**INF7235 — Programmation parallèle haute performance**
**Examen final (Hiver 2014)**

---

**Durée:** (3 heures) **Documentation autorisée:** Toute documentation personnelle.

---

**Nom:** _____

**Remarques :**

- **Vous devez répondre dans le cahier d'examen** qui vous est fourni.
- Lorsqu'il est demandé de **justifier** vos réponses, tant la clarté, la justesse que la pertinence des explications seront évaluées — donc si vous écrivez des choses *fausses ou «qui n'ont pas rapport»*, vous serez pénalisés.

---

## 1. Mesures de performances (10 pts)

Soit une machine parallèle *à mémoire distribuée* comportant 32 processeurs — donc un petit *cluster*. On a mesuré le temps requis par un **programme parallèle** pour traiter un même ensemble de données avec différents nombres de processeurs (1, 2, 4, 8, 16) et on a obtenu les temps suivants (en $ms$) :

| Nb. procs | Temps |
|---|---|
| 1 | 1200 |
| 2 | 800 |
| 4 | 300 |
| 8 | 200 |
| 16 | 120 |

Quant au temps d'exécution pour une version **séquentielle** de ce programme sur les mêmes données, il est de 1000 $ms$.

a. Quelle est **l'accélération absolue** lorsqu'on utilise 8 processeurs?

b. Quelle est **l'accélération relative** lorsqu'on utilise 8 processeurs?

c. En utilisant l'accélération **relative**, pour quel nombre de processeurs obtient-on la meilleure **efficacité** et quelle est cette efficacité?

d. Peut-on prédire quel sera le comportement du programme (accélération et efficacité) si on l'exécute sur les mêmes données mais en utilisant cette fois l'ensemble des 32 processeurs de la machine? Expliquez brièvement pourquoi le programme aurait ce comportement.

e. Peut-on prédire quel sera le comportement du programme (accélération et efficacité) si on l'exécute avec différents nombres de processeurs (1, 2, 4, ..., 32) mais cette fois en augmentant de façon importante la taille de l'ensemble des données à traiter — par exemple, en doublant la taille des données? Expliquez brièvement pourquoi le programme aurait ce comportement.

## 2. Construction d'un histogramme (mémoire partagée avec TBB) (10 pts)

On veut paralléliser une fonction C/C++ qui produit un histogramme pour une série de nombres entiers. L'extrait de code C 1 (p. 3) présente cette fonction.

Chaque nombre du tableau `elems` est un entier compris entre 0 et `valMax` (inclusivement). Soit alors `k` un entier compris inclusivement entre 0 et `valMax` et soit `histo` le tableau retourné par un appel à la fonction `histogramme(elems, n, valMax)`. On aura alors `histo[k]` = *nombre d'occurences de* `k` *dans* `elems`.

Par exemple, supposons qu'on ait les 12 valeurs suivantes et l'appel à la fonction `histogramme` comme suit :

```
elems = { 10, 1, 3, 3, 3, 2, 9, 1, 1, 1, 3, 10 }

histo = histogramme( elems, 12, 10 )
```

Alors l'histogramme résultant serait comme suit :

```
histo = { 0, 4, 1, 4, 0, 0, 0, 0, 0, 1, 2 }
```

Écrivez une version parallèle de la fonction `histogramme` **en utilisant les *Threading Building Blocks* d'Intel**.

- **Remarque :** Une simple parallélisation de la boucle `for` ci-bas ne fonctionnera pas, car l'opération «*var* `+= 1`» n'est pas atomique — donc il y a des interférences possibles entre les tâches qui accèdent à *var* !
- **Suggestion :** Utilisez une approche de **parallélisme de résultat!** (Et n'oubliez pas qu'on est en **mémoire partagée**!)

```
/*
    Fonction pour le calcul d'un histogramme.

   Données d'entrée:
   - elems: Tableau d'entiers non-négatifs (0 ≤ elems[i]≤ valMax)

   - n: Taille du tableau elems

   - valMax = Valeur maximum parmi les éléments d'elems

   Résultat:
   - Pointeur vers le tableau (dynamique) de l'histogramme résultant
*/


int* genererHistogramme( int elems[], int n, int valMax )
{
  // On alloue le tableau pour l'histogramme résultant.
  int *histo = (int*) malloc( (valMax+1) * sizeof(int) );

  // On initialise l'histogramme.
  for( int i = 0; i <= valMax; i++ ) {
    histo[i] = 0;
  }

  // On construit l'histogramme en analysant les données.
  for( int i = 0; i < n; i++ ) {
    histo[elems[i]] += 1;
  }

  return histo;
}
```

**Extrait de code C 1:** Fonction séquentielle, en C/C++, pour le calcul d'un histogramme.

### 3. Recherche des racines d'une fonction (mémoire partagée) (10 pts)

Soit $f$ une fonction sur les réels $f : \mathcal{R} \rightarrow \mathcal{R}$. Une **racine** de $f$ est un point $r$ tel que $f(r) = 0$. Étant donné un intervalle $[a, b]$ sur les réels, la fonction $f$ peut posséder 0, 1 ou plusieurs racines à l'intérieur d'un intervalle : voir figure 1 pour des exemples avec différentes valeurs de $a$ et $b$.

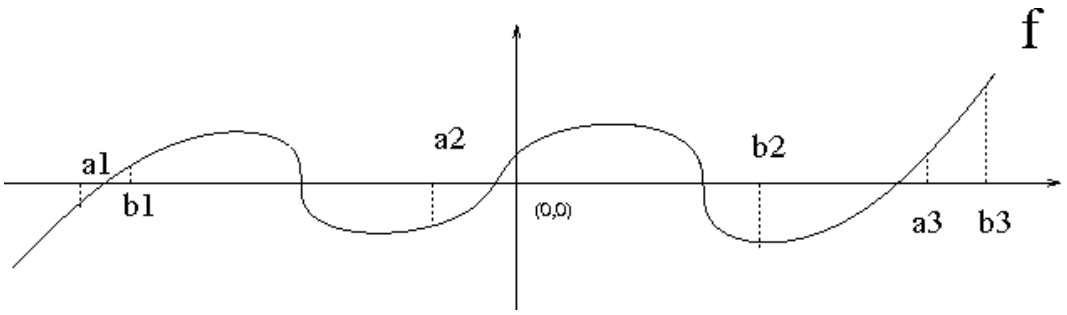


Figure 1: Racines d'une fonction $f$ sur divers intervalles : $f$ possède une (1) racine dans l'intervalle $[a1, b1]$, deux (2) racines dans l'intervalle $[a2, b2]$ et aucune (0) racine dans l'intervalle $[a3, b3]$.

Comme on manipule des nombres réels, donc avec erreurs possibles de troncation ou d'arrondissement, il n'est pas toujours assuré qu'on puisse trouver $r$ tel que, de façon exacte, $f(r) = 0$. Une façon d'obtenir une **approximation** d'une racine de $f$ consiste alors à trouver un intervalle $[a', b']$ *suffisamment petit* (avec $b' - a' < \epsilon$ où $\epsilon$ est «petit») tel que $f(a')$ et $f(b')$ sont des valeurs de signes opposés. Ainsi, soit $\epsilon > 0$ une *petite valeur réelle positive*. Soit $a'$ et $b'$ tel que $a' < b'$ avec $b' - a' < \epsilon$. Alors on dira que $f$ **possède une racine dans l'intervalle** $(a', b')$ **si** $f(a')$ **et** $f(b')$ **sont de signes opposés.**

Le pseudocode 1 présente un algorithme séquentiel pour trouver le nombre de racines d'une fonction $f$ entre $a$ et $b$. (Note importante : il s'agit d'un **algorithme simplifié** pour les besoins de l'exercice : un algorithme correct devrait aussi vérifier si un des points a un signe «nul», donc est une «vraie» racine.)

Votre tâche est de développer un programme MPD permettant de *trouver le nombre de racines* d'une fonction $f$ sur un intervalle. Plus précisément, ce programme s'exécuterait sur une machine semblable à `zeta` — machine multiprocesseurs avec mémoire partagée et petit nombre de processeurs ($\approx$ 6–8). Le programme recevrait les arguments suivants sur la ligne de commande :

- Les bornes inférieure et supérieure de l'intervalle — paramètres `a` et `b`. On suppose que `a < b`.
- L'erreur maximale acceptée — paramètre `epsilon`. On suppose que `epsilon > 0`.
- Le numéro d'identification de la fonction (entier non négatif) pour laquelle on désire trouver les racines.

  On suppose qu'il existe une bibliothèque de fonctions, liée au programme au moment de l'édition des liens, qui définit un ensemble de fonctions possibles. Cette bibliothèque exporte simplement une opération ayant l'interface suivante :

  ```
  double evaluer( int f, double x );
  //  Évalue la fonction identifiée par f  au point x.
  ```

  **On ne possède aucune information quant au temps d'exécution requis par ces fonctions.** Le temps requis pour évaluer une fonction en divers points pourrait être constant d'un point à un autre, ou au contraire pourrait être grandement variable.

Vous n'avez pas à écrire de code MPD. Vous devez simplement décrire et comparer — avantages et désavantages, forces et faiblesses — diverses façons d'écrire un tel programme. Surtout, vous devez conclure en indiquant l'approche que vous suggéreriez d'utiliser, et pourquoi.

```
Fonction auxiliaire
FONCTION signe( x )
DEBUT
  SI x < 0.0 ALORS
    RETOURNER -1
  SINON
    RETOURNER +1
  FIN
FIN


PROGRAMME PRINCIPAL
DEBUT
  nbZeros ← 0
  nbIntervalles ← (b − a) / epsilon

  POUR i ← 0 TO nbIntervalles-1 FAIRE
    a' ← a + i * epsilon

    fa ← evaluer( f, a' )
    fb ← evaluer( f, a' + epsilon )

    SI signe(fa) ≠ signe(fb) ALORS
      nbZeros ← nbZeros + 1
    FIN
  FIN
FIN
```

**Pseudocode 1:** Pseudocode d'un programme séquentiel (avec une fonction auxiliaire) pour trouver le nombres de racines d'une fonction $f$ comprises dans un intervalle $[a, b]$.

## 4. SAXPY en MPI/C (mémoire distribuée) (10 pts)

Un calcul fréquemment rencontré lorsqu'on manipule des matrices de nombres réels est celui du calcul *SAXPY* — «*Single-Precision A · X Plus Y*».

Plus précisément, un calcul *SAXPY* reçoit en entrée un scalaire `a`, deux vecteurs `X` et `Y` de taille `n`, puis il multiplie chaque élément `X[i]` par `a` et **ajoute** le résultat à `Y[i]`. Une mise en oeuvre séquentielle en C de `saxpy` a donc l'allure suivante :

```
void saxpy( float a, float x[], float y[], int n )
{
  for ( int i = 0; i < n; i++ ) {
      y[i] = a*x[i] + y[i];
  }
}
```

Le programme MPI 1 ci-bas présente un squelette pour une version parallèle en MPI/C d'une telle procédure, à laquelle deux arguments additionnels sont spécifiés :

- Le processus `racine` qui connait initialement la valeur `a` et les vecteurs `x` et `y` (ainsi que la taille `n`), processus vers lequel le résultat `y` doit ensuite être retourné ;
- Le communicateur qui doit être utilisé pour les communications durant l'exécution de `saxpy`.

Un appel de cette procédure aurait donc l'allure suivante :

```
//  Exemple d'appel
// On suppose que tous les processus de MPI_COMM_WORLD seront utilisés.
MPI_Comm comm;
MPI_Comm_dup( MPI_COMM_WORLD, &comm );

// Le scalaire a et les vecteurs x et y sont sur le processus 0.
saxpy( a, x, y, n, 0, comm );
// Le vecteur résultant y est maintenant sur le processus 0.

MPI_Comm_free( &comm );
```

Complétez cette procédure. On suppose qu'on utilise une machine semblable au *cluster* utilisé durant les labos, donc un petit *cluster* comptant au plus une dizaine de noeuds.

```
void saxpy( float a, float x[], float y[], int n, int racine, MPI_Comm comm )
{
  int nbProcs;
  MPI_Comm_size( comm, &nbProcs );
  assert( n % nbProcs == 0 );

  // À compléter.
  // ...
}
```

**Programme MPI 1:** Squelette de procédure `saxpy` en MPI/C — à compléter!

## 5. Calcul des maximums d'une matrice (échange de messages) (10 pts)

### Le problème

Soit $n$ un entier positif. On a de grosses matrices carrées, de taille $n \times n$, et on veut déterminer, *pour chaque ligne et chaque colonne*, la valeur maximum. De plus, ces maximums devront être connus d'un seul et unique processus, disons uniquement le processus 0.

Par exemple, on a ci-bas une matrice $9 \times 9$ avec, à droite et en bas, les différents maximums, qui doivent ultimement être tous connus du processus 0 :

```
15 23 34 36 74 81 91 11  2       | 91
46 74 24 14 95 52 31 42 18       | 95
21 29 38 33 34 32 35 36 37       | 38
48 55 69 77 86 11 14 22 13       | 86
24 22 36 58 65 73 77 79 71       | 79
71 21 13 29 22 24 28 25 26       | 71
79 76 71 75 73 37 22 18 14       | 79
 2  8  7  4  1  9 16 13  5       | 16
93 24 25 22 28 26 21 27 29       | 93

-------------------------

93 76 71 77 95 81 91 79 71
```

### Hypothèses

On travaille sur une machine parallèle *à mémoire distribuée* qui comporte $p$ processeurs. On veut un programme MPI/C **qui utilise uniquement des opérations collectives de communication** — donc pas de communication point-à-point (pas de `MPI_Send` ou `MPI_Recv`).

Soit $n$ la taille des matrices à traiter. On suppose que $n$ est divisible par $p$ ainsi que par $\sqrt{p}$, et que $p$ est un carré parfait — donc il existe $k$ tel que $k^2 = p$. La valeur de $n$ peut être très grande, donc $n$ est possiblement supérieur à $p$ de plusieurs ordres de grandeur — i.e., $p << n$.

On suppose aussi que les matrices à traiter sont déjà distribuées entre les processeurs. En d'autres mots, une matrice donnée est toujours distribuée sur les différents noeuds, et donc chaque noeud a une portion de chaque matrice.

### Ce que vous devez faire

- Présentez et comparez différentes façons de résoudre ce problème — quels sont les avantages vs. désavantages, forces vs. faiblesses de différentes approches?

  Vous n'avez pas à écrire de «vrai» code MPI/C. Vous devez simplement décrire *les grandes lignes de vos approches* — par exemple à l'aide de pseudocode de haut niveau — de façon suffisante pour qu'on ait une bonne idée de la stratégie utilisée. **Notamment, il est important d'indiquer les opérations collectives qui seraient utilisées**, mais sans nécessairement indiquer les détails de tous les arguments — au minimum, les données transmises et leur taille.
- Évidemment, **vous devez analyser l'impact du choix de distribution des matrices** sur le fonctionnement et l'efficacité.
- Si vous devez poser certaines hypothèses additionnelles pour simplifier votre analyse, indiquez-les explicitement.
- Concluez en indiquant **l'approche que vous suggéreriez d'utiliser**, de façon à bien exploiter les ressources de la machine.