# Parallélisation d'un simulateur d'agents situés

Makram Bouzid, Vincent Chevrier, François Charpillet, Stéphane Vialle

# Parallélisation d'un simulateur d'agents situés

M. BOUZID[1], V. CHEVRIER[1], F. CHARPILLET[1] et S. VIALLE[2]

[1] LORIA UMR 7503, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
[2] SUPELEC, 2 rue Edouard Belin, F-57070 Metz, France

Opérations *Systèmes multi-agents parallèles* et *VCM-ParCeL*

## 1 Description de l'application

Nous avons réalisé un simulateur d'un ensemble de robots, où chaque robot est représenté par un agent, se déplaçant dans un environnement réel, constitué des couloirs du Loria. La simulation se déroule par cycle: dans chaque cycle, chaque agent effectue sa phase de perception/décision/action.

Ce simulateur repose sur un modèle stochastique d'interaction entre l'agent et son environnement que nous avons proposé [1]. Ce modèle intègre les incertitudes et les erreurs qui peuvent se produire au niveau des capteurs et des effecteurs d'un agent, ainsi que la dynamique du système et la simultanéité des actions des agents. Il s'inspire des modèles de décision Markoviens partiellement observables (POMDP) ([5], [2]).

La modélisation de ces phénomènes peut entraîner une complexité importante du modèle et un grand besoin en capacité de calcul pour la simulation. Ceci nous a amené à une implantation parallèle du simulateur, en cherchant à profiter du parallélisme intrinsèque des SMA (systèmes multi-agents).

## 2 Parallélisation du simulateur

La simultanéité des actions des agents ne peut pas être toujours simulée de façon parallèle: deux actions simultanées, mais conflictuelles (partageant les mêmes ressources) doivent être traitées séquentiellement. On doit résoudre le conflit, puis mettre en œuvre les actions une par une. Donc la solution classique qui consiste à assimiler un agent à un processus ([7], [4]) est loin d'être la bonne solution pour paralléliser le simulateur. Il y aura beaucoup de perte de temps dans la synchronisation des processus pour gérer les conflits entre les agents.

### 2.1 Répartition spatiale

Une répartition spatiale de l'environnement en mailles, attribuant la gestion de chaque maille et des agents s'y trouvant à un processeur, ne résout pas ces problèmes de synchronisation. Les cases de l'environnement situées aux frontières des mailles peuvent être accédées simultanément par des agents situés sur des processeurs différents. De plus, la distance importante couverte par les capteurs des robots vis-à-vis de la largeur des couloirs du laboratoire donne aux frontières une taille importante comparée à celle des mailles. Les problèmes de synchronisation des accès aux cases frontières se posent donc souvent.

### 2.2 Mécanisme du double *work-pool*

Finalement, nous avons adopté une algorithmique à base de tâches traitables par tout processeur libre. Un agent représente une tâche, mais en cas de conflit entre deux ou plusieurs agents, ils sont fusionnés en une seule tâche, afin qu'elle soit traitée par un seul processeur. Ce qui implique des tâches variables en taille et en nombre au cours de chaque cycle de simulation, et nécessite une répartition dynamique de charge. Nous avons conçu un mécanisme original de double *work-pool*[1] en cascade (fig. 1). Dans le premier *work-pool*, une tâche représente un agent dans la phase de perception et de décision. Dans le second *work-pool*, une tâche représente un ensemble d'agents en conflit au cours de la phase d'action. La simulation va donc se dérouler de la façon suivante: Au

1. voir [3], [6] pour la définition des *work-pools*

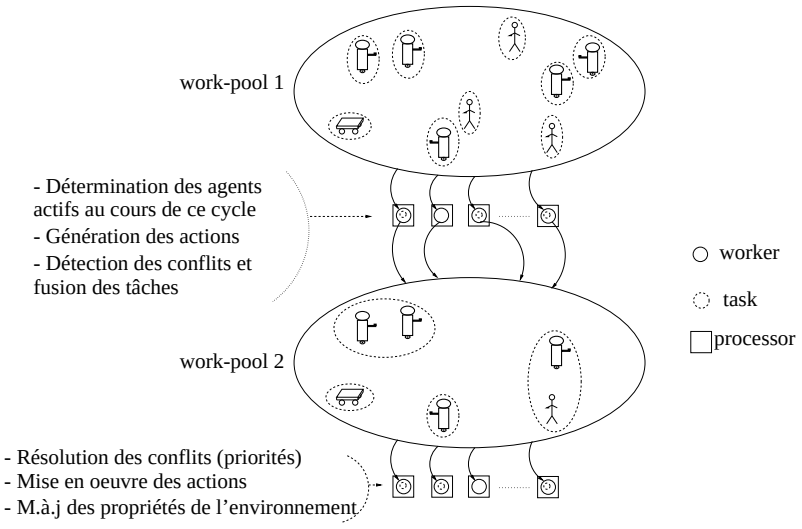


FIG. 1 – *Simulation parallèle utilisant deux work-pools*

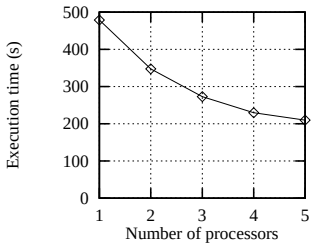


FIG. 2 – *Décroissance du temps d'exécution en fonction du nombre de processeurs*

début de chaque cycle, le premier *work-pool* contient tous les agents du système, où chaque agent représente une tâche. Chaque *worker* (un *worker* est un processus capable de traiter tout type de tâche dans le *work-pool*) cherche une tâche dans ce *work-pool* et teste si l'agent est actif (i.e s'il doit fournir une action) au cours de ce cycle ou non, en fonction de sa vitesse (la période d'action). S'il est actif, le *worker* va effectuer la phase perception/décision/action de l'agent. Ensuite, il l'insère avec l'action générée dans le second *work-pool* sous forme d'une tâche, tout en détectant la présence d'éventuels conflits entre cette nouvelle tâche et les autres déjà présentes dans le *work-pool*, et en fusionnant les tâches conflictuelles en une seule, pour qu'elles soient traitées par le même *worker*. Une fois toutes les tâches du premier *work-pool* traitées, tous les *worker*s sont synchronisés à l'aide d'une barrière, afin de passer au traitement des tâches du second *work-pool*. Chaque *worker* cherche une tâche dans le second *work-pool*. Il résout les conflits entre les différents composants de cette tâche et détermine l'ordre de priorité entre eux. Par la suite, il met en œuvre l'action de chaque composant, tout en respectant l'ordre de priorité décroissant établi. Enfin, il met à jour les propriétés de l'environnement qui ont été affectées suite à cette action. Une fois qu'on a traité toutes les tâches du second *work-pool*, les *worker*s sont synchronisés de nouveau à l'aide d'une barrière, avant de passer au cycle suivant.

# 3 Implantation et performances

L'implantation de notre simulateur a été réalisée sur la machine SGI-Origin 2000 du CCH, en utilisant la bibliothèque de processus ultra-légers ParCeL-3. Nous avons fait des tests sur des environnements de 200x300 cases, comportant 16 robots, 8 objets mobiles, 19 portes, des murs et des objets inertes. Nous avons utilisé un nombre de *worker* égal au nombre de processeurs, et nous avons lancé la simulation, en mode multi-utilisateurs, sur 20000 cycles. Le temps que nous avons mesuré comprend les calculs de chaque cycle de simulation et les opérations d'Entrées/Sorties (écriture des résultats en parallèle dans plusieurs fichiers), mais ne comprend pas la phase d'initialisation que nous n'avons pas encore parallélisée. Toutefois elle ne représente que 1% du temps total d'une

| Nombre de processeurs ($P$) | Temps d'exécution $T(P)$ (s) | Speed-up $S(P)$ | Efficacité $e(P)$ (%) |
|---|---|---|---|
| 1 | 478.79 | 1 | 100 |
| 2 | 347.27 | 1.38 | 69 |
| 3 | 272.75 | 1.76 | 59 |
| 4 | 229.52 | 2.09 | 52 |
| 5 | 209.57 | 2.28 | 46 |

TAB. 1 – *Les performances de la parallélisation*

simulation de 20000 cycles. La figure 2 récapitule ces mesures et nous montre la décroissance des temps de simulation en fonction du nombre de processeurs.

L'accélération est $S(P) = T(1)/T(P)$, $T(1)$ étant le temps d'exécution du même programme sur un processeur et avec un seul *worker* qui gère les 2 *work-pools*. Il y a donc peu de pertes de temps dans les synchronisations et nous estimons que ce temps est proche du temps d'exécution du programme séquentiel optimal de notre simulateur. L'efficacité de la parallélisation est $e(P) = S(P)/P$. Ces valeurs sont réunies dans la table 1. Nous voyons que si le temps d'exécution décroît l'accélération croit lentement et l'efficacité chute rapidement. Cependant nous obtenons avec cette première implantation parallèle un gain supérieur à 2 sur 4 processeurs. Ces résultats sont satisfaisants et nous permettent d'être optimiste pour les versions futures, car de nombreuses optimisations restent possibles.

Nous avons effectué un portage de notre simulateur sur une machine quadri-processeurs de SUPELEC (SGI-VWS 540) et des résultats équivalents jusqu'à 4 processeurs ont été retrouvés sur cette nouvelle architecture.

| Nombre de processeurs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Efficacité sur la SGI-Origin 2000 | 100% | 69% | 59% | 52% | 46% |
| Efficacité sur la SGI-VWS 540 | 100% | 83% | 69% | 55% | - |

# 4 Conclusion

Nous avons décrit dans ce papier la parallélisation de notre simulateur d'agents situés, basée sur un mécanisme de double *work-pool*. Nous avons présenté les performances obtenues avec cette parallélisation sur la machine SGI-Origin 2000 et les résultats du portage sur la SGI-VWS 540.

Comme perspectives, nous prévoyons la conception et la mise en œuvre de la parallélisation de la phase d'initialisation du simulateur, ainsi que son optimisation pour avoir de meilleurs performances.

# Références


[1] M. BOUZID, V. CHEVRIER, S. VIALLE, and F. CHARPILLET. A Stochastic Model of Interaction for Situated Agents and its Parallel Implementation. In *Proceedings of the IEEE/ACIDCA'2000 International Conference on Artificial and Computational Intelligence For Decision, Control and Automation In Engineering and Industrial Applications*, 2000. To appear.

[2] P. LAROCHE. Modélisation stochastique pour la planification en robotique: tendances actuelles. Rapport de recherche, CRIN/CNRS - INRIA Lorraine, 1997.

[3] B. LESTER. *The art of parallel programming.* Prentice Hall, 1993.

[4] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit For Building Multi-Agent Simulations. 1996. Santa Fe Institute Working Paper 96-06-042. http://www.santafe.edu/projects/swarm/overview.ps.

[5] R. D. SMALLWOOD and E. J. SONDIK. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[6] S. Vialle and T. Cornu. Parallélisme. Support de cours EPAP du DEA Informatique, Université Henri Poincaré - Nancy 1, Jan. 1997.

[7] G. M. WERNER and M. G. DYER. A Massively Parallel Simulation Environment for Evolving Distributed Forms of Intelligent Behavior. In *Massively Parallel Artificial Intelligence*, 1994.