Questionnaire TP AOD 2023-2024 à compléter et rendre sur teide

Binôme (Jennine Alaa – EL-khoundafi Badr : .....

1 Préambule 1 point

. Le programme récursif avec mémoisation fourni alloue une mémoire de taille $N.M$. Il génère une erreur d'exécution sur le test 5 (c-dessous) . Pourquoi ?
Il génère une erreur d'exécution car la mémoire allouée est très grande, on a un elimitation de mémoire , le programme ne trouve pas assez de mémoire pour allouer la matrice .

```
distanceEdition-recmemo    GCA_024498555.1_ASM2449855v1_genomic.fna 77328790 20236404   \
                           GCF_000001735.4_TAIR10.1_genomic.fna 30808129 19944517
```

Important. Dans toute la suite, on demande des programmes qui allouent un espace mémoire $O(N + M)$.

2 Programme itératif en espace mémoire $O(N + M)$ (5 points)

Expliquer très brièvement (2 à 5 lignes max) le principe de votre code, la mémoire utilisée, le sens de parcours des tableaux.

Ce programme parcourt les chaînes en remplissant un tableau de résultats intermédiaires (prev) en tenant compte des coûts d'insertion, de substitution, et de suppression pour aligner les caractères de chaque chaîne tout en gérant des cas spécifiques (bases inconnues ou caractères non valides).

Analyse du coût théorique de ce programme en fonction de N et M en notation $\Theta(...)$

1. place mémoire allouée (ne pas compter les 2 séquences X et Y en mémoire via `mmap`) : $\Theta(N)$
2. travail (nombre d'opérations) : $\Theta(M \cdot N)$, On effectue MxN opération dans les deux boucles.
3. nombre de défauts de cache obligatoires (sur modèle CO, y compris sur X et Y): $\Theta((M \cdot N)/L)$
4. nombre de défauts de cache si $Z \ll \min(N, M)$: $\Theta((M \cdot N)(N + M)/L)$

3 Programme cache aware (3 points)

Expliquer très brièvement (2 à 5 lignes max) le principe de votre code, la mémoire utilisée, le sens de parcours des tableaux.

On calcule la distance d'édition entre les chaînes X et Y en utilisant des blocs optimisés pour le cache, traités ligne par ligne en mémoire. L'algorithme parcourt les blocs en optimisant l'accès mémoire pour réduire les défauts capacitifs.

Analyse du coût théorique de ce programme en fonction de N et M en notation $\Theta(...)$)

1. place mémoire (ne pas compter les 2 séquences initiales X et Y en mémoire via `mmap`) : $\Theta(N)$
2. travail (nombre d'opérations) : $\Theta(M \cdot N)$, On effectue MxN opération dans les boucles comme dans le code itératif mais on divise la première boucle en blocs.
3. nombre de défauts de cache obligatoires (sur modèle CO, y compris sur X et Y): $\Theta((M + N)/L)$
4. nombre de défauts de cache si $Z \ll \min(N, M)$: $\Theta((M \cdot N)/L)$

4 Programme cache oblivious (3 points)

Expliquer très brièvement (2 à 5 lignes max) le principe de votre code, la mémoire utilisée, le sens de parcours des tableaux.

Le programme calcule la distance d'édition entre deux chaînes X et Y en utilisant une stratégie cache-oblivious.Il divise récursivement les calculs pour optimiser l'utilisation du cache sans connaissance spécifique de sa taille. Lorsqu une sous-matrice est suffisamment petite le programme effectue directement les calculs pour cette zone, puis combine les résultats pour obtenir la distance finale.

Analyse du coût théorique de ce programme en fonction de N et M en notation $\Theta(...)$)

1. place mémoire (ne pas compter les 2 séquences initiales X et Y en mémoire via `mmap`) : $\Theta(N)$

2. travail (nombre d'opérations) : Le travail total reste $\Theta(M \cdot N)$, car le programme parcourt tous les caractères de X et Y.

3. nombre de défauts de cache obligatoires (sur modèle CO, y compris sur X et Y): $\Theta((M+N)/L)$

4. nombre de défauts de cache si $Z \ll \min(N, M)$: $\Theta((M \cdot N)/L)$

5 Réglage du seuil d'arrêt récursif du programme cache oblivious (1 point)

Comment faites-vous sur une machine donnée pour choisir ce seuil d'arrêt? Quelle valeur avez vous choisi pour les PC de l'Ensimag? (2 à 3 lignes)

Le seuil d'arrêt récursif dans un programme cache-oblivious est choisi pour que chaque sous-problème tienne dans le cache sans dépasser sa taille. On peut tester différentes valeurs pour choisir celle qui maximise l'utilisation de cache avec un defaut de cache minimum. Pour les PC de l'Ensimag on a choisi :200

6 Expérimentation (7 points)

Description de la machine d'expérimentation :

- **Processeur :**
  - Architecture : x86_64
  - Modes opératoires des processeurs : 32-bit, 64-bit
  - Address sizes : 39 bits physiques, 48 bits virtuels
  - Boutisme : Little Endian
  - Caches (sum of all): L1d: 128 KiB (4 instances) L1i: 128 KiB (4 instances) L2: 1 MiB (4 instances) L3: 6 MiB (1 instance)

Mémoire :

```
       total       utilisé      libre     partagé tamp/cache   disponible
Mem:    7,5Gi       2,2Gi       4,6Gi       480Mi       1,4Gi       5,3Gi
Échange: 4,0Gi       680Mi       3,3Gi
```

Système :

```
Linux ensietu080 6.8.0-47-generic #47-Ubuntu SMP PREEMPT_DYNAMIC
Fri Sep 27 21:40:26 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
```

6.1 (3 points) Avec `valgrind -tool =cachegrind -D1=4096,4,64`

```
distanceEdition ba52_recent_omicron.fasta 153 N wuhan_hu_1.fasta 116 M
```

en prenant pour N et M les valeurs dans le tableau ci-dessous.

Les paramètres du cache LL de second niveau est : *LL cache: 6291456 B, 64 B, 12-way associative* [1] *pour LL.*
Le tableau ci-dessous est un exemple, complété avec vos résultats et ensuite analysé.

		récursif mémo		
N	M	#Irefs	#Drefs	#D1miss
1000	1000	217,201,229	122,130,086	4,930,604
2000	1000	433,517,982	243,481,237	11,027,454
4000	1000	867,421,282	487,521,539	23,230,948
2000	2000	867,231,489	487,948,668	19,904,091
4000	4000	3,465,527,823	1,950,449,674	80,012,497
6000	6000	7,795,922,454	4,387,938,297	180,348,707
8000	8000	13,857,331,210	7,799,829,695	320,934,866

itératif		
#Irefs	#Drefs	#D1miss
135,386,603	73,932,499	148,359
269,791,186	147,040,299	292,795
540,001,776	294,657,509	580,698
540,801,463	295,494,275	576,873
2,162,252,532	1,181,662,947	2,280,207
4,864,532,352	2,658,558,743	5,091,983
8,647,443,718	4,725,984,453	9,107,235

cache aware		
#Irefs	#Drefs	#D1miss
84,594,478	38,951,123	8,495
168,968,510	77,824,707	12,047
337,716,282	155,571,779	19,170
337,685,288	155,556,795	15,051
1,349,924,288	621,919,355	44,351
3,036,604,677	1,398,963,340	115,653
5,398,229,469	2,487,000,188	155,177

cache oblivious		
#Irefs	#Drefs	#D1miss
84,596,427	38,951,507	8,215
168,973,619	77,826,587	12,140
337,727,711	155,576,651	15,596
337,697,573	155,562,595	23,428
1,349,982,205	621,950,963	57,220
3,041,993,823	1,402,289,587	74,357
5,398,880,919	2,487,392,467	113,802

[1] par exemple: `valgrind -tool=cachegrind -D1=4096,4,64 -LL=65536,16,256 ...` mais ce n'est pas demandé car cela allonge le temps de simulation.

Important: analyse expérimentale: ces mesures expérimentales sont elles en accord avec les coûts analysés théroiquement (justifier) ? Quel algorithme se comporte le mieux avec valgrind et les paramètres proposés, pourquoi ?

- **Analyse :**
  - La méthode mémoisation a un nombre de défauts de cache qui augmente exponentiellement avec le nombre d'entrée.
  - La méthode itératif a un nombre considérablement plus réduit de défauts de cache que la méthode mémoisation. Mais les méthodes aware(avec comme taille de bloc k1 = 128 et k2 = 128) et oblivious avec un seuil 150 sont les plus performants avec un nombre de défaut de cache réduit encore davantage.
  - Les données expérimentales sont en accord avec l'analyse théorique. Les augmentations observées dans les références de données et d'instructions, ainsi que la hausse des manques de cache, sont cohérentes avec les défauts de cache calculés précedement (question 2,3,4).

6.2 (3 points) Sans valgrind, par exécution de la commande :

```
distanceEdition  GCA_024498555.1_ASM2449855v1_genomic.fna  77328790  M
                 GCF_000001735.4_TAIR10.1_genomic.fna      30808129  N
```

On mesure le temps écoulé, le temps CPU et l'énergie consommée avec : `time`
ou...
L'énergie consommée sur le processeur peut être estimée en regardant le compteur RAPL d'énergie (en microJoule) pour chaque core avant et après l'exécution et en faisant la différence. Le compteur du core K est dans le fichier `/sys/class/powercap/intel-rapl/intel-rapl:K/energy_uj` .
Par exemple, pour le cœur 0: `/sys/class/powercap/intel-rapl/intel-rapl:0/energy_uj`

Nota bene: pour avoir un résultat fiable/reproductible (si variailité), il est préférable de faire chaque mesure 5 fois et de reporter l'intervalle de confiance [min, moyenne, max].

		itératif			cache aware			cache oblivious		
N	M	temps cpu	temps écoulé	energie	temps cpu	temps écoulé	energie	temps cpu	temps écoulé	energie
10000	10000	0m1,478s	0m1,478s	23.1 J	0m0,712s	0m0,712s	10.5 J	0m0,691s	0m0,688s	10 J
20000	20000	0m5,972s	0m5,968s	97.5 J	0m2,788s	0m2,783s	43.8 J	0m2,733s	0m2,732s	40.7 J
30000	30000	0m13,670s	0m13,658s	206 J	0m6,237s	0m6,234s	89.8 J	0m6,343s	0m6,342s	96 J
40000	40000	0m25,016s	0m24,988s	344 J	0m11,083s	0m11,083s	157 J	0m11,080s	0m11,081s	158 J

Important: analyse expérimentale: ces mesures expérimentales sont elles en accord avec les coûts analysés théroiquement (justifier) ? Quel algorithme se comporte le mieux avec valgrind et les paramètres proposés, pourquoi ?

- **Analyse :**
  - méthode itérative : Plus lente avec les temps d'exécution CPU et écoulé les plus élevés, avec une consommation d'energie très élevée.
  - cache aware : Temps d'exécution CPU et écoulé considérablement réduits par rapport à la méthode itérative, avec une économie d'energie notable.
  - cache oblivious : Performances similaires à la méthode cache-aware. Globalement, cache-aware et cache-oblivious sont plus efficaces que l'itératif en termes de vitesse et d'énergie.

6.3 (1 point) Extrapolation: estimation de la durée et de l'énergie pour la commande :

```
distanceEdition  GCA_024498555.1_ASM2449855v1_genomic.fna  77328790  20236404
                 GCF_000001735.4_TAIR10.1_genomic.fna      30808129  19944517
```

A partir des résultats précédents, le programme *cache aware* est le plus performant pour la commande ci dessus (test 5); les ressources pour l'exécution seraient environ: *en utilisant la régression linéaire*

- Temps cpu (en s) : environ 6889.79 s (ou 1h 54m 50s)
- Energie (en kWh) : environ 96784.53 J (ou 26.91 kWh)

Question subsidiaire: comment feriez-vous pour avoir un programme s'exécutant en moins de 1 minute ? *donner le principe en moins d'une ligne, même 1 mot précis suffit!*
Blocking et parallélisme