IFT630
Processus concurrents et parallélisme
Projet

Application de compression de fichiers

Fait par
Michael Brunelle – 06 777 296
Mathieu Tourigny – 06 805 978

Présenté à
Gabriel Girard

Université de Sherbrooke
24 avril 2008

Présentation du projet

Notre projet consiste à un programme simple de compression de fichier. L'algorithme utilisé est celui de Huffman et son taux de compression en moyenne de 30% et il est sans perte.

Au départ, le programme était séquentiel, nous avons parallélisé la compression et la décompression tout en gérant la synchronisation avec des sémaphores.

Toutefois, pour paralléliser la compression il a fallu sacrifier un peu de performance sur la compression. Car, il manquait des informations supplémentaires pour réussir à faire le traitement parallèle sans que les processus se marchent sur les pieds.

Présentation de l'algorithme de Huffman.

L'algorithme de Huffman a été utilisé pour la compression. Il est simple à comprendre et est souvent utilisé, donc répandu. Pour sauver du temps, nous avons choisi une implémentation déjà codée. Nous avons pu nous concentrer sur l'implémentation de la parallélisation.

Le principe de l'algorithme de Huffman est qu'il compte le nombre de fois que se trouve chaque octet dans le fichier. Il construit ensuite un arbre en formant des nœuds avec des octets qui ont le même nombre d'occurrences. Le principe est que les octets qui se répètent le plus souvent se retrouvent dans le haut de l'arbre et ceux moins utilisés dans le bas. Ensuite, il construit un code binaire selon le chemin qu'il emprunte pour trouver un octet. Ex : un octet se trouve à gauche droite droit son code sera 011. Ensuite nous remplaçons chaque octet par son code trouvé dans l'arbre.

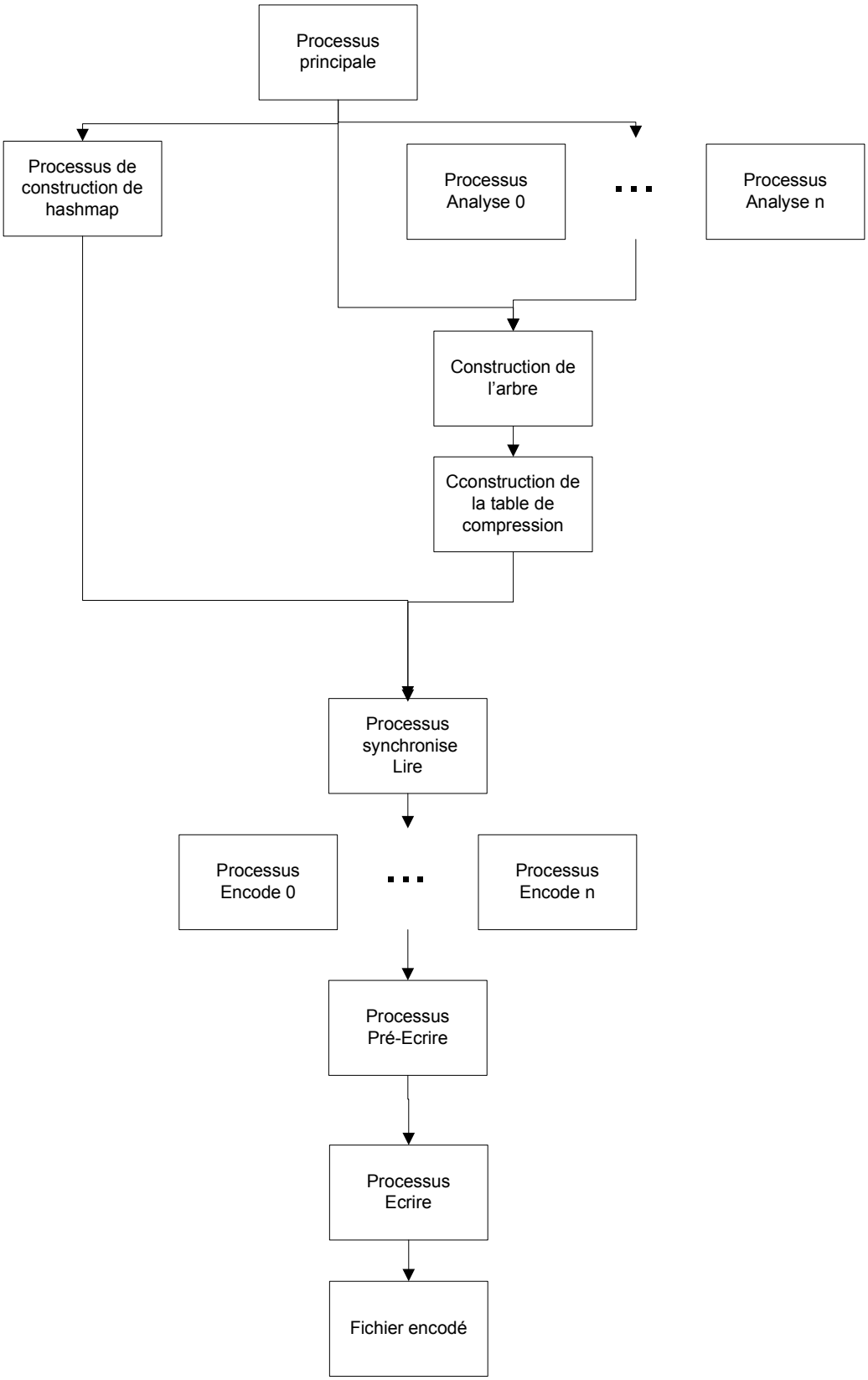
En principe, à moins que l'on retrouve les 256 octets possibles et seulement 1 fois, le fichier devrait être plus petit qu'au départ. (il faut considérer l'entête du fichier)

Voici la source de l'algorithme de Huffman :
http://www.developer.com/java/other/article.php/10936_3603066_3
Voici un texte qui explique le principe de l'algorithme de Huffman:
http://tcharles.developpez.com/Huffman/

Encodage

L’encodage se décompose en 2 étapes principales : l’analyse puis l’encodage. Pour accélérer, la construction de bitmap qui servira lors de l’encodage est démarrée en même temps que l’analyse du fichier se fait. Plusieurs analyseurs sont parti ayant chacun leur bout de fichier a analysé, mais toute la même table de comptage d’occurrence. Une fois cette étape finit(le processus principal attend que tous les processus enfant aient fini), la deuxième étape est lancée.

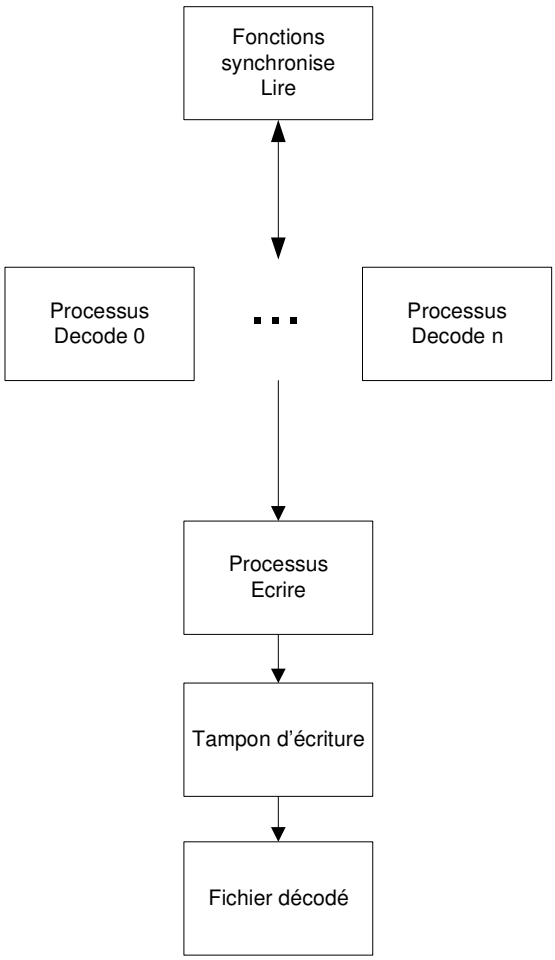
L’étape d’encodage est très complexe puisqu’elle imite un peu le comportement d’un pipeline de Unix. Pendant que la lecture s’effectue, une étape d’encodage est faite et les étapes pour l’écriture sont aussi faites. D'abord, un lecteur envoie à tour de rôle une partit du fichier à compresser à des processus numérotés de 0 à n processus (n’est passé en paramètre à l’application). À leur tour, ceux-ci écrivent les données encodées vers le préécrivain qui ne fait que la conversion de chaine de bit en octet. Ensuite, le préécrivain envoie les données à écrire à l’écrivain qui se charge de faire les sorties appropriées telles que l’entête du fichier et le pied du fichier contenant les « seek » et « offset » requis pour la décompression.


Processus principale
Processus de construction de hashmap
Processus Analyse 0
Processus Analyse n
Construction de l'arbre
Cconstruction de la table de compression
Processus synchronise Lire
Processus Encode 0
Processus Encode n
Processus Pré-Ecrire
Processus Ecrire
Fichier encodé

Décodage

Le décodage est composé de plusieurs processus qui vont chercher des données en utilisant une fonction synchronisée de lecture. Les données une fois traitées sont acheminées à un processus qui gère les écritures dans le fichier de sortie. Lorsqu'un processus n'a plus rien à lire, il termine.

Les actions qui doivent être traitées par le processus qui écrit sont triées en fonction du rang du processus qui a créé l'action, ceci permet d'écrire de dans l'ordre les données dans le fichier de sortie. Les actions sont traitées quand chaque processus en a créé une. Les sorties générées ne sont pas automatiquement écrites dans un fichier, mais plutôt dans un tampon pour éviter de faire des accès disque pour une poignée d'octets.



Le parallélisme est rendu possible grâce aux calculs de frontière durant la compression, il y a le « seek » et un « offset » qui sont respectivement des déplacements en octets et en bit. Chaque processus peut alors traiter son bout de fichier tout en gardant une cohérence des données.

Structure d'un fichier

entête

nomFichier : Nom du fichier original

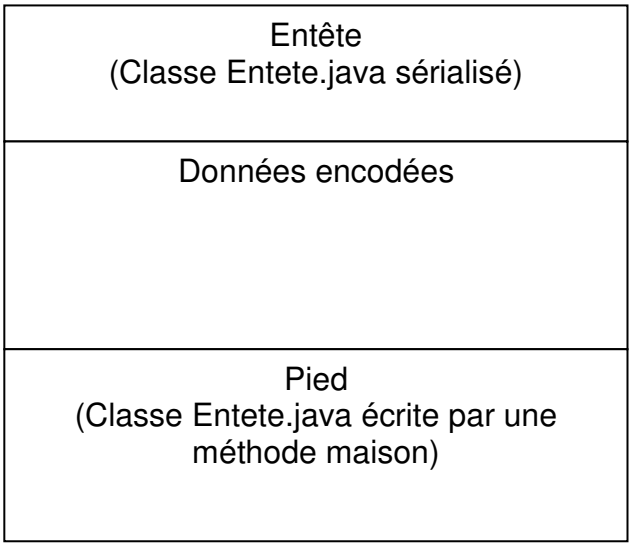
grosseur : Grosseur du fichier originale

grosseur du pied : Grosseur du pied pour pouvoir le retrouver

table : table de Huffman (table de conversion des caractères

Pied :

Contiens quelques frontières pour trouver des caractères; sers pour la décompression.



Compilation du programme

Pour compiler, il suffit de créer un projet avec les sources du projet. Les classes qu'il faut exécuter se nomment Main et se trouvent dans le package séquentiel ou parallèle selon que vous voulez exécuter le programme de façon parallèle ou séquentiel.

Utilisation du programme

Pour utiliser le programme, il faut passer les options suivantes :

-C ou –D "chemin du fichier" :

-C pour la compression et –D pour la décompression du fichier. La compression de répertoire n'est pas implémentée.

-P n:

Seulement pour la compression parallèle, l'option –P peut être signalée pour spécifier le nombre de processus à utiliser.

Note : il peut arriver que le programme manque de mémoire (Exception in thread "main" java.lang.OutOfMemoryError: Java heap space). Dans ce cas, il faut spécifier l'option –Xmx1g à la machine virtuelle de java. Dans Eclipse, la place pour spécifier de telle option se trouve en dessous de la place où inscrire les paramètres pour le programme.

Conclusion

Par rapport au même programme de compression et décompression, la version parallèle utilise entièrement les processeurs et termine son traitement pratiquement deux fois plus rapidement.

Si l'on compare notre programme de compression et décompression de fichiers par rapport aux utilitaires de compressions les plus populaires, on est plus lent et le taux de compression est moins bon de beaucoup. On aurait eu davantage de succès pour la compression et la vitesse si nous avions utilisé des « patterns » pour ne pas devoir traiter le fichier bit par bit.

Certaines améliorations peuvent être apportées pour augmenter les performances, par exemple on pourrait avoir une taille de cache, nombre de processus et la quantité de frontières calculées automatiquement en fonction de la grosseur du fichier. De plus, l'entête et le pied du fichier pourraient être allégés en utilisant des types de données restreints à nos besoins, par exemple utiliser un entier court de 2 octets au lieu d'un de 4 octets.

Pour conclure, il faudrait un peu plus que quelques jours de travail pour arriver à des performances comparables à ceux des logiciels de compression populaires.