SEREIR Tariq, ETHMANE Mohamed Lemine

Questionnaire TP AOD

Préambule 1 point. Pourquoi le programme récursif avec mémoisation fourni génère-t-il une erreur d'exécution sur test 5 ?

On a dépasser la limite d'utilisation de la pile (trop d'appels récursifs même avec l'élimination des appels redondants obtenue grâce à la mémoîsation) avec une trop grande allocation de la mémoire dynamique.

1 Programme itératif (4 points)

La fonction EditDIstance_NW_PD qui est dans le fichier Needllemann-Wunsch-PD.c utilise la programmation dynamique, avec un coût linéaire en espace (on a changé la matrice memo en un tableau de dimension 1). A l'entrée du programme, le tableau memo de taille N contient la ligne M, ou N est la plus petite dimension. A l'itération numéro *i* le tableau memo contient la ligne M-i, donc pour l'élément qui était dans la case memo[*i*][*j*] qui dépendait de memo[*i+1*] [*j*], memo[*i*] [*j+1*] et memo[*i+1*] [*j+1*], il devient l'élément memo[*j*] qui dépend seulement de memo [*j+1*], memo[*j*] de la ligne $M-i+1$, (et la variable res qui stocke a chaque fois memo[j+1] pour la ligne $M-i+1$.

1. place mémoire allouée (ne pas compter les 2 séquences *X* et *Y* en mémoire via mmap) :

   Le coût en espace sans les 2 séquences X et Y est : *Θ*(N), il est minimal.

2. travail (nombre d'opération) :

   Le nombre d'opérations (coût arithmétique) est : *Θ*(M*N). Il est minimal par rapport a la formule récursive.

3. nombre de défauts de cache obligatoires (sur modèle CO, y compris sur *X* et *Y*):

   Le nombre de défauts de cache obligatoires sur modèle CO est :

   - si le cache est très grand (si le programme s'exécute en cache) :

   $$Q(N,M,L,Z)=\frac{N}{L}+\frac{N}{L}+\frac{M}{L}+\Theta(1)=\frac{2N+M}{L}+\Theta(1)$$

   explications : N/L pour Y, N/L pour memo, et M/L pour X.

4. nombre de défauts de cache si $Z << \min(N, M)$:

   - si Z est très petit ($Z << \min(N, M)$) alors

   $$Q(N,M,L,Z)=\frac{2N}{L}+\frac{M}{L}+\frac{M*2N}{L}+\Theta(1)=\frac{2NM+M+2N}{L}+\Theta(1)$$

2 Programme cache aware (4 points)

On va utiliser 2 tableaux memo et strip pour stocker respectivement une ligne et une colonne courante. Puis on a utiliser la méthode de Blocking pour calculer à chaque fois un block de taille K qui tient dans le cache. Pour minimiser les défauts de cache il faut que : 2K1 + 2K2 + Θ(1) soit inférieur ou égal à Z. (ou K1 / L est égal au nombre de défauts de cache pour Y et memo et (K2 / L) le nombre de défauts de cache pour strip.

Afin de minimiser le nombre de défauts de cache on a choisi $K1=K2=\frac{Z}{4}$

1. place mémoire (ne pas compter les 2 séquences initiales *X* et *Y* en mémoire via mmap) :

   Le cout en espace sans les séquences X et Y est un $\Theta(N+M)$

SEREIR Tariq, ETHMANE Mohamed Lemine

2. travail (nombre d'opérations) :

   Le nombre d'opérations (coût arithmétique) est : La génération d'un bloc prend Θ(K1 * K2) donc au total on aura

   $$\left(\frac{N}{K1}\right) * \left(\frac{M}{K2}\right) * K1 * K2 + \Theta(1) = \Theta(M * N)$$

3. nombre de défauts de cache obligatoires (sur modèle CO, y compris sur *X* et *Y*):

   Si le cache est très grand (si le programme s'exécute en cache) :

   $$Q(N, M, L, Z) = \frac{N}{L} + \frac{N}{L} + \frac{M}{L} + \frac{M}{L} + \Theta(1) = \frac{2N + 2M}{L} + \Theta(1)$$

4. nombre de défauts de cache si *Z*< < min(*N, M*) :

   Le nombre de défauts de cache est de $\left(\frac{N}{K1}\right) * \left(\frac{M}{K2}\right) * \left(\frac{2K1 + 2K2}{L}\right) + \Theta(1) = \Theta\left(\frac{M * N}{Z * L}\right)$

   explications : $2K1/L$ **pour Y et memo, et** $2K2/L$ **pour X et strip.**

3 Programme cache oblivious (2 points)

On découpe récursivement jusqu'à un seuil S. Puis on utilise la fonction EditDistance_NW_CalculBloc.

1. place mémoire (ne pas compter les 2 séquences initiales *X* et *Y* en mémoire via mmap) :

   L'espace mémoire alloué est un Θ(M + N).

2. travail (nombre d'opérations) :

   Le surcoût de récursivité est un $\Theta\left(\frac{M * N}{S^2}\right)$

3. nombre de défauts de cache obligatoires (sur modèle CO, y compris sur *X* et *Y*):

   Si le programme s'exécute en cache, on est dans le même cas que pour le programme cache aware :

   $$Q(N, M, L, Z) = \frac{N}{L} + \frac{N}{L} + \frac{M}{L} + \frac{M}{L} + \Theta(1) = \frac{2N + 2M}{L} + \Theta(1)$$

4. nombre de défauts de cache si *Z*< < min(*N, M*) :

   Le nombre de défauts de cache au niveau i de la hiérarchie avec un cache LRU de taille Zi chargé par ligne de cache de taille Li est comme pour le cache aware : $\Theta\left(\frac{N * M}{Zi * Li}\right)$

4 Expérimentation (10 points)

Description:

-Processeur :Intel Core i5-10310u CPU 1.70 ghz, architecture X86_64
Caches : L1d : *128 KiB*, L1i : *128 Kib*, L2 : *1MiB*, L3 : 6 *Mib*
-Mémoire : 64-bits
- **Système : Linux**

4.1 (6 points)

Les paramètres du cache LL de second niveau sont:
D1 cache : 4096 B, 64B, 4-way associative
LL cache : 6291456 B, 64B, 12-way associative

		récursif mémo			itératif			cache aware			cache oblivious		
N	M	#Irefs	#Drefs	#D1miss	#Irefs	#Drefs	#D1miss	#Irefs	#Drefs	#D1miss	#Irefs	#Drefs	#D1miss
1000	1000	104, 706,321	51,057,337	277,673	87,542,586	44, 976,896	149, 340	93,585,037	49,003,090	150,675	102,567,447	53,754,704	8,377
2000	1000	208,444,347	101,305,513	550,443	174,122,418	89,148,320	293,404	186,199,873	97,196,515	296,068	204,820,149	107,283,743	11,153
4000	1000	417,337,709	203,207,657	1,096,248	348,683,520	178,892,786	581,548	372,830,975	194,984,981	586,866	409,314,564	214,357,555	16,390
2000	2000	417,378, 307	203,760,352	1,085,688	349,505,218	179,740, 557	575,546	373,695,748	195,862,801	578,496	409,467,379	214,568,162	16,547
4000	4000	1,667,238,604	814,299,875	4,291,647	1,397,295,354	718,808,765	2,269,783	1,493,887,959	783,193,058	2,292,504	1,636,588,551	857,377,406	46,169
6000	6000	3,749,737,706	1,831,622,566	9,649,947	3,143,572,297	1,617,265,122	5,088,120	3,360,778,980	1,762,051,465	5,147,144	3,810,778,312	2,002,655,630	88,261
8000	8000	835,517,536	407,404,298	2,183,886	698,197,756	358,773,706	1,157, 474	746,485,214	390,953,902	1,167,645	818,169,042	428,451,202	28,055

Analyse expérimetale: *commenter les mesures expérimentales par rapport aux coûts théoriques précédents. Quel algotithme se comporte le mieux avec valgrind et les paramètres proposés, pourquoi ?*

Pour commencer le programme itératif est plus performant que celui avec memoîsation ce qui est en accord avec la théorie. En théorie, le programme cache aware est censé avoir un nombre de défauts de cache plus faible que l'itératif, on peux voir que les valeurs sont très proches entre les 2 programmes (D1miss itératif et Aware), peut-être à cause de la différence entre la hiérarchie réelle (physique) et le modèle CO. L'algorithme qui se comporte le mieux avec valgrind est l'algorithme cache oblivious, car il ne dépend pas de la hiérarchie. Le nombre de défauts de cache de l'algorithme cache oblivious est optimal.

4.2 (3 points) Sans valgrind, par exécution de la commande :

On mesure le temps écoulé, le temps CPU et l'énergie consommée avec : time.

		itératif			*cache aware*			*cache oblivious*		
N	*M*	*temps cpu*	*temps écoulé*	*energie*	*temps cpu*	*temps écoulé*	*energie*	*temps cpu*	*temps écoulé*	*energie*
10000	*10000*	0m0.982s(user)	J0m0.983s (real)		Om0.994s	0m0.996s		0m0.972s(user)	J0m0.983s (real)	
20000	*20000*	0m3.994s (user)	0m3.996s (real)		Om4.39s	0m4.051		0m3.954s (user)	0m3.966s (real)	
30000	*30000*	0m8.957s (user)	0m9.967s (real)		0m9.127s	0.m9.137s		0m9.018s (user)	0m9.024s (real)	
40000	*40000*	0m15.865s (user)	0m15.878s (real)		0m16.085s(user)	0m16.094s		0m15.907s (user)	0m15.910s (real)	

NB : POUR L'ENERGIE SI JE FAIT COMME L'EXEMPLE MIS DANS LE FICHIER lisezMoi dans srcPerf y 'a à chaque fois une permission dinied pour le fichier energy_uj .

4.3 (1 point) Extrapolation: estimation de la durée et de l'énergie pour la commande :

A partir des résultats précédents, le programme cache oblivious *est le plus performant pour la commande ci dessus (test 5); les ressources pour l'exécution seraient:* (préciser la méthode de calcul utilisée)
COMME LE COÛT EST LINEAIRE ON PEUT ESTIMER LE TEMPS PAR PROPORTIONNALITÉ

- *Temps cpu (en s) : est de l'ordre de 0m1000.00 Energie (en kWh) : ... .*
- *Question subsidiaire: comment feriez-vous pour avoir un programme s'exécutant en moins de 1 minute ?*