Rapport de TP 4MMAOD : Génération d'ABR optimal

MASELBAS Jules (groupe 1)
TOURANGEAU Sarah-Jeanne (groupe 1)

13 avril 2017

Préambule

- *Compléter ce patron de rapport en supprimant toutes les phrases en italique : elles ne doivent pas apparaître dans le rapport pdf.*
- *Il sera attribué* **1 point** *pour la qualité globale du rapport : présentation, concision et clarté de l'argumentation.*

1 Principe de notre programme (1 point)

Nous avons choisi la méthode récursive. Un fois que le fichier est lu, et le vecteur de somme des probabilités calculé, voici les étapes de notre algorithme d'ordonnancement :
1- Si l'arbre n'a pas de feuille, on renvoie le poids de la racine
2- On calcule le poids de l'arbre dans le cas où l'on choisirait la première valeur (n) comme racine.
3- On calcule le poids de l'arbre dans les cas où l'on choisirsait les valeurs de n+1 à m-1 comme racine.
4- On calcule le poids de l'arbre dans le cas où l'on choisirait la dernière valeur (m) comme racine.
5- On choisit la racine qui nous donne le poids le plus faible, on le stocke et on renvoie ce poids.

Pour calculer le poids d'un arbre, on additionne le poids de la racine et des deux sous-arbres. Pour calculer le poids des sous-arbres, on vérifie si ce poids est stocké, et sinon on appelle la même fonction sur le sous-arbre.

2 Analyse du coût théorique (2 points)

2.1 Nombre d'opérations en pire cas :

Justification : On commence par calculer le vecteur de somme des propabilité. ce qui a un cout de n, puisqu'on boucle sur un vecteur n en faisant une addition à chaque fois.
On doit ensuite calculer le poids de n arbres (puisqu'il y a n noeuds). Le poids d'un arbre est un calcul prenant en compte deux valeurs dans le vecteur de somme des probabilités, et deux poids de sous-arbres. Les poids de ces sous-arbres seront calculé une première fois, puis stockés dans la matrice poids par la suite. Sans compter la récursion, on fait donc n fois une adition.
Pour la récursion, on va calculer le poids et la racine de $\frac{n*n}{2}$ sous-arbres. On a un arbre (l'arbre contenant les noeuds de 1 à n) pour lequel on doit tester n arbres. on a deux arbres (ceux contenant 2 à n et 1 à n-1) pour lequels on doit tester n-1 cas. Ce qui donne donc :

$$\sum_{k=1}^{n} k * (n + 1 - k)$$

Pour un total de

$$O(2n + \sum_{k=1}^{n} k * (n + 1 - k)) = O(2n + (n + 1)\sum_{k=1}^{n} k - \sum_{k=1}^{n} K^2)$$

$$= O(2n + \frac{(n + 1)^2(n)}{2} - \frac{n(n + 1)(2n + 1)}{6})$$

$$= O(\frac{n^3 + 3n^2 + 14n}{6}) = O(n^3)$$

2.2 Place mémoire requise :

Justification : On a trois matrices, la matrice des valeurs lues (taille n), la matrice des somme de probabilités (taille n) et la matrice des poids et racines ($\frac{n*n}{2}$ + $\frac{n*n}{2}$ = n*n). Pour un total de

$$n^2 + 2n$$

cases mémoire.

2.3 Nombre de défauts de cache sur le modèle CO :

Justification : Le nombre de défaut de cache n'est pas particulièrement optinisé dans ce programme. On peut donc supposer qu'on a un défaut de cache au maximum à chaque fois que l'on va récupérer une valeur dans un tableau. Pour trouver la racine, on a besoin d'aller chercher deux valeurs dans le vecteur de somme des probabilité, et n * 2 valeurs dans la matric de poids (au maximum). Puisqu'on doit calculer tous les sous-arbres, avant de revenir au calcul de la racine n, il est raisonnable de penser que les deux valeurs de vecteur somme des probabilité ne seront plus dans le cache. On a donc n*4 valeurs.

On a donc un arbre de n valeurs qui génère 4*n défaut de cache, 2 arbres de (n-1) valeurs qui génèrent 4*(n-1) défauts de cache chacun, pour un total de :

$$O(\sum_{k=1}^{n} 4 * (n + 1 - k)) = O(4n^2 + 4n - 4\sum_{k=1}^{n} k) = O(4n^2 + 4n - 4\frac{n(n+1)}{2}) =$$

$$O(2n^2 + 2n) = O(n^2)$$

3 Compte rendu d'expérimentation (2 points)

3.1 Conditions expérimentaless

3.1.1 Description synthétique de la machine :

Les tests ont été effectués sur les machines de l'ensimag. Aucun autres processus n'était en marche.
Processeur : Intel Core i3 CPU M370 @ 2.40 GHz x 4
Mémoire : 3.6 GiB
Système d'exploition : Linux CentOS

3.1.2 Méthode utilisée pour les mesures de temps :

Les 5 exécutions ont été effectuées les unes après les autres. Nous avons commencé par les 5 exécutions de benchmark1, puis de benchmark2, etc.

3.2 Mesures expérimentales

	coût du patch	temps min (s)	temps max (s)	temps moyen (s)
benchmark1		0.003	0.009	0.0052
benchmark2		0.002	0.004	0.0034
benchmark3		3.215	3.392	3.3188
benchmark4		28.220	28.538	28.3478
benchmark5		111.837	116.058	114.1602
benchmark6		583.154	596.717	587.564

FIGURE 1 – Mesures des temps minimum, maximum et moyen de 5 exécutions pour les 6 benchmarks.

3.3 Analyse des résultats expérimentaux

Les temps correspondent à peu près à notre analyse. On considère que le nombre d'opération est ce qui coutera le plus cher, et qu'il sera significativement plus cher que les autres, étant en n^3. Lorsqu'on cherche la courbe en x^3 qui s'approche le plus de nos données, on trouve

$$f(n) \approx 3 * 10^{-9} * n^3$$

Et des valeurs théoriques de

$$f(1000) = 3$$
$$f(2000) = 24$$
$$f(3000) = 81$$
$$f(5000) = 375$$

Ce qui varie à peu près comme nos résultats.