Rapport projet 2: Factorisation de nombres

Iserentant Merlin - Momin Charles

May 5, 2015

Ce rapport comporte une analyse détaillée de l'architecture et du fonctionnement de notre programme ainsi qu'un analyse paramétrique des perfomances de ce dernier.

Architecture globale du programme

Notre programme est divisé en 4 phases principales distinctes:

- Chargement des nombres: cette phase se compose d'une analyse des paramètres du programme suivie d'une phase de chargement des nombre à proprement parlée.)
- Factorisation des nombres: comme son nom l'indique, cette étape sert à factoriser les nombres obtenue grâce à la phase précédente en facteur premier.
- Traitement des facteurs premiers: chaque facteur produit par la factorisation et traité. Un compte de chacun des facteurs est tenu.
- Analyse des résultats: le facteur unique trouvé est cherché parmis les différents compteurs. Si plusieurs facteurs unique sont trouvés, une erreur est envoyée.

Ces trois étapes se déroulent de manière simultanée grâce à l'utilisation de threads selon un double schéma de producteur/consommateur. Les chargements de nombres se font à raison d'un thread par type d'entrée de nombre. Plus précisément, chacun de ces threads itère sur un tableau contenant les différents noms de fichier ou adresses URL. Il y a donc trois threads de chargement de nombre. La taille des buffers pour les deux schéma est de N. Ce choix n'est pas anodin, en effet il permet à N threads de factorisations fonctionner en simultanée. Le traitement des nombres se fait grâce à CHECK ICI LE NOMBRE DE THREADS un thread parcourant le second buffer qui met à jours les comptes des différents facteurs premiers trouvés. Une fois ces étapes faites, le programme principale analyse les compteurs obtenus.

Mécanisme de synchronisation

Les différents échanges de données entre threads se font grâce à deux schémas de producteur/consommateur

- Premier schéma: [Producteur]=Threads de chargement de nombres ⇒ sortie:nombres sur 64bits; [Consommateurs]=Threads de factorisation
- Second schéma: [Producteur]=Threads de factorisation ⇒ sortie:nombres premiers; [Consommateurs]=Threads d'analyse de nombres premiers

L'implémentation de ces schémas est réalisée grâce à l'utilisation de sémaphores, permettant ainsi l'accès aux buffers.

Les buffers sont des tableaux de taille N. Leurs accès est protégé en plus des sémaphores par des mutex qui évitent tout problème de synchronisation entre threads lors de l'écriture de données.

Structure de données

Les structures utilisées sont les suivantes:

- Premier schéma de producteur/consommateur: tableau de nombres de dimension [2xN]. La première dimension contient le nombre et la deuxième une référence vers le fichier d'origine
- Second schéma de producteur/consommateur: tableau de nombres de dimension [3xN]. La première dimension contient le nombre, la seconde un paramètre utile à l'algorithme de factorisation et la troisième une référence vers le fichier d'origine.
- Structure de nombre premier: structure contenant un nombre premier, le compte de celui-ci ainsi qu'une référence vers le fichier dans lequel la dernière itération de ce nombre a été trouvée.
- Liste chainée de structure de nombre premier: Liste des nombres premiers et leurs comptes. Elle est mise à jour si nécessaire.

Algorithmes principaux

Algorithme 1: génération de nombres premier

Pour tester si un nombre est premier, notre programme teste s'il est divisible par tous les nombres premiers inférieurs à sa racine carré (ceux-ci sont déjà calculés et stockés dans la liste chainée). Si ce n'est pas le cas, il s'agit bien d'un nombre premier.

Algorithme 2: Factorisation en produit de nombre premier

On procède par itération. Notre programme test le nombre afin de voir si il est divisible par le plus petit nombre premier (2). Si c'est le cas, on considère que ce nombre premier est facteur et on réitère le test sur le résultat de la division ainsi obtenu. Si le test venait à échouer, le test de division s'effectue avec le nombre premier suivant (si la liste chainée n'indique pas de nombre premier suivant, celui-ci est calculé à l'aide de l'algorithme 1 et rajouté à la liste chainée). Les tests de division s'arrête lorsque le nombre est devenu égale à l'unité.

Analyses de performances

Plusieurs tests de perfomance ont été effectués afin d'analyser la rapidité de l'implémentation de notre programme. Ceux-ci consistaient à changer le nombre de threads de factorisation pour des analyses de fichier de différentes tailles. Ces tests ont été réalisés avec une implémentation qui répondait le plus à notre idée initiale. Les résultats obtenus sont repris ici:

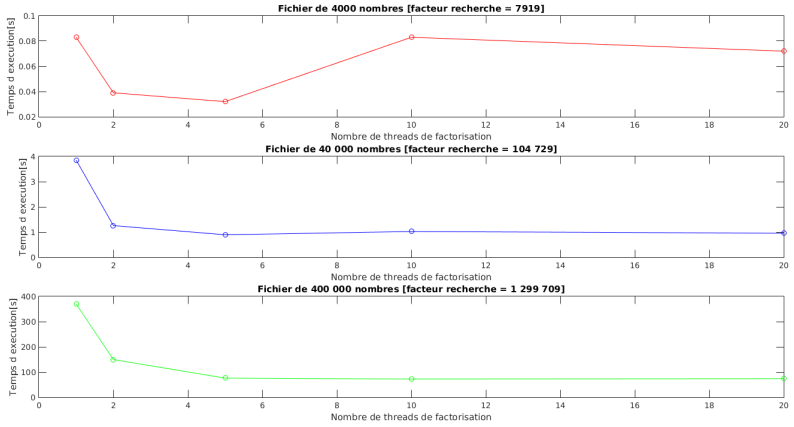

Figure 1: Analyse de l'évolution du nombre de thread

Au vue des résultats obtenus, nous en avons déduit qu'il y avait un nombre de threads optimal pour chaque analyse. Ce nombre dépend de la taille du fichier ainsi que la grandeur des nombres analysés. Lorsque le nombre de thread est trop important, ceux-ci se gênent mutuellement lors de l'accès aux différents buffers. EXPLICATION. Nous avons alors effectué au analyse en fonction du nombre de threads consommateurs du second schéma:

BLABLABLA. Après avoir adapté notre programme par rapport au résultats trouvés, nous avons es mesures différentes:

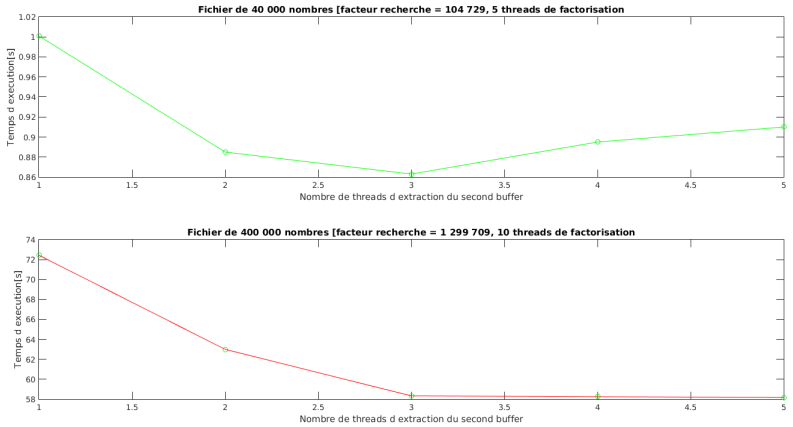

Figure 2: Analyse de l'évolution du nombre de thread

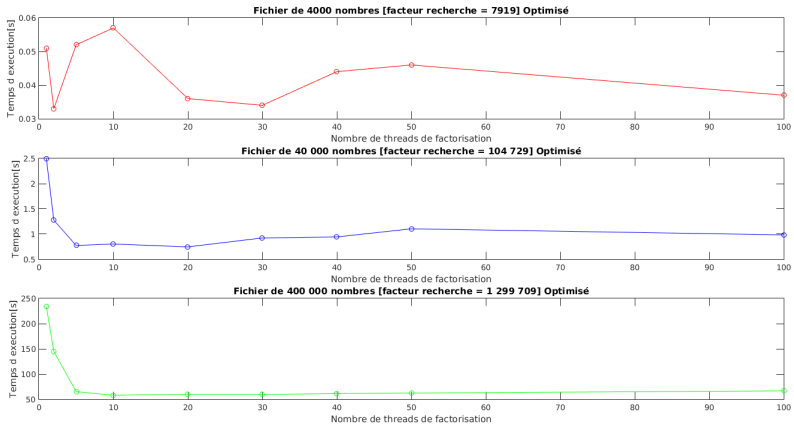

Figure 3: Analyse de l'évolution du nombre de thread