UE High Performance Scientific Computing
Examen de Algorithmes pour le Calcul à Hautes Performances

14 Decembre 2022

— **Tous les documents sont autorisés.**

1 Parallélisme mèmoire partagée (10pts)

1.1 Question 1 (7pts)

Soit A une matrice de taille $M \times N$ partitionnée en blocs carrés de taille B avec $M = B \times MB$ et $N = B \times NB$. On suppose $MB \geq NB$ et on considère le pseudo-code ci-dessous :

```
for(k=1; k<NB; k++) {
   A[k,k] = red(A[k,k-1])
   for(i=k+1; i<MB; i++) {
      A[i,k] = blue(A[i-1,k], A[i,k-1])
   }
}
```

Dans ce code nous avons supposé
— que le pseudo-code ne travaille que sur la partie triangulaire (trapeziodale si $MB > NB$) inférieure de A ;
— que $A(i, j)$ dénote le bloc de taille $B \times B$ de la matrice A sur le bloc-ligne i et le bloc-colonne j ;
— la notation `output = function(input`$_1$`, input`$_2$`,...)` pour les appels à fonction.
Le coût des fonctions est de B^3 pour `red` et $2B^3$ pour `blue`.

1. (2 Pts) En considérant une exécution séquentielle en arithmétique réelle double précision (1 coefficient = 8 Bytes), déterminez la valeur minimale pour la taille de bloc B pour que la vitesse d'exécution de chaque tâche (et donc de l'algorithme complet) ne soit pas limitée par la vitesse de la mémoire sur une machine avec une puissance crête de s $Gflop/s$ et une bande passante mémoire de b GB/s.

2. (2 Pts) Dessinez le graphe de dépendances du pseudo-code pour le cas $MB = 4$, $NB = 4$; identifiez le chemin critique et calculez le speedup idéal atteint par une exécution parallèle avec un nombre infini de processeurs ;
3. (3 Pts) En généralisant votre calcul de speedup idéal au cas d'une matrice de taille $MB \times NB$ quelconque, discutez la scalabilité de cet algorithme dans les deux cas suivants :
   — Matrice fortement surdeterminée : ceci est le cas où NB est fixe est MB augmente ;
   — Matrice carrée : ceci est le cas d'une matrice de taille croissante avec $MB == NB$.

1.2 Question 2 (3pts)

Expliquez pourquoi et comment la distribution des données peut influencer les performances d'un code parallèle sur un calculateur à **mémoire partagée** de type NUMA (i.e., avec une seule mémoire logique mais plusieurs modules de mémoire).

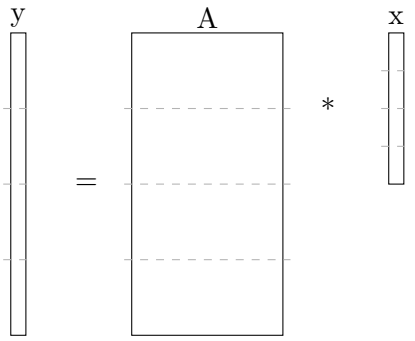
2 Parallélisme mèmoire distribuée (14pts)

2.1 Iso-efficacité (10pts)

Considérez le code séquentiel suivant realisant un produit matrice-vecteur

```
for(i=0; i<m; i++)
   for(j=0; j<n; j++)
      y[i] += A[i,j] * x[j];
```

où `x` est un vecteur de taille `n`, `y` est un vecteur de taille `m` et `A` une matrice de taille `m x n`. Toutes les opérations sont sur des réels, virgule flottante en précision double.



Sur un calculateur parallèle à mémoire distribuée on supposera que `x`, `y` et `A` sont repartis sur les `p` processus disponibles suivant un partitionnement en bloc-lignes comme illustré dans la figure ci-dessus. Chaque processus possédera donc une partie de `x` dans le tableau `x_loc`, une partie de `y` dans le tableau `y_loc` et une partie de `A` dans le tableau `A_loc`.

Le produit matrice-vecteur peut donc être parallélisé comme ceci

```
/* Rassembler les x_loc dans un vecteur x */
???
for(i=0; i<m/p; i++)
   for(j=0; j<n; j++)
      y_loc[i] = A_loc[i,j] * x[j];
```

Ici nous avons fait l'hypothèse que m et n sont multiples de p. En utilisant le modèle de Hockney avec paramètres γ (temps pour un calcul scalaire en précision double), α (latence d'accès au réseau) et β (temps de transfert d'un scalaire réel en précision double) :

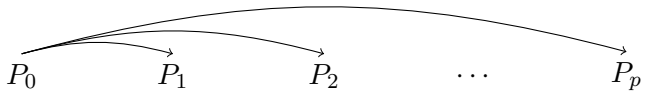
1. (2pts) Écrivez le modèle $T(m, n, 1)$ pour le temps d'exécution du code séquentiel ;
2. (2pts) Identifiez la communication collective permettant d'effectuer l'étape décrite dans le commentaire du pseudo-code ci-dessus.
3. (3pts) Écrivez le modèle $T(m, n, p)$ pour le temps d'exécution du code parallèle ;
4. (3pts) Étudiez l'iso-efficacité du code parallèle en faisant l'hypothèse simplificatrice $\alpha = 0$ et répondez aux questions suivantes :
   (a) cas m variable et n fixe : si le nombre de processus augmente de p à p', est il possible de garder une efficacité constante en augmentant m ? si oui, de combien faut-il l'augmenter ?
   (b) cas m fixe et n variable : si le nombre de processus augmente de p à p', est il possible de garder une efficacité constante en augmentant n ? si oui, de combien faut-il l'augmenter ?

   Fournissez une explication pour la différence de comportement dans les deux cas.

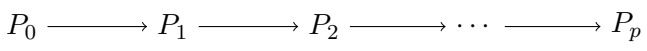
2.2 Communications collectives (4pts)

Considérons le broadcast de données de taille m bytes à partir d'un processus P_0 vers p processus $P_1, ..., P_p$ suivant deux schémas de communication. Dans le premier, c'est P_0 qui envoie p messages à chacun des destinataires (comme dans la partie haute de la figure ci-dessous) ; dans le deuxième, P_0 envoie le message à P_1 qui le fait suivre à P_2 et ainsi de suite jusqu'à P_p (comme dans la partie basse de la figure ci-dessous).

Schema 1 :



Schema 2 :



Sous les hypotheses du modèle de Hockney :
— α c'est la latence et β le temps necessaire à transferer 1 byte,
— un processus ne peut pas recevoir ou envoyer deux messages simultanement mais il peut envoyer un message et en recevoir un autre en même temps,

1. (1pt) calculez, pour les deux schémas, le temps necessaire à faire le broadcast d'une donnée de taille m bytes ;

2. (2pts) calculez, pour les deux schémas, le temps necessaire à faire le broadcast de deux données de taille m bytes ;
3. (1pt) sur la base des reponses aux questions precedentes, illustrez les avantages du deuxieme schéma.