Université Sidi Mohamed Ben Abdellah

Faculté des Sciences Dhar Mahraz Fès

Master Big Data Analytics & Smart System

TP 5 : Sélection Parallèle

Réalisé par le groupe 6:

EL BAGHDADI MOHAMED

BERRAG AYOUB

Enseignant :

Pr. MOHAMED MEKNASSI

2018 - 2019

SOMMAIRE

I. Introduction

Un algorithme de sélection est une méthode ayant pour but de trouver le k-nième plus petit élément d'un ensemble d'objets (étant donné un ordre et un entier k).

Dans ce travail pratique, on va effectuer ce qui suit :

- La réalisation d'un algorithme séquentiel de sélection parce que l'algorithme parallèle est base sur le séquentiel, ce dernier représente une procédure utilise dans le parallèle. On va implémenter cela en utilisant le langage de programmation Python.
- La réalisation d'un algorithme parallèle de sélection sur une machine EREW SM SIMD. Par suite appliquer ce dernier sur un exemple.

II. Algorithme de sélection séquentiel

- **Définition d l'algorithme de sélection**

Un algorithme de sélection est une méthode ayant pour but de trouver le k-nième plus petit élément d'un ensemble d'objets (étant donné un ordre et un entier k).

La question de la sélection est un problème essentiel en algorithmique, notamment dans la recherche du maximum, du minimum et de la médiane. Plusieurs algorithmes ont été proposés et plusieurs contextes ont été étudiés : algorithmes en ligne, complexité amortie, complexité en moyenne, ensemble d'objet particuliers etc.

Le problème de la sélection est très lié aux algorithmes de tri : l'un des algorithmes classiques, Quick select, utilise d'ailleurs le même principe que l'algorithme de tri Quick sort.

- **Algorithme :**

Procedure SEQUENTIAL SELECT (S, k)

Step 1: **if** $|S| < Q$ **then** sort S and return the kth element directly

else subdivide S into $|S|/Q$ subsequences of Q elements each (with up to Q-1 leftover elements)

end if. (Q is a small constant.)

Step 2: Sort each subsequence and determine its median.

Step 3: Call SEQUENTIAL SELECT recursively to find m, the median of the $|S|/Q$ medians found in Step 2.

Step 4: Create three subsequences S_1, S_2, and S_3 of elements of S smaller than, equal to, and larger than m, respectively.

Step 5: **if** $|S_1| \geq k$ **then** {the kth element of S must be in S_1}

call SEQUENTIAL SELECT recursively to find the kth element of S_1

else if $|S_1| + |S_2| \geq k$ **then** return m

else call SEQUENTIAL SELECT recursively to find the $(k-|S_1|-|S_2|)$the element of S_3

end if

end if.

- **Jeu d'essai :**

On applique SEQUENTIAL SELECT

On a un tableau de 40 éléments et Q=10 alors l'objectif est de trouver le k-nième plus petit élément avec K=4

13	50	51	31	19	12	16	73	54	60	80	90	93	32	44	33	1	23	43	63
3	11	14	40	0	88	7	67	22	18	9	10	98	17	58	93	8	34	66	5

Alors on le subdivise |S|/Q =40/10= 4 sous ensemble

On partitionne et on les trie on obtient :

S1

12	13	16	19	31	50	51	54	60	73

S2

1	23	32	33	43	44	63	80	90	93

S3

0	3	7	11	14	18	22	40	67	88

S4

5	8	9	10	17	34	58	66	93	98

On détermine pour chaque sous-ensemble la médiane

M1=37.9 donc la médiane = 31

M2=50.2 donc la médiane =44

M3=27 donc la médiane =22

M4=38.9 donc la médiane =34

On les stocke dans un tableau de la médiane

22	31	34	44

On détermine la médiane de la médiane dans ce cas m=22

Alors ont créé 3 sous-ensembles le premier possède les valeurs inférieures à 22 le deuxième qui sont égale à 22 et le troisième supérieur à 22

Si le nombre d'élément de sous-ensemble un est supérieure à k alors le k-nième élément se trouve dans S_1

Sinon si la somme des deux nombres d'élément du premier sous ensemble et le deuxième sous ensemble supérieure à k alors le k-nième élément se trouve dans S_2

Sinon on fait un appel récursif de la fonction séquentiel select pour chercher le (K-$|S_1|$-$|S_2|$) th élément of S_3

III. Algorithme de sélection parallèle

Nous sommes maintenant prêts à étudier un algorithme de sélection parallèle sur un EREW SM SIMD

- **Algorithme :**

Procedure PARALLEL SELECT (S, k)
Step 1:
if (S <4 **then** ***PI*** uses at most five comparisons to return the kth element
else
- S is subdivided into $|S|^{1-X}$ subsequences Si of length $|S|^X$ each, where $1 < i < |S|^{1-X}, and$
- subsequence Si is assigned to processor ***Pi.***

end if.
Step 2:
for $i = 1$ **to** $|S|^{1-X}$ **do in parallel**
- {Pi obtains the median mi, i.e., the $\lceil |Si|/2 \rceil$ element, of its associated Subsequence}

$$SEQUENTIAL\ SELECT\ (Si, \lceil |Si|/2 \rceil)$$

- Pi stores mi in ***M(i)***

end for.
Step 3:
{The procedure is called recursively to obtain the median m of ***M}***

$$PARALLEL\ SELECT\ (M, \lceil |Mi|/2 \rceil).$$

Step 4:

The sequence S is subdivided into three subsequences:

$L = \{s_i \in S: s_i < m\}$,
$E = \{s_i \in S: s_i = m\}$, and
$G = \{s_i \in S: s_i > m\}$.

Step 5: **if** $Ll > k$ **then** PARALLEL SELECT *(L,* k)
else **if** ***ILI*** + ***IEI2*** **k then** return **m**
else PARALLEL SELECT (*G*, k - *1 LJ - IEJ)*
End if
End if.

- **Jeu d'essai :**

N = 30 et soit k = 20, nous devons déterminer le vingt IIème éléments de S.

Supposons En outre, l'ordinateur EREW SM SIMD est constitué de six processeurs, (N = 6)

Etape 1

11	**1**	**21**	**7**	**8**	**11**	**6**	**9**	**8**	**12**	**3**	**10**	**20**	**15**	**7**
26	**30**	**21**	**3**	**6**	**4**	**0**	**1**	**7**	**5**	**13**	**31**	**11**	**2**	**9**

Après l'étape 1, chaque processeur a un sous-ensemble de S : Les 5 processeurs reçoivent six éléments.

T1	**11**	**1**	**21**	**7**	**8**
T2	**11**	**6**	**9**	**8**	**12**
T3	**3**	**10**	**20**	**15**	**7**
T4	**26**	**30**	**21**	**3**	**6**
T5	**4**	**0**	**1**	**7**	**5**
T6	**13**	**31**	**11**	**2**	**9**

Les Tableaux Triés

T1	**1**	**7**	**8**	**11**	**21**
T2	**6**	**8**	**9**	**11**	**12**
T3	**3**	**7**	**10**	**15**	**20**
T4	**3**	**6**	**21**	**26**	**30**
T5	**0**	**1**	**4**	**5**	**7**
T6	**2**	**9**	**11**	**13**	**31**

Etape 2

Maintenant, chaque processeur trouve la médiane de son sous-séquence

Médianes de 6 tableaux

8	**9**	**10**	**21**	**4**	**11**

Etape 3

Lorsque PARALLEL SELECT est appelé récursivement, il retourne la médiane

M = 9 des Médianes

Médiane des médianes = 9

Etape 4 :

Former les trois sous-séquences à savoir, L, E et G d'éléments plus petits,

Égale à, et supérieure à 9.

L=Eléments inferieur à la médiane **E** =médiane ***G***=élément supérieur à la médiane

3	**1**	**2**	**7**	**8**	**7**	**6**	**6**	**8**	**0**	**3**	**5**	**1**	**4**	**7**
9	**9**	**21**	**11**	**26**	**15**	**12**	**20**	**11**	**10**	**13**	**31**	**11**	**21**	**30**

Etape 5

|L|=12

|E|=2

|k|=20 > |L|+|E|=17

En doit travailler avec S=G et appeler récursivement à PARALLEL SELECT

Avec :

k = 20 - (12 + 2) = 6

Tableau S=G trié :

21	**11**	**26**	**15**	**12**	**20**	**11**	**10**	**13**	**31**	**11**	**21**	**30**

L=Eléments inferieur à la médiane E=médiane G=élément supérieur à la médiane

10	**11**	**11**	**11**	**12**	**13**	**15**	**20**	**21**	**21**	**26**	**30**	**31**

10	**11**	**11**	**11**

12	**13**	**15**	**20**

21	**21**	**26**	**30**	**31**

Médiane=**13**

On a **|L| < K**, alors : on doit travailler avec **|L|+|E|**

Puisque **|L|+|E|=6=K**

Alors on trouve que 20 -ème élément est 13

IV. Environnement de travail

- **Le matériel utilisé**

Processor:	Intel(R) Core(TM) i5 CPU M 540 @ 2.53GHz 2.53 GHz
Installed memory (RAM):	4.00 GB (3.80 GB usable)
System type:	64-bit Operating System

Figure 1: le matériel utilisé

- **Le logiciel utilisé**

Nous avons exécuté notre code Python sur le site :
https://www.tutorialspoint.com/execute_python_online.php

V. Conclusion

Nous avons présenté dans ce travail le fonctionnement d'algorithme de sélection dans une machine parallèle, ceci se base sur l'algorithme séquentiel qui est la procédure utilisée.

VI. Annexe

```
def QuickSelect(items, item_index):

   def select (lst, l, r, index):

      # base case
      if r == l:
         return lst[l]

      # choose random pivot
      pivot_index = random.randint(l, r)

      # move pivot to beginning of list
      lst[l], lst[pivot_index] = lst[pivot_index], lst[l]

      # partition
      i = l
        for j in xrange(l+1, r+1):
         if lst[j] < lst[l]:
```

```
            i += 1
            lst[i], lst[j] = lst[j], lst[i]

        # move pivot to correct location
        lst[i], lst[l] = lst[l], lst[i]

        # recursively partition one side only
        if index == i:
            return lst[i]
        elif index < i:
            return select(lst, l, i-1, index)
        else:
            return select(lst, i+1, r, index)

    if items is None or len(items) < 1:
        return None

    if item_index < 0 or item_index > len(items) - 1:
        raise IndexError()

    return select(items, 0, len(items) - 1, item_index)
```