



# MODÉLISATION ET ÉVALUATION DE PERFORMANCE D'UN ALGORITHME DE VOL DE TRAVAIL SUR DES ARCHITECTURES MULTI-COEURS

Leonardo Brenner, Sarah Nemmour, Ihab Sbeity

## ► To cite this version:

Leonardo Brenner, Sarah Nemmour, Ihab Sbeity. MODÉLISATION ET ÉVALUATION DE PERFORMANCE D'UN ALGORITHME DE VOL DE TRAVAIL SUR DES ARCHITECTURES MULTI-COEURS . MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimisation et Simulation, Nov 2014, Nancy, France. hal-01166686

**HAL Id: hal-01166686**

**<https://hal.science/hal-01166686>**

Submitted on 23 Jun 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# MODÉLISATION ET ÉVALUATION DE PERFORMANCES D'UN ALGORITHME DE VOL DE TRAVAIL SUR DES ARCHITECTURES MULTI-COEURS

L. BRENNER, S. NEMMOUR

Aix-Marseille Université, CNRS  
ENSAM, Université de Toulon  
LSIS UMR 7296  
13397, Marseille, France  
leonardo.brenner@lsis.org, nemmours@gmail.com

I. SBEITY

Département Mathématiques appliquées  
Faculté de Sciences I  
Université Libanaise  
Beyrouth, Liban  
isbeity@ul.edu.lb

**RÉSUMÉ :** *Les méthodes de vol de travail permettent la distribution et un équilibrage équitable de la charge de travail sur des machines parallèles. Lorsqu'une unité de calcul termine ses travaux, elle va voler des travaux des unités qui ne sont pas encore terminées. On s'intéresse dans cet article à la modélisation et à l'évaluation de performance d'une méthode de vol de travail stochastiques sur des machines multi-coeurs. On propose des modèles génériques pour l'algorithme de vol de travail. Ces modèles génériques seront ensuite utilisés pour modéliser une architecture constituée de trois coeurs de calcul. On termine par une analyse de mesures obtenues lors de la résolution du modèle.*

**MOTS-CLÉS :** *Vol de travail, Réseaux d'Automates Stochastiques, Évaluation de performance, Modèles stochastiques*

## 1 INTRODUCTION

Devant les limites physiques de la vitesse de calcul des microprocesseurs et le besoin de puissance pour résoudre certains problèmes (dans les applications industrielles, la simulation, la modélisation, le traitement d'images, etc.), le parallélisme apparaît comme une solution pour augmenter la puissance de calcul. Cette augmentation peut être réalisée grâce à une augmentation du nombre de processeurs et plus encore grâce à l'augmentation massive du nombre de coeurs de calcul. Actuellement, il n'est plus rare de trouver des machines de calcul composées de plusieurs processeurs, eux-mêmes constitués de plusieurs coeurs.

Bien que cette hiérarchisation des architectures actuelles assure un développement de la puissance de calcul, elle complexifie la parallélisation des applications sur les nombreuses unités de calcul qu'elles comportent.

À l'heure actuelle, cette évolution des architectures parallèles nécessite un effort important de parallélisation des applications. Les approches classiques de parallélisation reposent dans la plupart des cas sur l'utilisation de threads (Butenhof, 1997) dans le cas de plates-formes à mémoire partagée et de la bibli-

othèque Message Passing Interface (Daoudi *et al.*, 2005) dans le cas de plates-formes à mémoire distribuée. Il est tout à fait envisageable d'obtenir des performances importantes à l'aide de ces outils, mais il reste à la charge du programmeur d'équilibrer la charge de travail sur l'ensemble des ressources de calcul (Gautier *et al.*, 2007a).

Des méthodes d'équilibrage de charges efficaces sont apparues par l'introduction d'intergiciels comme par exemple, Cilk (Frigo *et al.*, 1998) et Kaapi (Gautier *et al.*, 2007b). Ces bibliothèques permettent au programmeur d'automatiser l'équilibrage de la charge de son application. Le rôle du programmeur dans ce cas, s'articule sur l'explicitation des travaux à exécuter et laisse à l'intergiciel la tâche de les ordonnancer sur les différentes ressources. Ces intergiciels se chargent de répartir les tâches sur les processeurs et coeurs de calculs à l'aide d'un algorithme d'ordonnancement de plus en plus répandu : *l'algorithme de vol de travail* (Blumofe, 1995), (Agrawal *et al.*, 2008). Ce type d'algorithme permet d'obtenir des performances élevées pour de nombreuses applications et sur de nombreuses architectures (Blumofe et Leiserson, 1999).

Bien entendu, la grande variété des architectures actuelles ainsi que la variété des bibliothèques inté-

grant le mécanisme vol de travail ont causé l'existence de plusieurs modèles d'algorithmes tels que: le vol de travail classique, hiérarchique, probabiliste, adaptatif, etc..

On s'intéresse dans cet article à évaluer les performances de l'algorithme de vol de travail classique du point de vue stochastique, sur des architectures parallèles. En particulier, on va prendre en compte les architectures multi-coeurs à mémoire partagée. Pour avoir un modèle stochastique de l'algorithme de vol de travail, on va utiliser le formalisme des Réseaux d'Automates Stochastiques (SAN). Ce formalisme consiste à décomposer le système global en plusieurs sous systèmes. La modélisation par sous-systèmes (modules) facilite la description des modèles complexes à grand nombre d'états sans perdre en expressivité.

Ce papier est organisé comme suit : la section 2 présente les concepts des paradigmes de vol de travail lorsque la section 3 présente le formalisme de Réseaux d'Automates Stochastiques. Dans la section 4, on propose des modèles génériques qui serviront à composer un modèle de l'algorithme de vol de travail pour une architecture multi-coeurs dans la section 5. Une évaluation de performances de l'algorithme modélisé est présenté dans la section 6 et on termine par adresser nos conclusions.

## 2 PARADIGMES DE VOL DE TRAVAIL

Le vol de travail est un paradigme d'ordonnancement de tâches pour les calculs parallèles. Ce paradigme a été proposé initialement par (Blumofe, 1995).

Il s'agit d'un algorithme d'ordonnancement décentralisé (Agrawal et al., 2008), (Tchiboukdjian et al., 2011) et (Tchiboukdjian, 2010), dont l'objectif est d'ordonner efficacement une application sur un nombre des machines non défini. Cette méthode de parallélisation est utilisée dans plusieurs bibliothèques, i.e. Cilk (Frigo et al., 1998), Kaapi (Gautier et al., 2007b), Satin (Van Nieuwpoort et al., 2005).

Le principe de fonctionnement de l'ordonnancement par vol de travail est le suivant (Blumofe et Leiserson, 1999) : chaque processeur, ou coeur de calcul, possède une liste qui contient l'ensemble de tâches qu'il doit effectuer. Si un processeur devient inactif (plus de tâches dans sa liste), il cherche à récupérer du travail dans la liste de tâches d'un autre processeur et continue cela tant qu'il n'arrive pas à en trouver.

Lors de l'exécution d'une tâche, le processeur exécute les instructions associées et cela peut générer d'autres tâches. Ces tâches seront placées dans sa liste pour la suite de l'exécution. L'ensemble des tâches à exécuter et leurs données décrivent entièrement l'application qui est représentée sous forme d'un graphe.

### 2.1 États des processeurs

Lorsqu'un processeur termine l'exécution d'une tâche, deux cas sont distingués :

- Sa liste locale contient encore des tâches. Dans ce cas, il prend la plus récente parmi celles-ci et démarre localement son exécution.
- Sa liste est vide. Dans ce cas il tente de trouver du travail auprès des autres processeurs.

Donc en fonction de la quantité de travail existante dans sa liste de tâches, un processeur peut évoluer entre deux états : *état travailleur* et *état voleur*. Un processeur qui est à l'état voleur doit effectuer une requête de vol auprès d'un autre processeur qu'il choisit d'une manière aléatoire et essaye de récupérer si possible une ou plusieurs tâches. Si un processeur reçoit une demande de travail et s'il a des tâches dans sa liste, il envoie du travail au voleur, qui passe à l'état travailleur. Dans le cas contraire l'état voleur reste inchangé tant qu'il ne trouve pas un autre processeur ou il pourra voler du travail. Le processeur volé s'appelle *victime*.

### 2.2 Envoi d'une demande de vol

Lorsqu'un processeur devient inactif (plus des tâches dans sa propre liste), il choisit le processeur victime dont il va prendre une partie de son travail. Un des points clef de l'algorithme de vol de travail est le choix de la victime lors d'une demande de travail. Ce choix est réalisé d'une façon aléatoire et uniforme parmi les processeurs participant à l'exécution. Cette politique, appelée *random*, est simple à implanter et largement utilisée dans les différentes bibliothèques.

Les analyses théoriques montrent qu'avec une telle sélection de la victime, la probabilité de sélectionner un processeur ayant une quantité importante de travail est grande, même avec un nombre limité de vols (Arora et al., 2001). Ce choix reste simple et un des plus performant parmi les autres implémentations et il peut être implanté facilement dans une architecture distribué sans nécessiter de synchronisation.

Un autre choix qui a été analysé, est le choix du processeur ayant la tâche la moins profonde dans sa pile. Son avantage est que le temps de vol est réduit par rapport à celui du choix aléatoire, mais cette solution nécessite d'avoir l'information du processeur possédant cette tâche sur toute la plate forme (Roch). Cela nécessite un nombre de synchronisation importants.

Les atouts principaux du choix de vol aléatoire sont les bornes théoriques sur le temps d'exécution et le nombre de requêtes de vol introduits (Blumofe et Leiserson, 1999).

### 2.3 Réponse à une demande de vol

Le processeur qui reçoit une requête du vol et ayant des tâches dans sa liste doit sélectionner une quantité de travail pour être transférée vers la liste du voleur. L'algorithme réalisant la sélection du travail à transférer vers le voleur s'exécute en concurrence du travail effectué par le processeur volé.

Dans le cas de l'algorithme de vol de travail classique, la tâche sélectionnée pour être exécutée est la dernière tâche ajoutée à la file. Blumofe et Leiserson (1999) ont proposé un algorithme pour le vol de travail basé sur des files à deux point d'accès (deque). Chaque processeur possède une deque de tâches, qu'il utilise comme une pile pour ses propres tâches : il empile les tâches qu'il crée en bas de la deque et lorsqu'il complète une tâche, il dépile une nouvelle tâche du bas de la deque, si elle n'est pas vide (ordre LIFO). Alors, pour ne pas perturber le travail de la victime, les tâches volées sont choisies prioritairement en haut de la deque (ordre FIFO).

Il reste à définir le nombre de tâches prises dans la file. Lors d'un vol, le nombre de tâches qui sont transférées vers le processeur voleur est déterminé par la fonction de vol de travail  $f$ . En particulier, si le processeur qui accepte une demande a  $l$  tâches stockées dans sa file, alors  $f(l)$  tâches sont transférées à celui qui est actuellement vide. Classiquement, on utilise la fonction de vol de travail  $f(l) = l/2$ , qui transfère (à peu près) la moitié des tâches (Hendler et Shavit, 2002), (Tchiboukdjian *et al.* 2010) et (Tchiboukdjian *et al.* 2011). L'objectif étant d'équilibrer le travail entre les deux processeurs (le voleur et la victime). Sur des architectures hétérogènes, la quantité de tâches volées peut être fonction de la vitesse de processeurs et/ou de vitesse de transfère des tâches d'une liste à l'autre.

Dans la prochaine section, on va présenter le formalisme de Réseaux d'Automate Stochastiques, qui sera utilisé pour la modélisation de l'algorithme de vol de travail stochastiques présenté ci-avant.

## 3 RÉSEAUX D'AUTOMATES STOCHASTIQUES

Le formalisme des Réseaux d'Automates Stochastiques (*Stochastic Automata Networks* - SAN) introduit par Plateau (Plateau, 1984) constitue un outil de modélisation efficace pour des systèmes complexes à grand espace d'état et également la résolution des ces modèles. Le principe de base de ce formalisme est de décrire un système comme un ensemble de sous-systèmes qui interagissent. Chacun de ces sous-systèmes est vu comme un automate stochastique. L'interaction entre les sous systèmes est modélisée par des règles établies entre les états internes de chaque automate (Fernandes, 1998). Le modèle SAN peut

être représenté par un seul automate global qui comporte tous les états possibles du système modélisé.

### 3.1 Automates stochastiques

Un automate stochastique est un modèle mathématique d'un système qui possède des entrées et sorties discrètes. On peut décrire un automate stochastique comme un *ensemble d'états* et un *ensemble de transitions* entre eux (Plateau et Fourneau, 1991). Le système décrit peut se trouver dans un quelconque état ou configuration interne. L'état dans lequel le système se trouve résume les informations sur les entrées précédentes et indique ce qui est nécessaire pour déterminer le comportement du système pour les entrées suivantes (Hopcroft et Ullman, 1979).

Graphiquement, les automates sont décrits par des graphes orientés où les noeuds représentent les états internes et les arcs représentent les transitions entre les états.

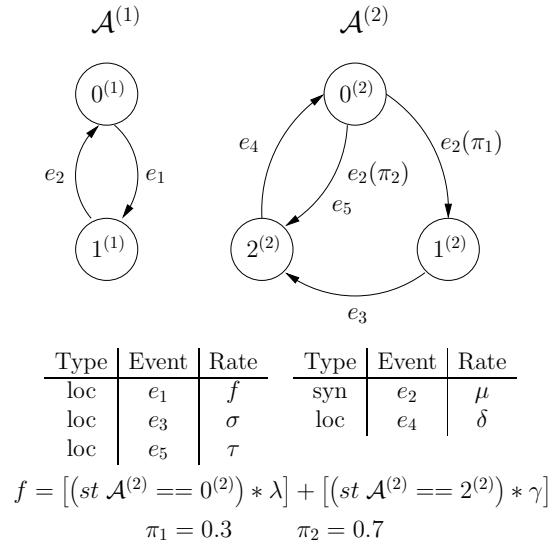


Figure 1 – Modèle SAN avec 2 automates

La figure 1 présente un modèle SAN avec 2 automates.

### 3.2 Événements

L'état d'un modèle SAN, appelé *état global*, est la combinaison de tous les *états locaux* de tous les automates.

Le changement de l'état global du système est le résultat de l'occurrence d'un événement. On a deux types d'événements : *local* et *synchronisant*.

- **événement local** : Un événement local apparaît dans un seul automate. Ce type d'événement modifie l'état interne uniquement de l'automate

où il apparaît, sans aucun changement d'état des autres automates du modèle. L'occurrence de cet événement permet aux automates de travailler de façon indépendante.

- **événement synchronisant** : Un événement synchronisant apparaît obligatoirement dans plusieurs automates. L'occurrence d'un tel événement correspond au changement simultané de l'état interne d'au moins deux automates, puisqu'il se produit aux points de synchronisation. Cette dernière n'est possible que si et seulement si les automates concernés par cet événement sont tous dans un état qui leur permet de déclencher l'événement. Notons que l'occurrence de cet événement exprime une interaction entre les automates qui sont concernés par l'événement.

Dans le modèle de la figure 1 les événements  $e_1, e_3, e_4$  et  $e_5$  sont des événements locaux (ils apparaissent chacun dans un seul automate) et vont provoquer le changement de l'état uniquement de l'automate que leur concernent.  $e_2$  est un événement synchronisant et va provoquer le changement des états internes des deux automates. L'occurrence de l'événement  $e_2$  pourra avoir lieu uniquement lorsque l'automate  $\mathcal{A}^{(1)}$  est dans l'état  $1^{(1)}$  et l'automate  $\mathcal{A}^{(2)}$  dans l'état  $0^{(2)}$ .

À chaque transition du modèle, un ou plusieurs événements sont associés. L'occurrence d'un événement déclenche la transition à laquelle il est associé. Si un événement est associé à deux transitions, ou plus, partent d'un même état, une probabilité de routage doit être associée au couple événement-transition. L'événement  $e_2$  dans l'automate  $\mathcal{A}^{(2)}$  de la figure 1 présente ce cas. Alors, on a associé une probabilité  $\pi_1 = 0.3$  pour la transition de l'état  $0^{(2)}$  vers  $1^{(2)}$  et  $\pi_2 = 0.7$  pour la transition de l'état  $0^{(2)}$  vers  $2^{(2)}$ . Ça représente un choix aléatoire déclenché par le même événement.

À chaque événement, on associe un nom et un taux d'occurrence. Le taux d'occurrence définit la fréquence avec laquelle l'événement aura lieu. Le taux d'occurrence peut avoir une valeur constante ou fonctionnelle.

### 3.3 Taux et probabilités fonctionnels

Le taux et la probabilité d'occurrence associés à un événement dans un automate peuvent être une fonction de l'état interne d'autres automates. Dans ce cas, on a de taux et/ou probabilités dites *fonctionnels*. Une fonction permet d'associer à un même événement différentes valeurs déterminant son taux d'occurrence en fonction de l'état local des autres automates du modèle. D'une manière similaire, les probabilités de routage d'un événement peuvent être exprimées par

des fonctions. Le modèle dans la figure 1 possède un taux fonctionnel associé à l'événement  $e_1$ .

La fonction<sup>1</sup>  $f$  associée à l'événement  $e_1$  dans l'automate  $\mathcal{A}^{(1)}$  est évaluée à  $\lambda$  si l'automate  $\mathcal{A}^{(2)}$  est dans l'état  $0^{(2)}$ , à  $\gamma$  si il est dans l'état  $2^{(2)}$  et zéro si il est dans l'état  $1^{(2)}$ .

On propose dans la section suivante de modèles SAN génériques pour la modélisation de l'algorithme de vol de travail classique.

## 4 MODÈLES SAN POUR VOL DE TRAVAIL

Le modèle général de l'algorithme de vol de travail classique est constitué de trois types d'automates:

- les coeurs de calcul de l'architecture du support d'exécution;
- les listes de tâches de chaque coeur;
- la liste de tâches restantes à exécuter.

Les états de ces automates décrivent l'architecture (coeurs et mémoire) de la machine pendant que les transitions et les événements associés décrivent l'algorithme de vol de travail.

Chaque type d'automate est détaillé par la suite.

### 4.1 Automate des coeurs de calcul

L'automate des coeurs de calcul est composé de deux états :

- État travailleur (**T**) : le coeur est en train d'exécuter une tâche, pendant l'exécution de la tâche, le coeur peut générer d'autres tâches, qui seront ajoutées à la liste de tâches du coeur.
- État voleur (**V**) : le coeur est en train de chercher des tâches à voler dans les listes des autres coeurs.

La figure 2 présente l'automate des coeurs de calcul. On a un automate  $C^{(i)}$  par coeur dans une architecture multi-coeurs avec  $c$  coeurs.

Dans cet automate, l'événement  $Rech\_trav\_C_i$  est un événement local. Cela permet le passage du coeur de l'état travailleur (T) à l'état voleur (V). Cet événement possède un taux fonctionnel qui permet son occurrence uniquement lorsque la liste de tâches associée à ce coeur est dans l'état 0 et il reste de tâches

<sup>1</sup>La notation utilisée dans les fonctions est celle du logiciel PEPS (Brenner et al, 2007) où **st** indique l'état d'un automate.



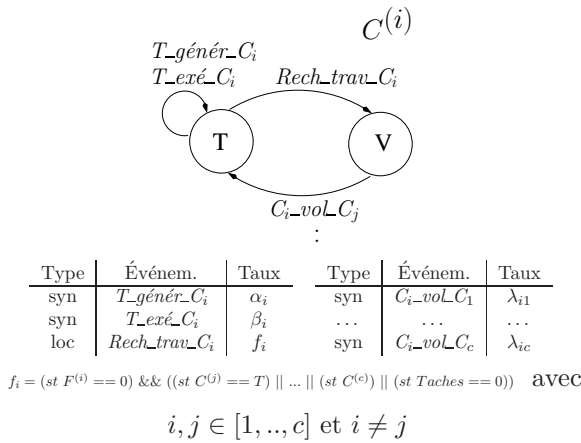


Figure 2 – Automate de l'unité de calcul

à exécuter dans la liste de tâches restantes ou un des autres coeurs est dans un état  $T$ .

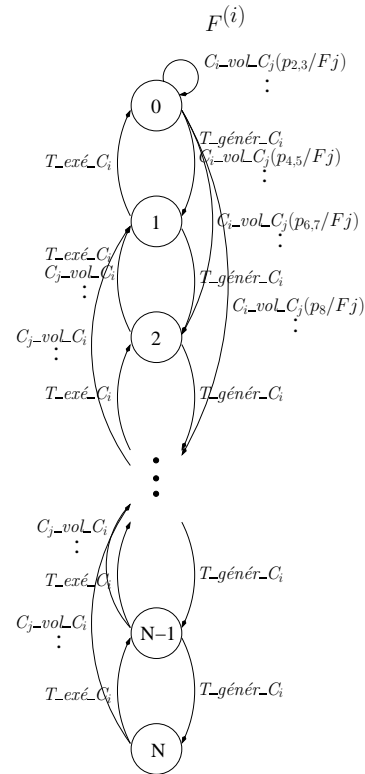
Les autres événements du modèle sont synchronisants, l'événement  $T\_génér\_C_i$  peut avoir lieu lorsque le coeur est en train d'exécuter des tâches, il représente la génération de tâches par le coeur au cours d'une exécution. À chaque fois que l'événement  $T\_génér\_C_i$  a lieu, une tâche est ajoutée à la liste de tâches du coeur et enlevée de la liste de tâches restantes. L'événement  $T\_exé\_C_i$  représente l'extraction d'une tâche de la liste pour être exécutée. L'ensemble d'événements  $C_i\_volC_j$  se produit quand le coeur réussit à voler des tâches à d'autres coeurs, ce qui va permettre le passage du coeur de l'état V à l'état T. On associe chaque événement  $C_i\_volC_j$  au vol d'une liste spécifique suivant l'indice  $j$ . On peut rajouter que l'indice  $i$  correspond au coeur courant (voleur) et l'indice  $j$  correspond aux autres coeurs (victimes). On remarque que  $i$  doit être différent de  $j$  car un coeur ne peut pas voler des tâches de sa propre liste.

## 4.2 Automate de listes de tâches des coeurs

Le deuxième type d'automate décrit la liste de tâches de chaque coeur, chaque état de cet automate indique la quantité de tâches présente dans la liste.

Chaque automate  $F^{(i)}$  est constitué de  $N + 1$  états, chaque état représente la quantité de tâches existante dans la liste d'exécution de chaque coeurs. L'état 0 nous informe que la liste de tâche est vide, l'état  $N$  représente la quantité maximale de tâches que la liste peut supporter. La figure 3 présente un exemple de cet automate. Comme pour les automates des coeurs de calcul, on a un automate de liste de tâches  $F^{(i)}$  par coeur  $i$  dans une architecture à  $c$  coeurs.

On constate que cet automate comporte les mêmes événements que l'automate de coeurs.



Type	Événem.	Taux	Type	Événem.	Taux
syn	$T_{génér}C_i$	$\alpha_i$	syn	$C_1volC_1$	$\lambda_{11}$
syn	$T_{exé}C_i$	$\beta_i$	...	...	...
			syn	$C_cvolC_c$	$\lambda_{cc}$

$$p_{l,l+1/F_j} = (st F^{(j)} == l) || (st F^{(j)} == l + 1)$$

avec  $i, j \in [1, \dots, c]$  et  $i \neq j$  et  $l \in [2, \dots, N]$

Figure 3 – Automate de listes de tâches des coeurs

L'événement synchronisant  $T_{\text{génér-}C_i}$  dans cet automate représente l'insertion des tâches générées par le coeur  $i$  au cours de son exécution. En effet, pour chaque insertion d'une tâche, l'état de la liste va s'incrémenter vers l'état suivant. L'événement  $T_{\text{exé-}C_i}$  peut avoir lieu lorsque le coeur accède à sa liste afin de récupérer une tâche pour continuer l'exécution. Cela va permettre de décrémenter l'état de la liste vers l'état précédent.

#### 4.2.1 Quantité de tâches volées

L'ensemble d'événements  $C_{i\_vol}C_j$  représente l'insertion d'une certaine quantité de tâches volées par le coeur  $i$  au coeur  $j$ . Ces événements permettent, dans la liste du coeur voleur, une transition de l'état 0 (liste vide) vers un autre état qui dépend de la quantité de tâches volées. Le nombre de tâches volées peut aller de 1 à  $N$  tâches. Dans tous les cas, l'ensemble d'événements  $C_{i\_vol}C_j$  reste inchangé. Uniquement les fonctions de probabilité de routage

et les transitions sont modifiées.

On présente par la suite les fonctions et transitions pour trois cas de vol.

**Une seule tâche** Dans le cas du vol d'une seule tâche, on aura une transition de l'état 0 vers lui-même dans la liste du coeur voleur car la tâche est exécutée immédiatement. À cette transition, on associe les événements de vol  $C_i\text{-vol}\text{-}C_j$  où le coeur  $i$  vole le coeur  $j$ .

Aucune fonction est nécessaire pour modéliser la quantité de tâches présentes dans la liste du coeur victime car pour tous les états où l'événement apparaît dans la liste du coeur victime, il aura au moins une tâche à voler. Dans la liste du coeur victime, on passe de l'état  $k$  vers l'état  $k - 1$  avec  $k \in [1..N]$ .

**50% des tâches** Quand on vole 50% des tâches d'une autre liste, il nous faut connaître la quantité des tâches présentes dans la liste qui sera volée.

Dans ce cas, les événements  $C_i\text{-vol}\text{-}C_j$  modélisent l'insertion d'une quantité de tâches  $l/2$  volée par le coeur  $i$  au coeur  $j$  où  $l$  représente le nombre de tâches dans la liste victime. Ces événements permettent une transition dans la liste de tâches de l'état 0 vers l'état  $((l/2) - 1)$ . On a, dans ce cas, le même événement associé à différentes transitions partent du même état. On associe, à chaque couple événement-transition, une probabilité de routage fonctionnelle  $p_{l,l+1/F_j}$ . Ces probabilités servent à déterminer la quantité de tâches existante dans la liste du coeur victime au moment de vol ( $l$  ou  $l + 1$ ). En fonction de ces quantités, on peut savoir la quantité de tâches  $l/2$  volée par ce coeur. Par exemple  $p_{2,3/F_2}$  indique que le coeur voleur a trouvé 2 ou 3 tâches sur la liste du coeur victime 2. Puisque le coeur voleur peut voler jusqu'à la moitié de la quantité existante dans la liste de victime, alors dans les deux cas le coeur voleur peut voler une seule tâche. La liste du coeur voleur va donc rester dans l'état 0, car ce coeur va commencer l'exécution de cette tâche instantanément avec le vol. L'indice  $j$  pour cet événement correspond au numéro du coeur victime.

De même manière, l'ensemble d'événements  $C_i\text{-vol}\text{-}C_j$  représente l'opération de vol effectuée par un coeur quelconque de l'architecture vers cette liste. Cela va permettre la transition de la liste vers un état qui correspond à peu près à la moitié de la quantité de tâche existante au moment de vol. Par exemple, si la liste est dans l'état  $N$ , lors du vol, elle va passer à l'état  $N/2$ . Si elle est dans l'état 3, lors du vol, elle va passer à l'état 2, car on a une seule tâche volée. L'indice  $j$  pour cet événement renvoie sur le numéro du coeur voleur et l'indice  $i$  correspond au coeur courant. La figure 3 représente ce cas.

**100% des tâches** Dans le cas du vol des 100% des tâches, il nous faut également connaître la quantité de tâches de la liste du coeur victime.

Comme pour le cas de vol de 50% des tâches, on associe à chaque couple événement-transition une probabilité de routage fonctionnelle  $p_{l/F_j}$ . Cette fonction indique que le coeur voleur a trouvé  $l$  tâches dans la liste du coeur  $j$ . La liste du coeur voleur va passer de l'état 0 vers l'état  $l - 1$ . Par exemple, on associe la fonction  $p_{4/F_3}$  à l'événement  $C_1\text{-vol}\text{-}C_3$  et à la transition de l'état 0 vers l'état 3 pour indiquer que le coeur 1 a volé quatre tâches de la liste du coeur 3. La liste du coeur victime va au même instant passer de l'état 4 à l'état 0.

### 4.3 Automate de la liste de tâches restantes

L'automate *Taches* représente le nombre totale de tâches que le programme doit exécuter et qui n'ont pas encore été générées.

Cet automate n'a qu'un type de transition, d'un état  $k$  vers un état  $k-1$ , comme on peut voir sur la figure 4. On associe à chaque transition tous les événements de génération de tâches ( $T\text{-génér}\text{-}C_i$ ) de tous les coeurs du modèle. À chaque fois qu'une tâche est générée par un coeur, le nombre total de tâches restant à exécuter se réduit d'une.

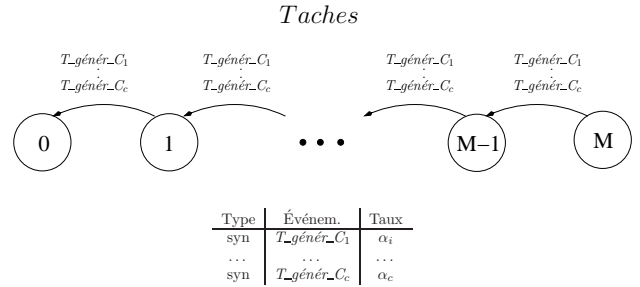
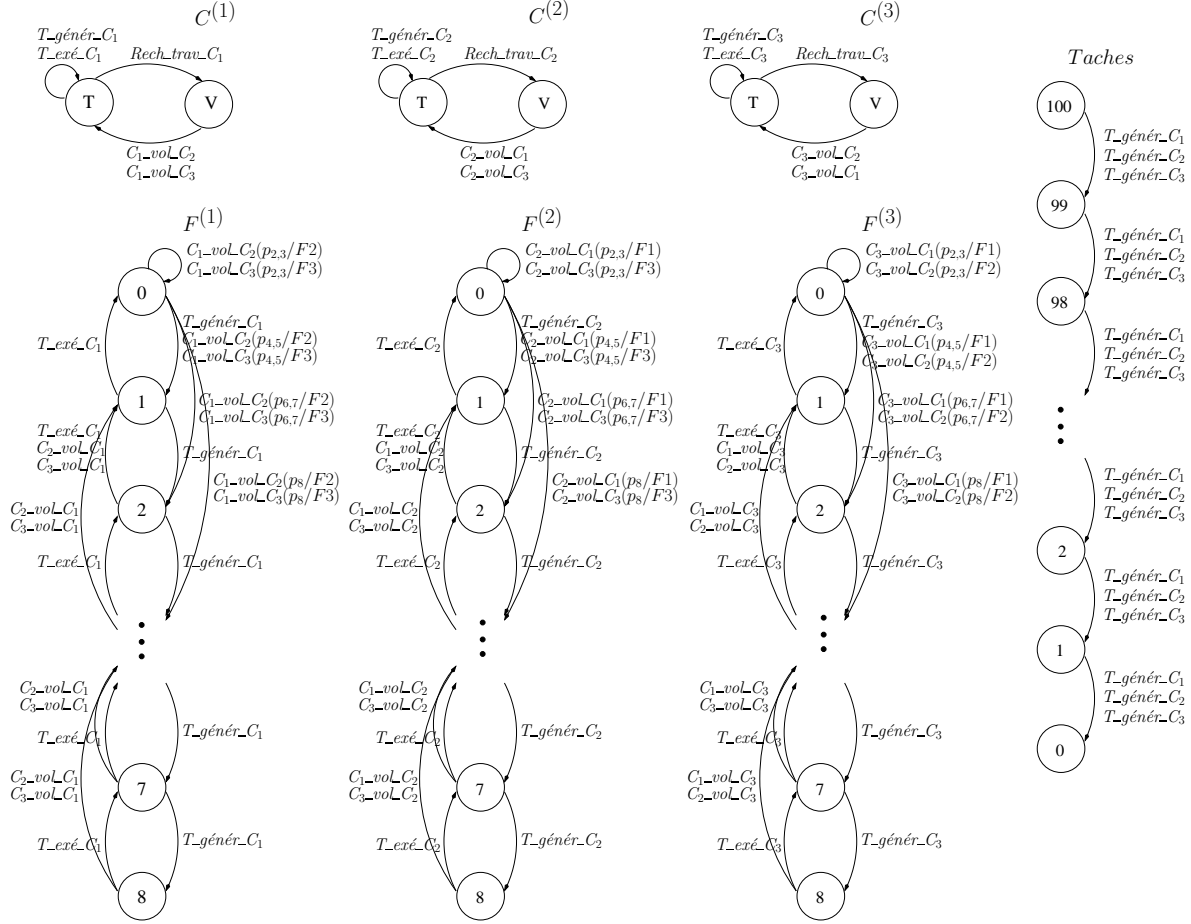


Figure 4 – Automate de la liste globale de tâches

Une fois que l'automate est dans l'état 0, plus aucune tâche n'est générée et les coeurs vont exécuter les tâches encore dans ses propres listes de tâches, jusqu'à les terminer.

## 5 EXEMPLE DE MODÉLISATION

En utilisant le modèle général vu dans la section précédant, on propose dans cet section la modélisation de l'algorithme de vol de travail classique sur une architecture constituée de trois coeurs. On aura pour ce modèle trois automates pour les unités de calcul (coeurs), trois automates pour les listes de tâches et un automate pour la liste de tâches restantes. La figure 5 présente le modèle SAN pour cette architecture



Type	Événem.	Taux	Type	Événem.	Taux	Type	Événem.	Taux
syn	$T\_génér\_C_1$	1.2	syn	$C_1\_vol\_C_2$	5.0	loc	$Rech\_trav\_C_1$	$f_1$
syn	$T\_génér\_C_2$	0.4	syn	$C_1\_vol\_C_3$	5.0	loc	$Rech\_trav\_C_2$	$f_2$
syn	$T\_génér\_C_3$	0.7	syn	$C_2\_vol\_C_1$	5.0	loc	$Rech\_trav\_C_3$	$f_3$
syn	$T\_exé\_C_1$	0.5	syn	$C_2\_vol\_C_3$	5.0			
syn	$T\_exé\_C_2$	0.5	syn	$C_3\_vol\_C_1$	5.0			
syn	$T\_exé\_C_3$	0.5	syn	$C_3\_vol\_C_2$	5.0			

$$\begin{aligned}
 f_1 &= (st\ F^{(1)} == 0) \ \&\& \ ((st\ C^{(2)} == T) \ || \ (st\ C^{(3)} == T) \ || \ (st\ Taches == 0)) \\
 f_2 &= (st\ F^{(2)} == 0) \ \&\& \ ((st\ C^{(1)} == T) \ || \ (st\ C^{(3)} == T) \ || \ (st\ Taches == 0)) \\
 f_3 &= (st\ F^{(3)} == 0) \ \&\& \ ((st\ C^{(1)} == T) \ || \ (st\ C^{(2)} == T) \ || \ (st\ Taches == 0))
 \end{aligned}$$

$$\begin{aligned}
 p_{2,3}/F2 &= (st\ F^{(2)} == 2) \ \&\& \ (st\ F^{(2)} == 3) \\
 p_{4,5}/F2 &= (st\ F^{(2)} == 4) \ \&\& \ (st\ F^{(2)} == 5) \\
 p_{6,7}/F2 &= (st\ F^{(2)} == 6) \ \&\& \ (st\ F^{(2)} == 7) \\
 p_8/F2 &= (st\ F^{(2)} == 8)
 \end{aligned}$$

$$\begin{aligned}
 p_{2,3}/F1 &= (st\ F^{(1)} == 2) \ \&\& \ (st\ F^{(1)} == 3) \\
 p_{4,5}/F1 &= (st\ F^{(1)} == 4) \ \&\& \ (st\ F^{(1)} == 5) \\
 p_{6,7}/F1 &= (st\ F^{(1)} == 6) \ \&\& \ (st\ F^{(1)} == 7) \\
 p_8/F1 &= (st\ F^{(1)} == 8)
 \end{aligned}$$

$$\begin{aligned}
 p_{2,3}/F3 &= (st\ F^{(3)} == 2) \ \&\& \ (st\ F^{(3)} == 3) \\
 p_{4,5}/F3 &= (st\ F^{(3)} == 4) \ \&\& \ (st\ F^{(3)} == 5) \\
 p_{6,7}/F3 &= (st\ F^{(3)} == 6) \ \&\& \ (st\ F^{(3)} == 7) \\
 p_8/F3 &= (st\ F^{(3)} == 8)
 \end{aligned}$$

Figure 5 – Exemple de modélisation : architecture à 3 cœurs de calcul et vol à 50% des tâches

pour le vol de 50% des tâches.

Pour montrer l'efficacité de l'algorithme de vol de travail, on a choisi trois taux distincts pour les événements de génération de nouvelles tâches  $T\_génér\_C_i$ . Dans les cœurs 1 et 3 (automates  $C^{(1)}$  et  $C^{(3)}$ ) les tâches sont générées beaucoup plus rapidement (respectivement, 1.2 et 0.7 tâches par seconde) qu'elles

sont exécutées (0.5 tâches par seconde). Cela a pour objectif de créer une accumulation de tâches dans les listes des cœurs 1 et 3 (automates  $F^{(1)}$  et  $F^{(3)}$ ). Cet accumulation sera, en partie, compensée par le cœur 2 (automate  $C^{(2)}$ ) qui a un taux d'exécution (0.5 tâches par seconde) supérieur au taux de génération de tâches (0.4).



## 6 ÉVALUATION DE PERFORMANCES

On s'intéresse dans cette section à l'évaluation transitoire du modèle proposé. On est particulièrement intéressé par des indices tels que : le taux d'utilisation des coeurs de calcul, le nombre moyen de tâches dans la liste de chaque coeur, la probabilité que toutes les tâches soient terminées dans un temps  $t$  ainsi que le nombre moyen de tâches restantes.

L'état initial du modèle est :  $C^{(1)} = T$ ,  $C^{(2)} = T$ ,  $C^{(3)} = T$ ,  $F^{(1)} = 1$ ,  $F^{(2)} = 1$ ,  $F^{(3)} = 1$  et  $Taches = 100$ . On considère que l'exécution du système est terminée lorsque le total de 103 tâches sont exécutées et tous les coeurs se trouvent à l'état  $V$ .

On a résolu le modèle SAN proposé différents temps  $t$  et pour trois quantités de tâches volées: 1 tâche, 50% des tâches et 100% des tâches. Cela nous donne l'évolution des indices qu'on veut observer sur les différentes configurations. Cela permet d'avoir une idée de l'impact de la recherche des tâches à voler. Les valeurs choisies pour  $t$  varient entre 0 et 300, avec des intervalles de 10 pour  $t$  entre 0 et 100 et des intervalles de 50 pour  $t$  entre 100 et 300.

On va par la suite discuter les indices de performances mentionnés ci-avant.

### 6.1 Utilisation des coeurs

Le taux d'utilisation des coeurs de calcul donne un bon indice de la performance de l'algorithme de vol de travail. On rappelle qu'un des objectifs du vol de travail est d'équilibrer la charge de coeurs de calcul pour accélérer la résolution d'un problème.

On peut voir sur les figures 6, 7 et 8 que dans le cas du vol de 50% des tâches, les taux d'utilisation de trois coeurs de calcul restent proches et élevés que les autres. L'utilisation des coeurs décline au peu près au même temps lorsque la quantité de travail commence à diminuer. Contrairement, on observe que dans le cas du vol d'une seule tâche et de 100% des tâches, l'utilisation des coeurs reste plus basse au début de l'exécution mais devient supérieure au vol de 50% des tâches à la fin, car il reste encore de tâches à exécuter lorsque elles commencent à se terminer pour le vol de 50% des tâches. Cette utilisation moindre des coeurs au début de l'exécution montre une mauvaise distribution des charges à ce moment là, qui peut pénaliser le temps de calcul.

### 6.2 Utilisation des listes

Le nombre moyen de tâches dans les listes de chaque coeur permet d'avoir une idée de la quantité de mémoire utilisée. La réduction des pics d'utilisation des listes (et de la mémoire) est une conséquence du

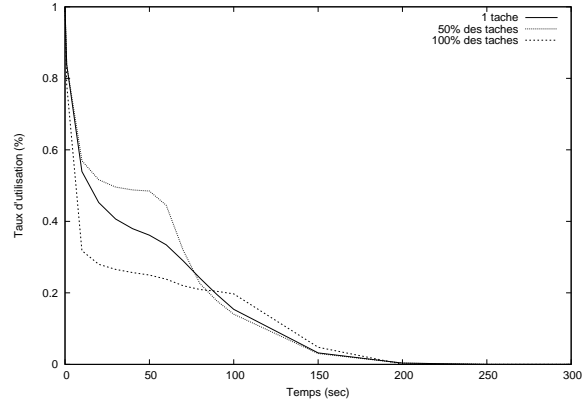


Figure 6 – Utilisation du coeur 1

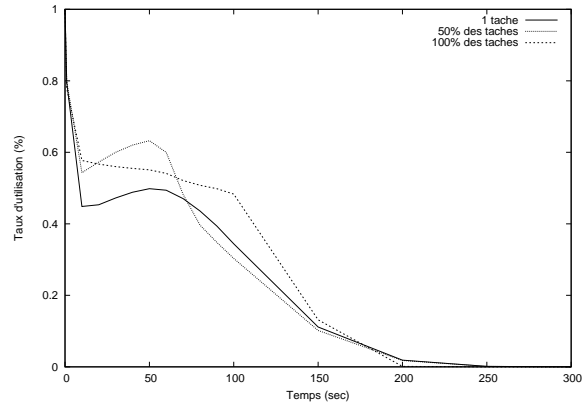


Figure 7 – Utilisation du coeur 2

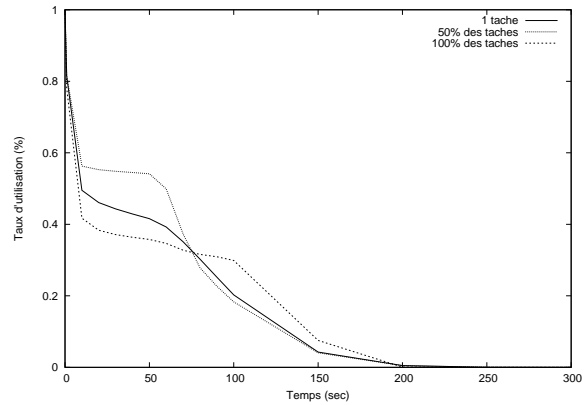


Figure 8 – Utilisation du coeur 3

vol de travail.

On peut observer sur les figures 9, 10 et 11 que le nombre moyen de tâches pour le vol d'une seule tâche et 100% des tâches restent inférieurs au vol de 50% des tâches. Cette différence peut être expliquée, dans le cas du vol d'une seule tâche, par des vols récurrents et, dans le cas du vol de 100% des tâches, par la grande quantité de tâches volées. Ça peut s'expliquer

également par une utilisation plus basse des coeurs, qui vont à leur tour générer des tâches plus lentement.

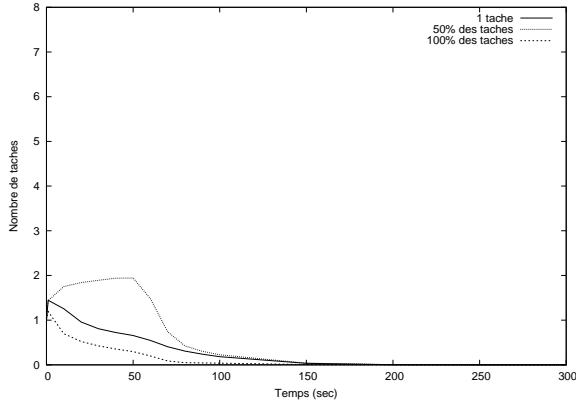


Figure 9 – Utilisation de la liste du coeur 1

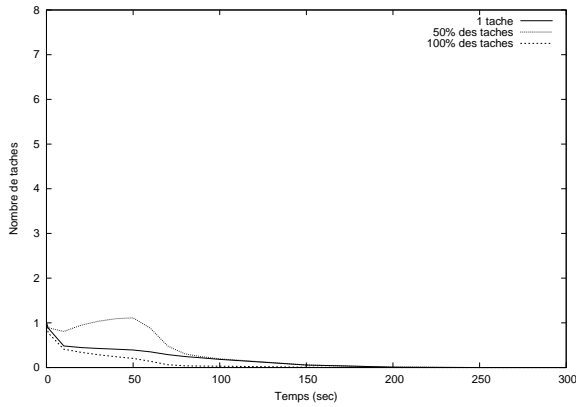


Figure 10 – Utilisation de la liste du coeur 2

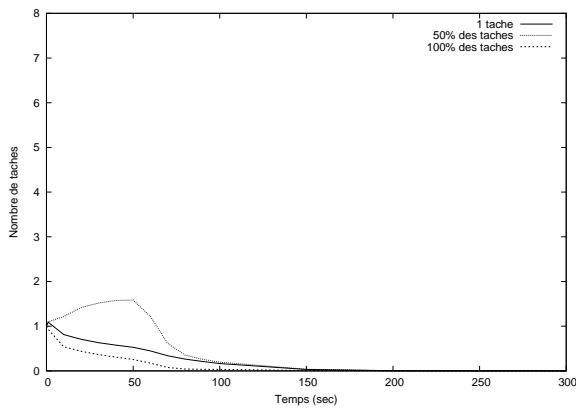


Figure 11 – Utilisation de la liste du coeur 3

### 6.3 Nombre de tâches restantes et temps d'exécution

Le nombre de tâches sur la liste de tâche restantes donne une idée de l'évolution des calculs. Plus rapidement le nombre de tâches restantes décroît, plus

rapidement le calcul sera terminé. On peut observer ce comportement sur la figure 12, où la courbe de vol de 50% des tâches décroît plus rapidement que les autres.

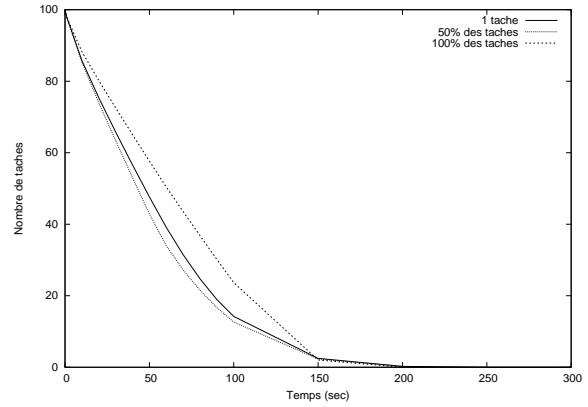


Figure 12 – Nombre de tâches restantes à exécuter

La figure 13 montre la probabilité que le calcul soit terminé pour chaque temps  $t$  observé dans la résolution du modèle.

Plus rapidement la probabilité monte, plus des tâches ont été générées et exécutées. Cependant, on peut observer un rattrapage dans le cas du vol de 100% des tâches à la fin de l'exécution. Cela peut être le reflet d'un équilibrage tardif des charges. Toutefois, une analyse plus fine est nécessaire pour vérifier cette hypothèse.

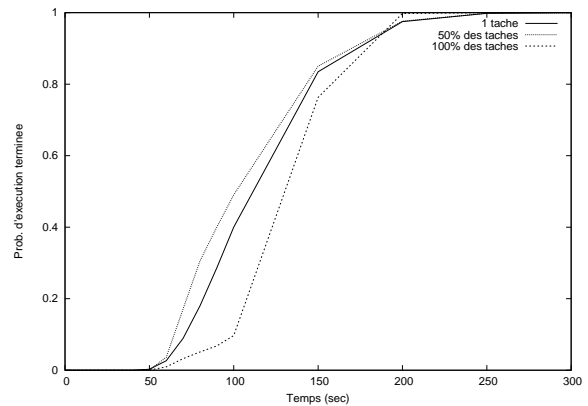


Figure 13 – Temps d'exécution

Étant donnée les mesures obtenues et pour une probabilité supérieur à 99.9% , on estimé que le cas du vol de 100% des tâches a terminé le calcul au peu près au temps  $t = 200$  lorsque les cas du vol d'une seule tâche et de 50% des tâches à  $t = 250$ .

## 7 CONCLUSION

On a proposé dans cet article un modèle stochastique pour la modélisation d'un algorithme de vol de travail classique. Cet algorithme décentralisé d'ordonnancement de tâches a été modélisé, en utilisant le formalisme SAN, par trois automates qui représentent les aspects les plus intéressants lors d'une évaluation de performance, comme par exemple, le taux d'utilisation des processeurs d'une machine, la quantité de mémoire utilisée et le temps moyen de calcul.

On a pu observer lors de la comparaison de l'algorithme de vol de travail pour trois quantités différentes de tâches volées une supériorité du cas de 50% des tâches volées dans l'utilisation des coeurs. Autrement, une meilleur utilisation, plus homogène, des coeurs de calcul. Ce qui corrobore les résultats théoriques obtenus dans d'autres travaux.

L'utilisation des fonctions dans le modèle présenté permet de facilement le modifier pour s'adapter à différentes configurations. Comme par exemple, un nombre global variable de tâches à exécuter ainsi un temps de vol variable selon la quantité de tâches volées.

La continuation de ce travail se donne par la modélisation d'autres algorithmes de vol de travail, tel que, l'algorithme de vol de travail hiérarchique. Le modélisation d'autres algorithmes permettra une analyse de performance plus fine entre ces algorithmes.

## REFERENCES

- Agrawal K. and C.E. Leiserson and Y. He and W.J. Hsu, 2008. Adaptive work-stealing with parallelism feedback. *ACM Transactions on Computer Systems*, 26(3), p. 7:1-7:32.
- Arora N.S. and R.D. Blumofe, C.G. Plaxton, 2001. Thread scheduling for multiprogrammed multiprocessors. *Theory of Computing Systemes*, 34(2), p. 115-144.
- Blumofe R.D., 1995. Executing multithreaded programs efficiently. PhD thesis, Massachusetts Institute of Technology.
- Blumofe R.D. and C.E. Leiserson, 1999. Scheduling multithreaded computations by work stealing. *Journal of ACM*, 46(5), p. 720-748.
- Butenhof D.R., 1997. *Programming with POSIX threads*. Addison-Wesley.
- Brenner L. and P. Fernandes and B. Plateau and I. Sbeity, 2007. PEPS 2007. *International Conference on Quantitative Evaluation of Systems (QEST 2007)*. p. 163-164.
- Daoudi E.M. and T. Gautier and A. Kerfali and R. Revire and J.-L. Roch, 2005. Algorithmes parallèles à grain adaptatif et applications. *Technique et Science Informatiques*. 24(5), p. 505-524.
- Fernandes P., 1998. Méthodes numériques pour la solution de systèmes markoviens à grand espace d'états. PhD thesis, Institut National Polytechnique de Grenoble.
- Frigo M. and C.E. Leiserson and K.H. Randall, 1998. The implementation of the cilk-5 multithreaded language. *ACM SIGPLAN conference on programming language design and implementation*. 33(5), p. 212-223.
- Gautier T. and J.-L. Roch and F. Wagner, 2007a. Fine grain distributed implementation of a dataflow language with provable performances. *Computational science - ICCS*, 4488, p. 593-600.
- Gautier T. and X. Besseron and L. Pigeon, 2007b. KAAPI: A thread scheduling runtime system for dataflow computations on cluster of multiprocessors. *International Workshop on Parallel Symbolic Computation*, p. 15-23.
- Hendler D. and N. Shavit, 2002. Non-blocking steal-half work queues. *Symposium on Principles of Distributed Computing (PODC'02)*, p.280-289.
- Hopcroft J.E. and J.D. Ullman, 1979. *Introduction to automata theory, languages and computation*. Addison-Wesley.
- Plateau B., 1984. L'évaluation du parallélisme et de la synchronisation. PhD thesis, Paris-Sud Orsay.
- Plateau B. and J.-M. Fourneau, 1991. A methodology for solving Markov models of parallel systems. *Journal of Parallel Distributed Computation*, 12(4), p. 370-387.
- Roch J.-L.. On the number of steal operations in a greedy schedule. Technical report, personal Communication.
- Tchiboukdjian M. and V. and Danjean and T. Gautier and F. Mentec and B. Raffin, 2011. A Work Stealing Scheduler for Parallel Loops on Shared Cache Multicores. *Euro-Par 2010*, LNCS 6586, p. 99-107.
- Tchiboukdjian M. and N. Gast and D. Trystam and J.-L. Roch and J. Bernard, 2010. A tighter analysis of work stealing. *Algorithms and Computation*, LNCS 6507, p. 291-302.
- Van Nieuwpoort R. and J. Maassen and T. Kielmann and H.E. Bal, 2005. Satin: Simple and efficient java-based grid programming. *Journal for Parallel and Distributed Computing*, 6(3), p/ 19-32.