Rapport de TP 4MMAOD : Génération de patch optimal

CASTRO LOPEZ Walter Ferney$_1$ (groupe 7$_1$)
BUALO GIORDANO Lucas Santiago$_2$ (groupe 7$_2$)

28 novembre 2015

1 Principe de notre programme

Nous avons implémenté notre programme avec le méthode récursive. Pour cela, nous avons créé un tableau de dimension nXm où n est le nombre de lignes du fichier original (d'entré) et m est le nombre de lignes du fichier resultat (de sorti) ; et dans lequelle nous gardons les valeurs des coûts optimals pour calculer les patchs qui transforment le fichiers A_i en B_j, cet-à-dire, le fichier qui va la ligne 1 jusqu'à la ligne i du fichier d'entré, et le fichier qui va de la ligne 1 jusqu'à la ligne j du fichier de sorti respectivement.

Autre que le coût, on mémorise pour chaque patch optimal la dernière instruction qui a été éfectué, permettant de retrouver le patch optimal une fois on a déjà le coût du patch optimal (le tableau de coût rempli).

Finalement, pour optimiser le temps et limiter les entrées/sorties vers les fichiers, nous gardons dans notre code une copie de chaque fichiers au début, faisant plus facile l'accès à l'information de chaque fichier.

2 Analyse du coût théorique

Justification : Nous allons calculer le coût de chaque fonction dans notre code, tenant en compte que il y en a quelq'unes qui s'appellent entre elles. Le patch comprend une première partie dans laquelle tous les variables grobales qui ont besoin sont initialisés. Et après avec l'appele de la fonction le chemin de coût minimum est calculé. On ne decrira pas les instruction dont le cout n'est pas significative ou celles avec un cout fixe.

2.1 Nombre d'opérations en pire cas :

Justification :

— Décompte de lignes : n + m
Allocation des arrays de 1 seule dimension qui represent des matrix
— F1 - Pour les lignes du "file source" (textes)
— F2 - Pour les lignes du "file target" (textes)
— instruction - Pour les lignes du differents operation (textes)
— optimal - Pour les lignes du differents coûts de operation (valeurs)
— patch - Pour les lignes du chemin optimal (textes)
Asignation des contenues
— F1 : n_1 (les lignes sont copiés du fichier source au tableau)
— F2 : n_2 (les lignes sont copiés du fichier target au tableau)
— optimal : $(n_1 + 1) * (n_2 + 1)$ (les couts sont initaliser en infinito)
Pire Cas de la fonction avec $i = n_1$ et $j = n_2$
— DelMul – Allocation d'un tableau pour stocker et analiser après le cas destruction multiple
— Un cycle for de taille n_1 pour charger le tableau DelMax
— Deux appele de la fonction findPatch de manière recursive, qui aura un cout de $(n_1 + 1) * (n_2 + 1)$
— Generation d'un tableau pour stocker les strings des instructions du chemin optimal : $n_1 + n_2$

Si on analyse le cout d'obtenir le chemin optimal, on observe qu'il dèpend surtout des appels recursives, c'est-à-dire que le cout théorique de l'algorythme est d'ordre $O(n_1 * n_2)$.

2.2 Place mémoire requise :

Justification : On va exprimer une approximation de la place en memoire requise, en regardant que un char et un int, les deux occupent un octet :

— F1 : n_1* LINE_ TAILLE_ MAX octets.

— F2 : n_2* LINE_ TAILLE_ MAX octets.
— optimal : $(n_1 + 1) * (n_2 + 1)$ octets.
— instruction : $(n_1 + 1) * (n_2 + 1)*$LINE_ MAX_ SIZE octets.
— patch : $(n_1 + n_2)*$LINE_ MAX_ SIZE octets.

Mais après les differentes appel de malloc pour obtenir les tableaux DelMul ce fait chaque fois jusqu'à l'index de l'appel, cela ajoute un cout de memoire de $n_1!$ octets.

2.3 Nombre de défauts de cache sur le modèle CO :

Justification : Nous vu que les défauts de cache. Nous croyons que cela est parce que nous utilisons le méthode récursive et cela posse un problème d'allocation dans la mémoire cache. Chaque fois que le code a besoin d'un tableau, il faut le charger dans la mémoire caché, et comme la mémoire caché n'est pas très grande, il faut l'actualiser plusieur fois.

Si l'on voit le cas del'allocation des fichiers dans ses tableaux, on voit la cause la plus grande des défauts de cache. Tout d'abord, l'ordinateur utilisé pour les test a une mémoire cache de 4 Mo. Nous suposons que pour la mémoire, le nombre de lignes est le même que la longeur de la ligne dedans. Dans ce cas, le nombre de lignes serait environ 1700 et la taille de chaque ligne serait d'environ 1700 octets. De plus, les lignes bien du fichier d'entré ou bien du fichier de sorti fait au maximum 100 octets selon notre modèle. En fin de compte, on utilise une ligne de 1700 octets juste pour alloquer une ligne de 100 octets, ce qui n'est pas du tout optimal, génerant plus des défauts de cache.

3 Compte rendu d'expérimentation

3.1 Conditions expérimentaless

3.1.1 Description synthétique de la machine :

Pour les test, on a laisé l'ordinateur sans autres exécutions pour réduire le temps d'exécution. L'ordinateur à les spécifications suivantes :

OS : Ubuntu 14.04.2 LTS
CPU : Intel(R) Core(TM) i7-3537U CPU @ 2.00GHz
GPU : NVIDIA GF117M [GeForce 610M/710M/820M / GT 620M/625M/630M/720M]
RAM : 5855 MiB
CACHE : 4096 KB

Information trouvé avec les commandes :
cat /proc/cpuinfo
lshw
lsb_ release - a

3.1.2 Méthode utilisée pour les mesures de temps :

Pour le temps, nous avons utilisé l'outil *temps* d'ubunu, en disant :

3.2 Mesures expérimentales

Il faut dire que nous n'avons pas pu faire l'exécution du benchmark 6, parce qu'il y avait un error pour la taille du matrice DelMax, génerant une exeption dans l'éxecution.

3.3 Analyse des résultats expérimentaux

À la fin, les résultats sont un peut trop grandes, tenant en compte le nombre de défauts de cache que l'on fait. C'était impossible de trouver un moyen de comparer expérimentalement la théorie avec la pratique, parce que la matrice DelMul fait une appel a malloc, et donc le temps d'exécution n'ont pas permis réaliser la preuve.

	coût du patch	temps min	temps max	temps moyen
benchmark1	2486	5.989	7.342	6.634s
benchmark2	3093	58.572s	1m7.256s	1m1.299s
benchmark3	786	3m0.393s	3m16.467s	3m6.636
benchmark4	1638	6m.52.723s	7m39.563s	7m24.734s
benchmark5	7376	17m28.635s	18m18.974s	17m51.708s

FIGURE 1 – Mesures des temps minimum, maximum et moyen de 5 exécutions pour les 5 benchmarks.

4 Question : et si le coût d'un patch était sa taille en octets ?

Pour ce cas, il faudrait changer les coûts de chaque opération, et les relattioner avec les octets du patch :

En cas d'une addition, la première ligne qui a un char avec le "+" et le numéro de la ligne. Et la deuxième ligne s'agit de la ligne à ajoute, ça veut dire (2 + la quantité de chiffre du nombre de caractères à ajouter). Son coût est le nombre de caractères de cette ligne.

Le cas de la substitution c'est similaire.

Pour une supression multiple le coût est similaire me il faut tenir en compte les chiffres des numéros de la ligne d'origin et de la quantité à supprimer.

En cas d'une supression simple, il y a un coût de 2 + les chiffres du numéro de la ligne.

Finalement, la copie n'a pas de coût parce qu'il n'ajoute pas des lignes au patch.