisées comme *Repast* ou *Netlogo* [1]). Mais malgré un rapport performance prix imbattable, l'architecture spécifique d'un GPU rend le portage des SMA complexe et difficile et cette approche peine à se généraliser.

Dans cet article, nous présentons tout d'abord un état de l'art des simulations multi-agents sur GPU. La section 2 introduit le GPGPU et certaines notions qui lui sont liées. La section 3 présente les premiers travaux liés à l'utilisation du GPGPU dans les SMA et utilisant les fonctions graphiques des GPUs. La section 4 se focalise sur les contributions utilisant les interfaces de programmation spécialisées pour GPU puis la section 5 présente l'apparition de systèmes hybrides utilisant à la fois le CPU et le GPU. Enfin, la section 6 comporte l'énoncé des critères et l'analyse de l'existant.

2 Présentation du GPGPU

De par son design et son architecture, le CPU permet d'effectuer du calcul généraliste. Au contraire, les cartes graphiques ont été développées dans le but de soulager les processeurs de la charge que demandait la gestion des pixels et de l'affichage sur l'écran. Mais en évoluant, ces cartes sont devenus de véritables centres de calculs bien plus performants que les CPU (voir figure 1). Le *GPGPU* (General-Purpose computing on Graphics Processing Units) désigne donc le fait d'utiliser l'architecture massivement parallèle des GPU pour effectuer du calcul générique.

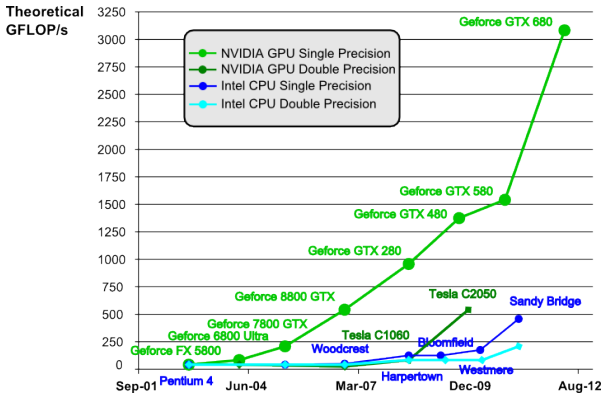

FIGURE 1 – Performance entre CPU et GPU

C'est en 1999 que le GPGPU a commencé à être utilisé. Sans véritable interface de programmation dédiée, il était nécessaire d'utiliser les fonctions graphiques du GPU. Ainsi, il fallait utiliser les *textures* comme structure de données. Chaque *texel* (élément le plus petit constituant une texture) pouvait accueillir trois valeurs à la place des composantes couleurs classiques. Ces données étaient manipulées par des *shaders*, des scripts s'exécutant sur le GPU.
Très peu accessible et extrêmement difficile, ces techniques étaient réservées à un public averti. En 2007, dans l'optique de simplifier ce processus, Nvidia mit à disposition *CUDA* [1], véritable environnement logiciel permettant d'utiliser le GPU à l'aide d'une extension du langage C mais limité aux cartes de la marque. Il fut suivi fin 2008 par un standard ouvert propulsé par Khronos Group prenant en charge toutes les cartes sans restriction de marques, *OpenCL* [2]. La philosophie derrière ces deux interfaces est la même : faciliter l'utilisation des nombreux cœurs présents sur le GPU. Le CPU, appelé *host*, va jouer le rôle de chef d'orchestre et va gérer les données, leurs copies sur le GPU et l'appel de procédures nommées *kernels*, destinées au GPU. Le GPU, le *device*, a pour mission d'exécuter ces *kernels* en utilisant l'architecture parallèle divisée en *blocs*, eux mêmes divisés en *threads* (fils d'instructions).
CUDA et OpenCL sont actuellement les deux solutions majeures permettant de mettre en œuvre du GPGPU.

3 Premiers SMA sur GPU

SugarScape est le premier SMA qui a profité d'un portage sur carte graphique grâce au GPGPU [2]. C'est un modèle multi-agents très simple dans lequel des agents réactifs évoluent dans un environnement discrétisé en cellules contenant une ressource, le sucre, et suivent des règles comportementales basiques (déplacement, reproduction, etc.). Entièrement exécutée sur le GPU, il était possible d'avoir un rendu en temps réel de la simulation avec plusieurs millions d'individus, ce qui reste encore aujourd'hui quasi impossible avec les CPU actuels.
Les très bonnes performances obtenues dans ce travail ont poussé ses auteurs à généraliser leur approche. Dans [3], ils présentent un *framework* mettant à la disposition des utilisateurs des méthodes facilitant la conversion de leurs modèles de simulation sur GPU (fonctions de gestions de données, d'interactions agents, de naissance, de mort, etc.). Ce travail a ainsi permis de lever des contraintes liées à la programmation sur GPU afin de la rendre plus accessible.
Suivant cette tendance, le framework *ABGPU* se proposait d'être une interface de programmation un peu plus générique et surtout plus accessible grâce à une utilisation transparente du GPU reposant sur une extension du langage C/C++ basés sur un système de mots clés [4]. À cela s'ajoutaient des fonctions et des classes pré-programmées, facilitant d'avantage l'implémentation de comportements et fonctions agents complexes. *ABGPU* était capable de simuler et d'afficher un banc de poissons composé de 65 536 agents en temps réel et en 3D
Même dans le cas d'architectures agents très simples, ces travaux mettent en évidence la difficulté de mettre en place une méthode de conversion générique pour le portage des SMA sur GPU, du fait de la grande hétérogénéité des modèles et des techniques de programmations archaïques. [1] a étudié les difficultés associées à de telles conversions en réalisant l'implémentation de différents modèles sur CPU (avec *NetLogo* et *Repast*), puis sur GPU. Ces cas d'études montrent bien que les spécificités de la programmation sur GPU rendent difficile, voire impossible, le processus de conversion qui est bien plus complexe qu'un simple changement de langage de programmation. Ainsi le GPGPU possède sa propre philosophie et requière un savoir particulier qui font de lui un paradigme à part entière.

4 L'essor du GPGPU

L'apparition de CUDA et OpenCL simplifie l'utilisation du GPGPU qui devient une solution incontournable dans les domaines où le temps de calcul est critique.
Flame GPU [3] est un exemple de ce qu'a permis l'émergence des outils spécialisés. En effet, il est une solution clef en main pour la création de simulations multi-agents sur GPU qui possède plusieurs avantages. Tout d'abord, *Flame GPU* se focalise sur l'accessibilité en s'appuyant sur une utilisation transparente du GPGPU. Pour cela, il utilise les *X-machines* [5] : un formalisme de représentation d'agents s'appuyant sur une extension du langage XML : le XMML. Ainsi, les états des agents sont implémentés dans des fichiers XMML pendant que leurs comportements sont programmés en C. Le code GPU est quant à lui généré grâce aux fichiers XMML combinés dans des templates GPUXMML puis vérifiés par des processeurs XSLT.
De plus, *Flame GPU* fournit un framework open source contenant des modèles agents pré-programmés facilement réutilisables. Ainsi, *Flame GPU* peut être utilisé pour simuler un large éventail de modèles dans des contextes différents : en biologie, e.g. simulation de cellules de peau [6, 7], en intelligence artificielle, e.g. simulation proie - prédateur [8], ou pour des simulations de foules e.g. [9].
Malgré les différents frameworks proposés jusqu'à lors de très nombreuses contributions repartent de zéro pour la création de SMA sur GPU avec comme seul objectif la performance. C'est le cas des travaux sur les simulations de *flocking* qui se focalisent sur la possibilité de simuler un nombre d'agents toujours plus important. Citons ainsi les travaux de *Passos et al.* [10] (premier modèle de *flocking* sur architecture parallèle) et ceux de *Li* [11] (intégration d'évitement d'obstacle dans le modèle). Les travaux d'*Erra* [12] proposent une description complète et détaillée des étapes suivies pour implémenter un modèle sur GPU. Ce travail offre une vision globale de la faisabilité et des performances que l'on peut obtenir en utilisant le GPGPU pour des modèles de flocking. Dans ces

1. https://developer.nvidia.com/what-cuda
2. http://www.khronos.org/opencl
3. http://www.flamegpu.com/

simulations, des millions d'individus sont rendus en temps réel et en 3D. *Silva* [13] s'inspire toujours du modèle de *flocking* originel et intègre une méthodologie appelée *self-occlusion* : les individus vont être plus ou moins visibles selon la distance à laquelle ils se trouvent de l'agent sur lequel on se focalise. [13] propose en outre une comparaison entre deux implémentations de son modèle (fonctions graphiques et CUDA). Les résultats de cette comparaison montrent que selon le type de calcul effectué, l'interface CUDA est moins performante qu'une implémentation utilisant des *shaders*. Enfin, *Husselmann* et *Hawick* [14] proposent le premier modèle autorisant une hétérogénéité des agents. La possibilité d'intégrer dans une même simulation des espèces différentes, ayant des réactions et buts différents, va permettre l'émergence de nombreux comportements nouveaux. Les performances sont excellentes avec un temps moyen de 0.1 secondes par itération pour plus de 37 000 agents simulés.

Les modèles de simulations de *foules* et de *trafics* font partie des autres domaines où le GPGPU a rapidement apporté une réelle plus-value. Le GPGPU permet en effet la simulation d'environnements plus grands qui permettent de mieux comprendre le déplacement des individus dans des quartiers ou sur des réseaux routiers complexes. Ce gain en performance est bien visible dans les simulations de trafic publiées dans [15] et [16]. L'outil de simulation de foule *Pedestrian framework* [17] propose plus qu'une simple recherche de performance. Évolution d'*ABGPU* basée sur *CUDA*, ce framework offre la possibilité de modéliser des agents plus cognitifs et permet par exemple d'implémenter les forces sociales d'*Helbing* [18]. C'est aussi la première contribution à effectuer une séparation explicite entre agent et environnement, ceci afin de modéliser et gérer des forces environnementales (attraction des vitrines, événement spécial, etc.). Avec ce modèle, il est possible de simuler 65536 agents en temps réel et en 3D.

Jusqu'ici une grande majorité des contributions présentaient des solutions ponctuelles et associées à des contextes particuliers. Afin de moins limiter la portée des outils développés, des recherches misent alors sur la généralisation de fonctions utilisables par tous. C'est le cas des algorithmes pour la navigation autonome et la planification d'itinéraires pour des milliers d'agents évoluant dans un environnement. En effet ces algorithmes se prêtent bien à un portage sur GPU et l'accélération obtenue est très importante. Ces calculs peuvent devenir très complexes dans les SMA où chaque agent devient un obstacle pour les autres. *Bleiweiss* [19] est le premier à porter des algorithmes de *Pathfinding* populaires (*A** et *Dijkstra*) sur GPU. D'autres travaux apportant l'aspect temps réel suivront [20].

Parallèlement, le développement des jeux vidéos a permis l'émergence de nouveaux algorithmes mieux adaptés aux SMA car ils introduisent la notion de localité. Ils permettent ainsi d'obtenir des comportements plus réalistes. L'algorithme *BVP Planner* [21] va utiliser une carte globale de l'environnement et des cartes locales générées par les agents. Ces cartes locales contiennent des buts intermédiaires et vont permettre à l'agent de réagir en temps réel dans un environnement dynamique. La méthode *RVO* (Reactive Velocity Obstacles) [22] se focalise sur l'évitement d'obstacles statiques et dynamiques. Enfin, [23] donne la possibilité aux agents de définir des *ROI* (Region Of Interest) tout au long de leurs parcours afin de rendre celui-ci unique.

Le GPGPU offre donc des performances jamais atteintes sur des machines classiques surtout depuis l'arrivée de *CUDA* et *OpenCL*. Cependant, les travaux de *Aaby* [24] rappellent que ce n'est pas juste en portant un modèle sur GPU que l'on gagne en performance et que, malgré l'avènement de *CUDA* et *OpenCL*, il reste nécessaire de prendre en compte les spécificités de l'architecture matérielle et du paradigme de programmation sous-jacents au GPGPU.

5 Apparition de systèmes hybrides

Jusqu'ici nous avons vu des contributions qui exécutent toute la simulation sur le GPU. Nous présentons maintenant des systèmes *hybrides*, c'est-à-dire des solutions où l'implémentation du SMA est divisée de telle sorte que le CPU et le GPU sont utilisés conjointement pour l'exécuter. Ce type d'approche permet notamment d'élargir les possibilités de portage sur GPU des SMA et d'assurer une plus grande compatibilité avec les autres technologies. Par exemple, [25] propose un framework de simulation de trafic qui assiste le développeur dans la réalisation de réseaux représentés par des graphes. L'utilisation d'une approche hybride permet ici d'offrir une compatibilité avec *MATSim* (Multi-Agent Transport Simulation).

Dans le cadre du portage du modèle *Sworm* sur GPU [26], l'utilisation d'une architecture hybride est motivée par une simulation multi-niveaux comportant deux modèles d'agent différents, l'un plutôt réactif (niveau micro) et l'autre plutôt cognitif (niveau macro). Cet article démontre aussi qu'un portage sur GPU n'implique pas forcément un gain de performance. En particulier, ce travail illustre l'importance des structures de données : celles qui seront utilisées par le GPU doivent se prêter à un calcul massivement parallèle, l'indépendance étant ici un point clé. Ainsi, une approche hybride permet l'intégration d'agents hétérogènes en offrant plus de liberté et en levant les contraintes d'une implémentation intégrale sur GPU.

Un autre exemple de l'intérêt des systèmes hybrides est donné dans [27]. Ces travaux présentent trois approches différentes pour l'implémentation d'un gestionnaire de tâche et d'ordonnancement des actions dans un SMA : (1) approche *tout-sur-CPU*, (2) approche *tout-sur-GPU*, et (3) *approche hybride*. Les auteurs discutent des avantages et inconvénients de chacune des solutions et montrent que l'approche hybride est la plus prometteuse pour ce contexte applicatif, car la contrainte d'exécuter des tâches simples et indépendantes n'existe plus.

Dans [28], une approche hybride a été choisie pour intégrer des modules GPU indépendants dans *TurtleKit*, une plate-

forme de simulation multi-agents générique. L'objectif est de profiter du GPGPU tout en conservant l'accessibilité et la facilité de réutilisation, notamment grâce à une conservation de l'interface de programmation orientée objet originelle. Ainsi, l'intégration de deux modules GPU (diffusion de phéromones digitales et perception de gradients), est entièrement transparente pour l'utilisateur et ne change donc pas l'utilisation de la plate-forme. Encore une fois, ce résultat est obtenu grâce à une approche hybride qui facilite la séparation explicite entre le calcul du comportement des agents (CPU) et les dynamiques environnementales (GPU).
Enfin, certaines recherches proposent une architecture logicielle de haut niveau qui se focalise sur le déploiement de SMA sur des systèmes matériels hétérogènes et distribués. Par exemple, pour des simulations de foules, [29] définit une architecture logicielle divisée en deux : le *Action Server* (AS) et le *Client Process* (CP). L'AS doit prendre en charge le calcul de la simulation pendant que le CP s'occupe de la gestion du comportement et des états des agents.
Pour conclure, la figure 2 donne une chronologie (1) des événements marquants relatifs au GPGPU et présente aussi les différentes contributions classées et colorées selon leurs caractéristiques d'implémentations : (2) utilisation direct des fonctions graphiques du GPU, (3) utilisation de CUDA ou OpenCL, (4) utilisation d'une approche hybride.

6 Analyse de l'existant

Des sections précédentes ressortent trois perspectives essentielles : la généricité des solutions, le type de modèles multi-agents et l'accessibilité.

Généricité des solutions
La première remarque est la quasi inexistence de plates-formes génériques pour les systèmes multi-agents sur GPU. Les solutions étudiées se placent pour la grande majorité dans un contexte de simulation particulier, les rendant ponctuelles et difficilement généralisables. Seules les travaux de *Lysenko* [3], *ABGPU* [4], *Flame GPU* [6, 7, 8, 9] et *Michel* [28] essaient d'offrir une réelle alternative en proposant des outils et frameworks qui sont le moins possible liés à un domaine d'application particulier.

Type de modèles multi-agents
La nature des modèles multi-agents portés sur GPU est fortement liée à l'évolution technologique de ces derniers. Au début du GPGPU, le faible nombre de contributions s'explique par l'extrême complexité d'implémentation (utilisation des fonctions graphiques) mais aussi par les limites du matériel (taille de la mémoire, bande passante). Ainsi, les modèles étaient composés d'agents purement réactifs. En outre, la modélisation de l'environnement n'était pas forcément explicitée et celui-ci n'avait qu'un rôle très limité dans le modèle. Malgré ces restrictions, *ABGPU* fut le premier à proposer un framework non restreint à un domaine, même s'il restait limité aux agents purement réactifs.
Avec la sortie des interfaces spécialisées, le nombre de SMA sur GPU augmente. Cependant, malgré les simplifications importantes apportées par ces technologies, peu de travaux se focalisent sur l'expressivité des modèles développés et seule la recherche de performance est privilégiée. De fait, une grande majorité des portages sont faits en repartant de zéro et les modèles d'agents restent donc limités au contexte pour lequel ils ont été créés. Néanmoins, l'architecture des agents évoluent et on voit apparaître des architectures cognitifs [17] et hétérogènes [14, 30]. *Flame GPU* fournit même des modèles agents prédéfinis pouvant s'adapter à une grande quantité de domaines. Par ailleurs, grâce aux simplifications apportées par *Cuda* puis *OpenCL* sur la gestion des structures de données, on voit aussi que l'environnement retrouve une place importante, comme dans la simulation de foules [17] avec la gestion des forces d'influences ou dans les algorithmes de déplacements [22, 21, 23] qui délèguent la gestion des objets statiques et dynamiques qui le composent.
Parallèlement, dans le but de concilier efficacité et modèles multi-agents plus complexes, les systèmes hybrides sont apparus. Utilisant le couple CPU / GPU, ces systèmes ne limitent plus les types de modèles envisageables et vont ainsi autoriser une réelle modularité dans le design du SMA (e.g. [29]). Notamment, les approches hybrides permettent d'envisager des simulations composées à la fois d'agents réactifs et cognitifs (e.g. [26]), ou encore d'aller plus loin dans l'indépendance entre agent et environnement à des fins de généricité (e.g. [28, 27]).

Accessibilité
De par la complexité du GPGPU, l'importance des travaux visant à faciliter son utilisation est rapidement identifiée dès les débuts [1]. *Lysenko* [3] et *ABGPU* [4] ont été les premiers à prendre en compte le facteur *accessibilité* afin de limiter les connaissances en GPGPU nécessaires. Le premier en fournissant des fonctions GPU prédéfinis, le second en proposant une programmation classique en C/C++.
Bien que l'apparition de *CUDA* puis d'*OpenCL* aient ensuite fortement contribué à faciliter l'accès au GPGPU, la problématique de l'*accessibilité* reste peu abordée et les outils multi-agents qui ne nécessitent pas de compétences en GPGPU sont rares. Seuls *Flame GPU* et [17] le permettent. D'autres contributions se focalisent sur la mise à disposition de fonctions déjà programmées et réutilisables, comme c'est le cas pour les algorithmes de *Pathplanning* dans [21, 22, 23].
Néanmoins ces exemples restent peu nombreux et l'utilisation du GPGPU limite encore aujourd'hui fortement l'accessibilité des travaux et leur diffusion dans la communauté. À ce titre, l'accessibilité est sans doute un point crucial pour l'avenir immédiat du GPGPU dans les SMA. Dans ce contexte, les approches hybrides apportent un début de solution intéressant car elles permettent une ouverture effective aux techniques de programmation usuelles. Par exemple, dans [26] et [28], l'interface de programmation orientée objet est conservée et le GPU est utilisé de manière transparente. Les systèmes hybrides ont ainsi cet avantage qu'ils permettent de garder une interface de programmation classique en plus de celle du GPU.

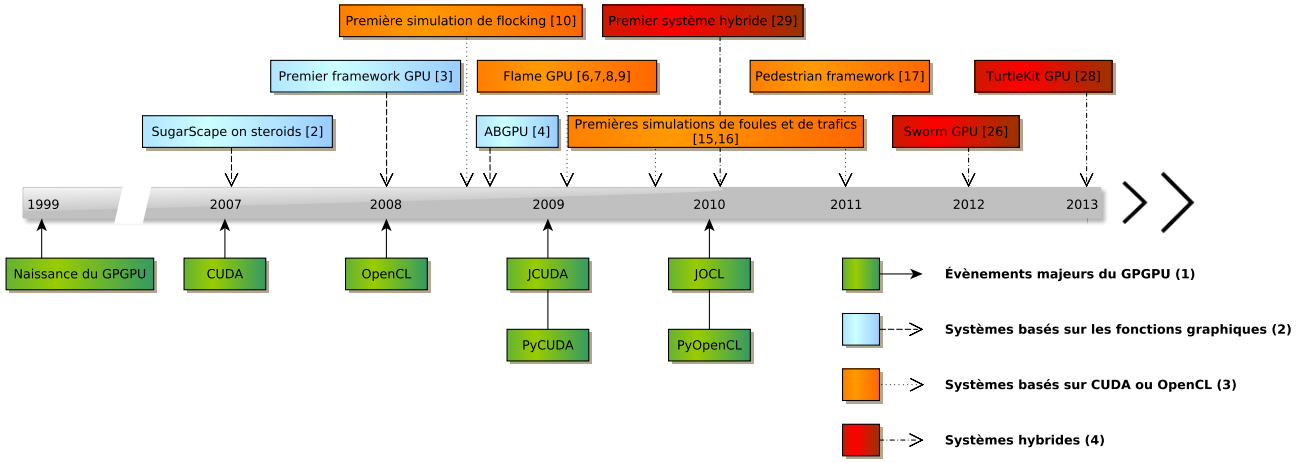

FIGURE 2 – Chronologie de l'interaction entre SMA et GPGPU

7 Conclusion et perspectives

De l'état de l'art que nous avons réalisé dans le domaine des SMA, il est clair que le GPGPU représente une technologie très intéressante pour tous les domaines où le temps d'exécution et/ou les aspects temps réels sont cruciaux.

Il est ainsi évident que le GPGPU est une technologie d'avenir pour la simulation multi-agents dans la mesure où elle permet des gains de performances très importants. Toutes les contributions ici présentées le prouvent. Pour un rapport performance prix imbattable, le GPU permet d'augmenter la taille des simulations, diminuer le temps de calcul et avoir une visualisation temps réel. Cependant, la difficulté de programmation liée à l'architecture spécifique des GPUs limite fortement le portage ou la création de modèles multi-agents.

Une des solutions est de se tourner vers les systèmes hybrides. En effet, les quelques travaux utilisant ces systèmes bénéficient de nombreux avantages. Il est ainsi possible de garder des interfaces de programmations classiques, de développer des modèles plus poussés, tout en gardant une accessibilité et une compatibilité importante. Les performances de ces systèmes hybrides peuvent être moins grandes qu'avec une approche *tout-sur-GPU*, mais c'est une concession à faire au vu des avantages offerts.

À court et moyen terme, il est donc intéressant de continuer dans ce sens. La compréhension dans ces systèmes hybrides va augmenter et faire disparaître la différence de performances avec l'approche *tout-sur-GPU*. Ils vont ainsi permettre la création ou le portage de modèle inenvisageable jusqu'à lors. Et bien sur, permettre la création de solution plus générique pouvant s'adapter à plusieurs contextes de simulations différents.

À long terme, il sera intéressant de réfléchir à des modèles multi-agents mieux adaptés aux architectures massivement parallèles des GPU, c'est-à-dire à la définition de patterns permettant une adéquation plus naturelle entre ces architectures et les systèmes multi-agents.

Références

[1] K. Perumalla and B. G. Aaby, "Data parallel execution challenges and runtime performance of agent simulations on gpus," *Proceedings of the 2008 Spring simulation multiconference*, pp. 116–123, 2008.

[2] R. D'Souza, M. Lysenko, and K. Rahmani, "SugarScape on steroids : simulating over a million agents at interactive rates," *Proceedings of Agent 2007 conference*, 2007.

[3] M. Lysenko and R. M. D'Souza, "A framework for megascale agent based model simulations on graphics processing units," *Journal of Artificial Societies and Social Simulation*, vol. 11, no. 4, p. 10, 2008.

[4] P. Richmond and D. Romano, "Agent based gpu, a real-time 3d simulation and interactive visualisation framework for massive agent based modelling on the gpu," in *In Proceedings International Workshop on Super Visualisation (IWSV08)*, 2008.

[5] S. Coakley, R. Smallwood, and M. Holcombe, "Using x-machines as a formal basis for describing agents in agent-based modelling," *Proceedings of 2006 Spring Simulation Multiconference*, pp. 33–40, 2006.

[6] P. Richmond, D. Walker, S. Coakley, and D. Romano, "High performance cellular level agent-based simulation with FLAME for the GPU," *Briefings in bioinformatics*, vol. 11, no. 3, pp. 334–47, 2010.

[7] P. Richmond, S. Coakley, and D. Romano, "Cellular Level Agent Based Modelling on the Graphics Processing Unit," in *2009 International Workshop on High Performance Computational Systems Biology*, pp. 43–50, IEEE, 2009.

[8] P. Richmond, S. Coakley, and D. M. Romano, "A high performance agent based modelling framework

on graphics card hardware with CUDA," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1125–1126, 2009.

[9] T. Karmakharm, P. Richmond, and D. M. Romano, "Agent-based Large Scale Simulation of Pedestrians With Adaptive Realistic Navigation Vector Fields," in *Theory and Practice of Computer Graphics*, pp. 67–74, 2010.

[10] E. Passos, M. Joselli, and M. Zamith, "Supermassive crowd simulation on gpu based on emergent behavior," in *In Proceedings of the Seventh Brazilian Symposium on Computer Games and Digital Entertainment*, pp. 81–86, 2008.

[11] H. Li and A. Kolpas, "Parallel simulation for a fish schooling model on a general purpose graphics processing unit," *Concurrency and Computation : Practice and Experience*, vol. 21, pp. 725–737, 2009.

[12] U. Erra, B. Frola, V. Scarano, and I. Couzin, "An Efficient GPU Implementation for Large Scale Individual-Based Simulation of Collective Behavior," *2009 International Workshop on High Performance Computational Systems Biology*, vol. 0, pp. 51–58, 2009.

[13] A. Silva, W. Lages, and L. Chaimowicz, "Boids that see : Using self-occlusion for simulating large groups on GPUs," *Computers in Entertainment*, vol. 7, no. 4, 2009.

[14] A. V. Husselmann and K. A. Hawick, "Simulating Species Interactions and Complex Emergence in Multiple Flocks of Boids with GPUs," in *International Conference on Parallel and Distributed Computing and Systems*, pp. 100–107, IASTED, 2011.

[15] D. Strippgen and K. Nagel, "Multi-agent traffic simulation with CUDA," in *International Conference on High Performance Computing& Simulation*, pp. 106–114, 2009.

[16] Z. Shen, K. Wang, and F. Zhu, "Agent-based traffic simulation and traffic signal timing optimization with GPU," *Intelligent Transportation Systems*, pp. 145–150, 2011.

[17] P. Richmond and R. Daniela, "A High Performance Framework For Agent Based Pedestrian Dynamics On GPU Hardware," *European Simulation and Modelling*, 2011.

[18] D. Helbing, I. Farkas, P. Molnar, M. Vicsek, T.and Schreckenberg, and S. D. Sharma, "Simulation of pedestrian crowds in normal and evacuation situations," *Pedestrian and Evacuation Dynamics (Berlin, Germany)*, pp. 21–58, 2002.

[19] A. Bleiweiss, "GPU accelerated pathfinding," *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pp. 65–74, 2008.

[20] G. Caggianese and U. Erra, "GPU Accelerated Multi-agent Path Planning Based on Grid Space Decomposition," in *Proceedings of the International Conference on Computational Science*, vol. 9, pp. 1847–1856, Elsevier, 2012.

[21] L. G. Fischer, R. Silveira, and L. Nedel, "Gpu accelerated path-planning for multi-agents in virtual environments," in *Proceedings of the 2009 VIII Brazilian Symposium on Games and Digital Entertainment*, pp. 101–110, IEEE Computer Society, 2009.

[22] A. Bleiweiss, "Multi agent navigation on the gpu," *Games Developpement Conference*, 2009.

[23] A. Demeulemeester and C. Hollemeersch, "Hybrid path planning for massive crowd simulation on the GPU," *Motion in Games*, pp. 304–315, 2011.

[24] B. G. Aaby, K. S. Perumalla, and S. K. Seal, "Efficient simulation of agent-based models on multi-GPU and multi-core clusters," *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010.

[25] Y. Sano and N. Fukuta, "A GPU-based Framework for Large-scale Multi-Agent Traffic Simulations," in *Proceedings of the 2013 Second IIAI International Conference on Advanced Applied Informatics*, pp. 262–267, 2013.

[26] G. Laville, K. Mazouzi, C. Lang, N. Marilleau, and L. Philippe, "Using gpu for multi-agent multi-scale simulations," in *Distributed Computing and Artificial Intelligence*, vol. 151 of *Advances in Intelligent and Soft Computing*, pp. 197–204, Springer Berlin Heidelberg, 2012.

[27] J. R.Pavlov, "Multi-Agent Systems meet GPU," in *Doctoral Conference on Computing, Electrical and Industrial Systems*, vol. 394, pp. 115–122, Springer, 2013.

[28] F. Michel, "Intégration du calcul sur GPU dans la plate-forme de simulation multi-agent générique TurtleKit 3," in *Dynamiques, couplages et visions intégratives - JFSMA 13 - Vingt-et-unièmes journées francophones sur les systèmes multi-agents, Lille, France, Juillet 3-5, 2013*, pp. 189–198, 2013.

[29] G. Vigueras, J. M. Orduña, and M. Lozano, "A GPU-Based Multi-agent System for Real-Time Simulations," in *Advances in Practical Applications of Agents and Multiagent Systems, 8th International Conference on Practical Applications of Agents and Multiagent Systems*, vol. 70, pp. 15–24, 2010.

[30] R. M. D'Souza, M. Lysenko, S. Marino, and D. Kirschner, "Data-parallel algorithms for agent-based model simulation of tuberculosis on graphics processing units," in *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, (San Diego, CA, USA), pp. 21 :1–21 :12, Society for Computer Simulation International, 2009.