**UNIVERSITÉ DE SHERBROOKE**

**DÉPARTEMENT D’INFORMATIQUE**

**IFT630 – Processus concurrents et parallélisme**

**Parallélisation du problème des N-reines**

Rapport présenté à

*Gabriel Girard*

Par

*Mathieu Gagnon, 10116135*

*Maël Valma, 10021923*

Le 26 avril 2013

Table des matières

[1. Introduction 3](#_Toc354769753)

[1.1. Mise en contexte 3](#_Toc354769754)

[1.2. Utilisation de notre programme 4](#_Toc354769755)

[2. Résultats 4](#_Toc354769756)

[Algorithme séquentiel 4](#_Toc354769757)

[Algorithme parallèle 5](#_Toc354769758)

[3. Analyse 5](#_Toc354769759)

[3.1. Problèmes rencontrés 5](#_Toc354769760)

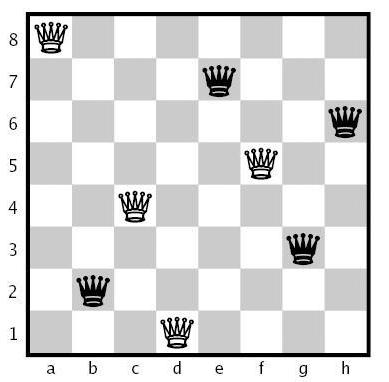
[3.2. Analyse des résultats 5](#_Toc354769761)

[4. Conclusion 6](#_Toc354769763)

# Introduction

## Mise en contexte

Le but du problème des N reines (mieux connu sous le nom de « problème des huit dames » en français) est de placer huit reines d’un jeu d’échecs sur un échiquier de N x N cases de telle sorte que les dames ne puissent se menacer en un coup, selon les règles traditionnelles du jeu d’échecs (en ignorant la couleur des pièces).



Pour la résolution du problème avec 8 reines, il est assez simple de trouver les solutions avec un algorithme séquentiel. Cela ne prend que quelques secondes. C’est pourquoi nous avons décidé de généraliser le problème à un nombre quelconque de reines sur un échiquier de taille plus grande, mais toujours de même taille que le nombre de reines. S’il est trivial pour un processeur de trouver toutes les solutions avec 8 reines, il en est moindre avec 16 reines, par exemple. C’est ici que la parallélisation de l’algorithme entre en jeu pour accélérer le traitement du calcul.

Pour la conception de la version parallèle des n reines nous avons utilisés les threads POSIX sur le serveur Solaris.

**Première solution**

Nous avons conçu une version parallèle qui divise les tâches en N threads (un thread pour chaque case de la 1ère ligne). Chaque thread démarre un arbre DFS (Depth First Search) afin de rechercher les solutions avec la racine de l’arbre correspondant à la case de départ.

**Deuxième solution**

Une autre approche que nous avons essayée est l’utilisation d’un thread par case recherchée. Cette version accélère le calcul des solutions au problème mais elle requiert un très grand nombre de processeurs. Comme nous sommes limités en nombre de cœurs sur le serveur Solaris, cette approche n’est pas privilégiée mais pourrait être envisagée ultérieurement. Nous avons tenté de concevoir une solution basée sur cette deuxième approche, mais nous nous sommes rendu compte que celle-ci retournait les solutions attendus ainsi que leurs rotations et réflexions. Bien qu’à l’exécution cette solution semble très rapide, elle devrait retourner uniquement les solutions uniques. Cette deuxième solution ne fonctionne qu’avec 8 reines maximum à cause du grand nombre de threads utilisés.

Nous nous sommes donc concentrés sur la première approche.

Pour information, la plus grande taille calculée à ce jour, pour le problème des N reines, est sur un échiquier 25x25, elle a mis 6 mois à être calculée sur pas moins de 260 machines. Le nombre de solutions trouvée est de 2,207,893,435,808,352.

(Source : <http://www.game-corp.net/topic-951-plateforme-de-calcul-le-probleme-des-n-reines.html>)

**Comparaison de la version séquentielle et la version parallèle**

Pour cette comparaison, nous avons pris en compte le temps de calcul de l’algorithme et non le temps que la console prend pour afficher les solutions. La fonction chargée d’afficher les résultats a donc été mise en commentaire dans le code et les solutions ne seront donc pas affichées à l’écran afin de procéder aux tests de comparaison.

Afin de vérifier l’exactitude des résultats des solutions proposées, il faut décomenter la ligne portant la mention ‘’ //DECOMENTER PR AFFICHER LES RESULTATS’’. (Ligne 82 pour nqueens\_sequentiel.cc

et ligne 103 et 124 pour nqueens\_parallel1.cc)

## Utilisation de notre programme

Pour utiliser notre programme dans son état actuel, il faut nous connecter par SSH sur le serveur Tarin de l’université de Sherbrooke (un serveur Solaris) et télécharger en amont (*upload*) notre programme sur notre espace de disque dur sur Tarin. Ensuite, il faut procéder à la compilation du programme. On fait ainsi :

g++ -o nqueensP nqueens\_parallel1.cc

ou

g++ -o nqueensP nqueens\_parallel2.cc

Ensuite, il nous faut exécuter le programme :

./nqueensP

Nous fournissons aussi la version séquentielle de notre programme, pour des fins de tests et de comparaison. Vous devez le compiler en appelant la commande :

g++ -o nqueensS nqueens\_sequentiel.cc

Puis, l’exécuter en faisant :

./nqueensS

# Résultats

### Algorithme séquentiel

|  |  |
| --- | --- |
| Nombre de reines | Temps (sec) |
| 8 | ~0 |
| 9 | ~0 |
| 10 | ~0 |
| 11 | 3 |
| 12 | 20 |
| 13 | 141 |
| 14 | 1031 |

### Algorithme parallèle

|  |  |
| --- | --- |
| Nombre de reines | Temps (sec) |
| 8 | ~0 |
| 9 | ~0 |
| 10 | ~0 |
| 11 | 1 |
| 12 | 5 |
| 13 | 35 |
| 14 | 252 |

# Analyse

## Problèmes rencontrés

Nous avons eu beaucoup de trouble pour ce qui est de la prise des résultats. En effet, en utilisant la fonction clock() de <time.h>, nous obtenions des valeurs visiblement incorrectes. Par exemple, lors du calcul en parallèle avec 12 reines, notre programme prenait environ 5 secondes (en regardant sur une montre) pour s’exécuter complètement tandis que notre calcul de temps affichait environ 19 secondes. Après plusieurs tentatives et un peu de recherche, nous avons décidé d’utiliser la fonction time(), qui retourne le nombre de secondes écoulées depuis le 1er janvier 1970 00:00 UTC. Cette fonction est beaucoup moins précise que clock() (précision à la seconde près au lieu de la milliseconde près), mais au moins elle donne des valeurs crédibles, semblables au WallClock Time.

De plus, nous nous sommes rendus compte que l’affichage à l’écran des multiples solutions de notre algorithme prend énormément de temps. En effet, pour un essai à 12 reines, il s’est écoulé plus de 4 minutes pour l’exécution de notre programme tandis qu’en omettant l’impression des solutions, notre programme s’exécutait en à peine 6 secondes.

## Analyse des résultats

En observant le tableau des résultats à la section 2, nous pouvons voir que l’algorithme exécuté en parallèle complète le calcul en moins de temps que la version séquentielle. Cette accélération est de plus en plus notable lorsqu’on augmente le nombre de reines.

Nous pouvons ainsi en arriver à la conclusion que paralléliser la résolution du problème des N reines s’avère rentable pour un nombre de reines supérieur à 10.

# Conclusion

Pour conclure ce rapport, nous voulons rappeler que ce projet consistait à paralléliser la résolution du problème des N reines pour ainsi accélérer le calcul des solutions. Notre programme n’est pas des plus optimisés, mais il nous permet tout de même d’accélérer substantiellement le calcul par rapport à la version séquentielle de l’algorithme. Ce projet nous a permis d’améliorer nos connaissances en conversion d’algorithmes existants pour le rendre parallélisable sur plusieurs processeurs et analyser les temps de traitement pour voir si c’est rentable d’utiliser une version multiprogrammée de l’algorithme.