Exercices : solutions de la série #7

Solution à l'exercice 1.

a. Temps : $\Theta(n)$.

b. Nombre de processeurs requis : n^2.

c. Coût : $\Theta(n^3)$.

d. Travail (en comptant uniquement, et ce de façon détaillée, les opérations *explicites* d'affectation) :

$$\begin{aligned} W(n) &= \sum_{i=1}^{n}(\sum_{j=1}^{n}((1+n)+1)) \\ &= \sum_{i=1}^{n}(n^2+2n) \\ &= n^3+2n^2 \\ &\in \Theta(n^3) \end{aligned}$$

Solution à l'exercice 2.

a. Temps : $\Theta(\lg\ n)$ (nombre d'itérations pour le calcul des sommes, en utilisant la stratégie du doublage récursif).

b. Nombre de processeurs requis : n^3 (pour le calcul des différents produits).

c. Coût : $\Theta(n^3\ \lg\ n)$.

d. Travail (en comptant, et ce de façon détaillée, les opérations *explicites* d'affectation) :
Tout d'abord, analysons le travail pour `sommeVect` :

$$\begin{aligned} W_0(n) &= \sum_{i=1}^{\lg\ n}(2\times\frac{n}{2^i})+1 \\ &= 2n\sum_{i=1}^{\lg\ n}(\frac{1}{2})^i+1 \\ &= 2n(\frac{\frac{1}{2}^{\lg\ n+1}-1}{\frac{1}{2}-1}-1)+1 \\ &= 2n(\frac{\frac{1}{2n}-1}{\frac{1}{2}-1}-1)+1 \end{aligned}$$

$$
\begin{aligned}
&= 2n(\frac{1-\frac{1}{2n}}{1-\frac{1}{2}}-1)+1\\
&= 2n(2-\frac{1}{n}-1)+1\\
&= 4n-2-2n+1\\
&= 2n-1
\end{aligned}
$$

$$
\begin{aligned}
W(n) &= \sum_{i,k,j=1}^{n}(1+1)+\sum_{i,j=1}^{n}(2n-1)\\
&= 2n^3+2n^3-n^2\\
&= 4n^3-n^2\\
&\in \Theta(n^3)
\end{aligned}
$$

Solution à l'exercice 3.

Pour améliorer le coût, donc obtenir un coût $\Theta(n^3)$ équivalent à celui de l'algorithme séquentiel classique, on pourrait modifier l'algorithme comme on l'a fait pour la recherche du minimum. Plus précisément, il faudrait réduire le nombre de processeurs à $n^3/\lg\ n$. Ceci peut être réalisé en faisant en sorte, durant la première partie de l'algorithme (calcul des produits), que chaque processeur traite $\Theta(\lg\ n)$ multiplications plutôt qu'une seule. Le reste de l'algorithme peut alors s'effectuer de la même façon. Le code MPD approprié est présenté à la Figure 1.

```
# Determiner les index (i, k, j) associes a l'item numElem.
procedure calculerIndex( int numElem, int n,
                         res int i, res int k, res int j )
{
   i = (numElem-1) / (n**2) + 1;
   k = ((numElem-1) / n) % n + 1;
   j = (numElem-1) % n + 1;
}

# Calculer une collection de produits.
procedure calculerProduits( int numProc, int a[*,*], int b[*,*], int n,
                            res int t[*,*,*] )
{
  int minNumElem = (numProc-1) * lg(n) + 1;
  int maxNumElem = numProc * lg(n);

  for [numElem = minNumElem to maxNumElem] {
    int i, j, k;
    calculerIndex( numElem, n, i, k, j );
    t[i,k,j] = a[i,k] * b[k,j];
  }
}


procedure sommeVect( int t[*], int n ) returns int s
... Inchange ...

# Multiplications de matrices.
procedure multMatrices( int a[*,*], int b[*,*], ref int c[*,*], int n )
{
  int t[n,n,n];

  co [i = 1 to (n**3)/lg(n)]
    calculerProduits( a, b, t, i, n );
  oc
  co [i = 1 to n, j = 1 to n]
    c[i,j] = sommeVect( t[i,*,j], n );
  oc
}
```

Figure 1: Solution parallèle de coût optimal pour la multiplication de deux matrices $n \times n$