INF7235 — Programmation parallèle haute performance
Examen final (Hiver 2009)

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

Remarques :

- Vous devez toujours **justifier** vos réponses. Tant la clarté, la justesse que la pertinence des explications seront évaluées.
- Vous devez **obligatoirement** répondre aux questions 1, 2 et 3.
- Vous devez ensuite répondre **à l'une des questions 4 ou 5**.
- Si vous répondez aux deux dernières questions, les points de l'une d'entre elle (la moins bien réussie) seront comptés **en bonus**, et ce à hauteur de 50 % (donc 10 points max.).

---

1. Problème des mots clés (laboratoire 2 en MPI) (15 pts)

Soit le programme de recherche de mots-clés fait au labo il y a quelques semaines — voir le listing distribué en classe.

Supposons qu'on connaisse, dès le départ, le nombre exact de fichiers à analyser et, surtout, qu'on sache *que tous les fichiers à analyser ont exactement la même taille.*

a. Expliquez brièvement pourquoi dans ce cas l'approche utilisée dans le labo ne serait pas nécessairement la meilleure et proposez une autre approche.

b. Écrivez le code MPI/C mettant en oeuvre cette autre approche.

   Vous pouvez supposer que les fichiers de données qui doivent être analysés sont directement accessibles à partir des différents noeuds (système de fichiers répartis). Vous pouvez aussi supposer que le nombre de fichiers à analyser est une valeur d'une forme appropriée — forme que vous devez préciser à l'aide d'une assertion.

   Tout comme dans la solution faite au labo, le fichier contenant la liste des mots clés à rechercher (nom de fichier spécifié comme premier argument sur la ligne de commande) et le fichier contenant la liste des noms de fichiers à analyser (deuxième argument) ne doivent être lus (chargés en mémoire) que par un unique processus.

   Vous pouvez évidemment utiliser toutes les procédures auxiliaires déjà définies. Vous pouvez aussi en introduire de nouvelles.

   (Il n'est évidemment pas nécessaire d'indiquer des assertions pour vérifier le résultat.)

2. Transposition de matrices (10 pts)

Rappels sur les matrices

Pour simplifier le problème (et l'analyse) on suppose qu'on traite des matrices carrées, de taille $n \times n$.

Soit une matrice A :

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n-1} & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n-1} & a_{3n} \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn-1} & a_{nn} \end{pmatrix}$$

La transposée de A est définie par :

$$A^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} & \dots & a_{n-11} & a_{n1} \\ a_{12} & a_{22} & a_{32} & \dots & a_{n-12} & a_{n2} \\ a_{13} & a_{23} & a_{33} & \dots & a_{n-13} & a_{n3} \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ a_{1n} & a_{2n} & a_{3n} & \dots & a_{n-1n} & a_{nn} \end{pmatrix}$$

Soit A une matrice décomposée en sous-matrices comme suit :

$$A = \begin{pmatrix} \overline{A_{11}} & \overline{A_{12}} & \dots & \overline{A_{1n}} \\ \overline{A_{21}} & \overline{A_{22}} & \dots & \overline{A_{2n}} \\ \overline{A_{31}} & \overline{A_{32}} & \dots & \overline{A_{3n}} \\ \vdots & \vdots & \dots & \vdots \\ \overline{A_{n1}} & \overline{A_{n2}} & \dots & \overline{A_{nn}} \end{pmatrix}$$

Alors :

$$A^T = \begin{pmatrix} \overline{A_{11}}^T & \overline{A_{21}}^T & \dots & \overline{A_{n1}}^T \\ \overline{A_{12}}^T & \overline{A_{22}}^T & \dots & \overline{A_{n2}}^T \\ \overline{A_{13}}^T & \overline{A_{23}}^T & \dots & \overline{A_{n3}}^T \\ \vdots & \vdots & \dots & \vdots \\ \overline{A_{1n}}^T & \overline{A_{2n}}^T & \dots & \overline{A_{nn}}^T \end{pmatrix}$$

Le problème

On travaille sur une machine parallèle *à mémoire distribuée* (communication par échange de messages) et on doit effectuer une série de transpositions de matrices. Donc, étant donné diverses matrices A, on désire produire les matrices $B = A^T$. De quelle façon serait-il préférable de distribuer les éléments de A entre les divers processeurs?

Indices :

- Imaginez une solution MPI et comparez aussi le nombre total de messages (de communications) échangés.
- Rappel mathématique : $x^2 - 1 = (x-1)(x+1)$

3. Mesures de performances (10 pts)

a. On a mesuré le temps d'exécution d'un programme pour divers nombres de processeurs sur une machine parallèle *à mémoire distribuée* comportant à 32 processeurs :

Nb. procs	**Temps**
1	1000
2	400
4	200
6	150
8	125
10	100
12	100
16	110

Peut-on prédire quel sera le comportement du programme lorsqu'exécuté sur l'ensemble des processeurs de la machine? Et, de façon générale (petit vs. grand nombre de processeurs), comment peut-on expliquer le comportement de ce programme?

b. Soit une machine parallèle *à mémoire partagée* (multi-processeurs) comptant 16 processeurs. Quelle doit être (en pourcentage) la fraction maximale d'exécution séquentielle de ce programme pour qu'on obtienne une efficacité de 50 %? Quelle sera alors l'accélération résultante?

Addendum : Indiquez ici si vous supposer que la taille du problème va/doit rester *constante* ou si elle va varier en fonction de l'augmentation du nombre de processeurs.

4. Distribution de la chaleur sur un circuit imprimé (20 pts)

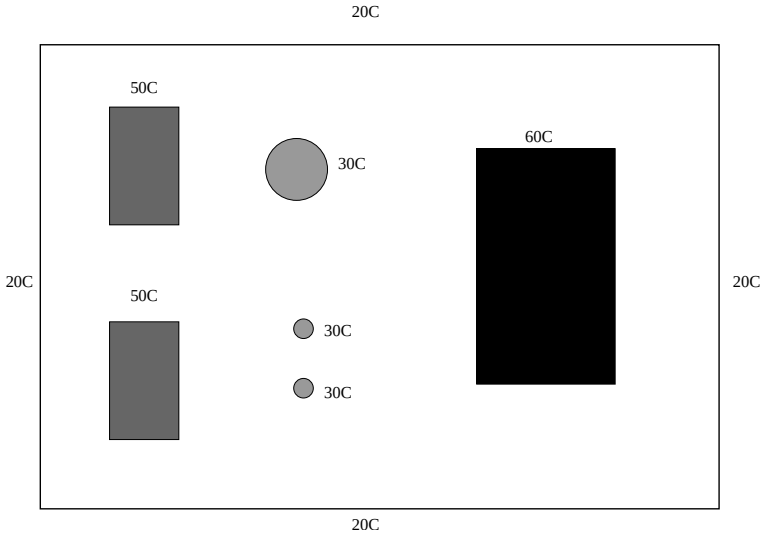

Figure 1: Composants d'un circuit imprimé.

La figure 1 présente un exemple de circuit imprimé avec un certain nombre de composants électroniques : le nombre indiqué spécifie la température (uniforme et fixe) générée par le fonctionnement de chaque composante. On suppose aussi que la température sur la bordure du circuit est stable et fixe à 20°C. On désire connaitre la distribution de la chaleur après une longue période de fonctionnement du circuit.

Plus précisément, on veut écrire un programme parallèle, dans un langage tel que MPD, qui permettrait de simuler la diffusion et distribution de chaleur pour de tels circuits. Le programme reçevrait, sur la ligne de commande, les arguments suivants :

- Argument 1 : Nom du fichier contenant la configuration du circuit imprimé : nombre de composants ; forme (cercle ou rectangle), position et température de chacune des composants.
- Argument 2 : Température sur la bordure.
- Argument 3 : Durée de la simulation = Nombre d'itérations durant lesquelles on veut simuler l'évolution de la température.

On veut exécuter ce programme sur une machine **à mémoire partagée** avec 16 processeurs (style machine Sun).

Décrivez et comparez (avantages et désavantages, forces et faiblesses) diverses façons d'écrire un programme MPD pour résoudre ce problème.

Vous n'avez pas à écrire de code MPD ; vous devez uniquement décrire les grandes lignes des approches — entre autres, du pseudo-code peut possibement être utile — et, surtout, conclure en indiquant l'approche que vous suggéreriez d'utiliser.

Vous pouvez évidemment supposer que vous définissez une structure de données appropriée — permettant de détecter rapidement les régions représentant les composants électroniques — mais sans nécessairement en donner les détails de mise en oeuvre.

5. Recherche des racines d'une fonction (20 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 de cet intervalle : voir figure 2, pour différents a et b.

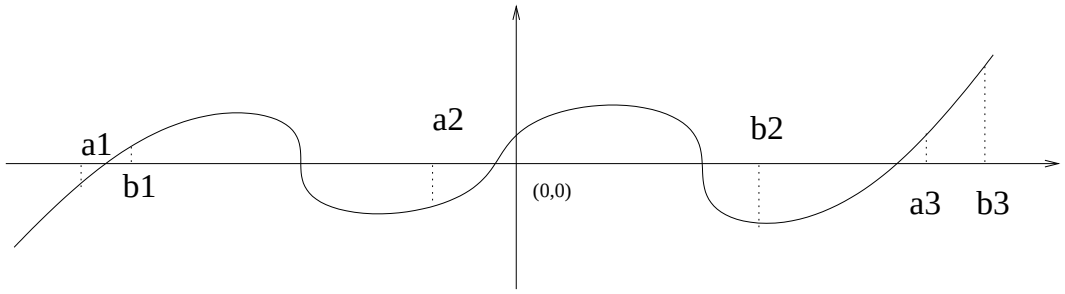

Figure 2: Racines d'une fonction f sur divers intervalles : f possède une (1) racine dans l'intervalle $[a1, b1]$, deux (2) racines dans l'intervalle $[a2, b2]$ et aucune (0) racine dans l'intervalle $[a3, b3]$.

Comme on manipule des nombres réels, donc avec erreurs possibles d'arrondissement, il n'est pas toujours assuré qu'on puisse trouver r tel que, de façon exacte, $f(r) = 0.0$. Une façon d'obtenir une *approximation* d'une racine de f consiste alors à trouver un intervalle $[a, b]$ *suffisamment petit* (avec $b - a < \epsilon$, et ϵ «petit») tel que $f(a)$ et $f(b)$ produisent *des valeurs de signes opposés.*

Décrivez et comparez (avantages et désavantages, forces et faiblesses) diverses façons d'écrire un programme MPI permettant de trouver *toutes les racines* d'une fonction f sur un intervalle et de les imprimer sur le processeur 0.

Ce programme recevrait les arguments suivants sur la ligne de commande :

- Les bornes inférieure et supérieure de l'intervalle — paramètres `a` et `b` — on suppose (précondition) que `a < b`.
- L'erreur maximale acceptée (paramètre `epsilon`, précondition `epsilon > 0`).
- Le numéro d'identification de la fonction 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 numFonction, double x );
  // Evalue la fonction identifiee par numFonction au point x.
  ```

  On ne possède **aucune information** quant au temps d'exécution requis par ces diverses fonctions — donc, le temps pourrait être relativement constant d'une fonction à une autre, ou au contraire, être éminemment variable.

Vous n'avez pas à écrire de code MPI ; vous devez uniquement décrire les grandes lignes de chacune des approches — entre autres, du pseudo-code peut être utile — et, surtout, conclure en indiquant l'approche que vous suggéreriez d'utiliser.