

PROG

TD#4: Tri

2020-10-14

Exercice 1 : Tri par dénombrement

Le tri par dénombrement est un exemple de tri qui ne se fait pas par comparaison. Pour trier un tableau de nombres tous inférieurs à k , on procède d'après le pseudo-code suivant :

```
1 denomsort(T, Ts, k)
2     for (i = 0; i < k; i++)
3         C[i] = 0;
4     for (i = 0; i < len(T); i++)
5         C[T[i]] += 1;
6     for (i = 1; i < k; i++)
7         C[i] += C[i-1];
8     for (i = len(T) - 1; i >= 0; i--)
9         C[T[i]] = C[T[i]] - 1;
10        Ts[C[T[i]]] = T[i];
11    return Ts;
```

Q.1 : Justifiez la correction de cet algorithme, et le fait que le tri effectué est *stable* (c'est à dire que les positions relatives d'éléments égaux de A sont conservées dans B).

Q.2 : Écrivez une implémentation en C du tri par dénombrement de tableaux de nombres non signés stockés sur 32 bits. Vous ferez particulièrement attention aux allocations et désallocations mémoire.

Q.3 : Le tri par dénombrement peut être amené à être utilisé en sous-routine d'un autre algorithme de tri (cf. TP). Comment pouvez-vous limiter l'impact du coût des allocations mémoire de votre implémentation dans ce cas là ? Modifiez la en conséquence (si nécessaire), dans un premier temps quand k est garanti d'être "très petit", et dans un second temps pour des valeurs de k plus grandes.

Q.4 : Que pouvez-vous dire sur le type d'accès faits dans les différents tableaux ? Quel impact sur le temps d'exécution peut avoir la présence (ou non) de l'intégralité de ces tableaux dans la mémoire cache ?

Exercice 2 : Sheep and goat

On définit une permutation *sheep and goat* de la façon suivante : soit en entrée deux listes L1 et L2 de même taille, où L1 prend ses valeurs dans un ensemble arbitraire et L2 prend les siennes dans $\llbracket 0, 1 \rrbracket$, une permutation π appliquée conjointement à L1 et L2 est

dite sheep and goat si $\exists k \in \mathbb{N}$ t.q. $\forall i < k$, $L2[\pi(i)] = 1$ et $\forall i \geq k$, $L2[\pi(i)] = 0$ et si $\forall i, j > i$, $L2[i] = L2[j] \Rightarrow \pi(i) < \pi(j)$.

Q.1 : Comment pouvez vous exprimer une permutation sheep and goat à l'aide d'un algorithme de tri ? Quel algorithme de tri proposeriez vous d'utiliser dans une implémentation ?

L'instruction `pext`, du jeu d'extension x86 *BMI2* prend en entrée deux entiers non signés de 64 bits `a` et `mask` et exécute le pseudocode suivant :

```
1 tmp := a
2 dst := 0
3 m := 0
4 k := 0
5 DO WHILE m < 64
6     IF mask[m] == 1
7         dst[k] := tmp[m]
8         k := k + 1
9     FI
10    m := m + 1
11 OD
```

Cette instruction est de plus accessible via l'intrinsèque `_pext_u64`.

Q.2 : Ecrivez une fonction utilisant `pext` pour implémenter efficacement une permutation sheep and goat quand L1 est de taille 64 et à valeur dans $\llbracket 0, 1 \rrbracket$ ¹.

1. En guise d'indice, on notera l'existence d'une instruction `popcnt` (disponible via l'intrinsèque `_mm_popcnt_u64`) qui renvoie le nombre de bits à 1 de sa source, ainsi que celle des opérateurs `<<` (qui décale l'écriture binaire de son opérande de droite vers la gauche d'autant de positions que son opérande de gauche) et `~` (qui complémente bit à bit son unique opérande).