CONCOURS EXTERNE D'INGENIEUR DE RECHERCHE

DIS-20 / DIS-40

BAP E – INFORMATIQUE ET CALCUL SCIENTIFIQUE

Ouvert au titre de 2008
Arrêtés du 1er avril 2008 parus au JO du 5 avril 2008
Epreuve écrite
Note sur 20 – Coefficient 3 – Durée trois heures

le 24 juin 2008 de 9h30 à 12h30

Le candidat peut traiter les questions dans l'ordre de son choix.

Veillez à respecter l'anonymat dans les réponses.

Une attention particulière sera portée à la qualité de la rédaction, de la présentation, à l'orthographe et à la grammaire.

Exercice 1 : (6 points au total)

Q1 : (1 point)

En programmation, expliquez brièvement ce qu'est un *wrapper* et quel en est l'intérêt. Citez un outil de *wrapping* automatique de code.

Q2 : (2 points)

L'intégration continue est un ensemble de bonnes pratiques utilisées en génie logiciel. Elles consistent à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression de l'application en cours de développement.

- Donnez les bonnes pratiques élémentaires permettant de faire de l'intégration continue.
- Citez 2 outils logiciels permettant de faire de l'intégration continue.

Q3 : (3 points)

Attention : pour cet exercice, vous devez utiliser l'annexe

Lors d'envois / réceptions de messages sur des architectures parallèles, des situations de blocage surviennent. L'exemple élémentaire de blocage arrive avec des envois / réceptions implémentés de manière basique :

```
Processeur 1 :    int entier1 = 1
                  Envoyer entier1 à processeur 2
                  Recevoir entier2 de processeur 2
                  int somme = entier1 + entier2
Processeur 2 :    int entier2 = 2
                  Envoyer entier2 à processeur 1
                  Recevoir entier1 de processeur 1
                  int somme = entier1 + entier2
```

- Expliquez ce que fait ce code sur chacun des 2 processeurs. Quel scénario d'utilisation et quelles caractéristiques des fonctions d'envois / réceptions conduisent à un blocage ?
- Proposez un algorithme et une implémentation permettant de faire la somme de tous les entiers stockés sur 10 processeurs, en utilisant uniquement les fonctions MPI_ISEND, MPI_IRCV, MPI_WAITALL et MPI_COMM_RANK de MPI. Cette implémentation peut être faite en C, C++ ou Fortran. Pour cela, utilisez l'annexe fournie, et prenez MPI_COMM_WORLD comme *handle* de *comm* et 0 comme *tag* pour toutes les communications. À noter que MPI_INT est le type entier de MPI.

Exercice 2 : (7 points au total)

Attention de respecter l'anonymat de votre copie

Vous devez donner un séminaire pour les développeurs débutants (c'est-à-dire : jeunes stagiaires, doctorants, et même des scientifiques moins jeunes). L'exposé doit tenir en 45 minutes, questions de l'auditoire comprises.

Choisissez **un** des sujets suivants :

- *Profiling* et optimisation des performances logicielles : méthodes et outils.
- Documentations (développeurs et utilisateurs) et outils associés.

Donnez un plan de présentation avec le *timing* et vos transparents : (5 points).

Donnez également un résumé en français et en anglais de 5 à 10 lignes chacun : (2 points).

Exercice 3 : (7 points au total)

Q1 : (1,5 points)

Donnez les 2 principales familles de méthodes et au moins un algorithme pour chacune pour résoudre un système linéaire. Citez au moins deux solutions logicielles (librairies, exécutables, …) permettant de faire de telles résolutions.

Q2 : (2 points)

Nous cherchons à minimiser une fonction quelconque, à valeurs réelles, $F:\Re^n \rightarrow \Re$.

- Donnez les catégories de méthodes existantes pour minimiser la fonction $F(X)$, $X \in \Re^n$.
- Donnez les critères opportuns pour choisir une méthode de résolution adaptée.

Dans le cas d'une fonctionnelle issue d'une simulation (c'est-à-dire, que le calcul en un point nécessite la résolution d'une EDP), quelles contraintes additionnelles faut-il prendre en compte pour faire la minimisation et en quoi ces contraintes orientent-elles le choix méthodologique ?

Q3 : (2 points)

Dans le cas d'objets ou de scènes 3D on utilise des maillages non structurés pour décrire des volumes.

- Décrivez une structure de données typique de maillage 3D non structuré.
- Citez un exemple de format de fichier permettant de stocker des maillages non structurés.
- Citez 2 autres types de représentation pour décrire des données 3D.
- On veut partitionner un maillage non structuré 3D pour faire un calcul sur une architecture parallèle. Citez au moins un outil logiciel pour créer des partitions automatiquement.

Q4 : (1,5 points)

Pour de la simulation numérique :

- Citez les grandes familles de méthodes de résolution d'EDP.
- Il existe des méthodes *explicites* et *implicites* pour la résolution temporelle :
  - En pratique, et de façon générale, quelle catégorie de méthodes est plus coûteuse sur une itération et pourquoi ?
  - Quel est alors l'intérêt d'utiliser cette catégorie de méthodes ?
- Expliquez la phrase suivante : "le calcul converge au zéro machine".

ANNEXE

MPI_Isend

Begins a nonblocking send

Synopsis

```
#include "mpi.h"
int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest, int tag,
               MPI_Comm comm, MPI_Request *request )
```

Input Parameters

buf initial address of send buffer (choice)

count number of elements in send buffer (integer)

datatype datatype of each send buffer element (handle)

dest rank of destination (integer)

tag message tag (integer)

comm communicator (handle)

Output Parameter

request

communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

MPI_Irecv

Begins a nonblocking receive

Synopsis

```
#include "mpi.h"
int MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm, MPI_Request *request )
```

Input Parameters

buf initial address of receive buffer (choice)

count number of elements in receive buffer (integer)

datatype datatype of each receive buffer element (handle)

source rank of source (integer)

tag message tag (integer)

comm communicator (handle)

Output Parameter

request

communication request (handle)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

MPI_Waitall

Waits for all given communications to complete

Synopsis

```
#include "mpi.h"
int MPI_Waitall(
        int count,
        MPI_Request array_of_requests[],
        MPI_Status array_of_statuses[] )
```

Input Parameters

count	lists length (integer)
array_of_requests	array of requests (array of handles)

Output Parameter

array_of_statuses

array of status objects (array of Status). May be `MPI_STATUSES_IGNORE`

Note on status for send operations

For send operations, the only use of status is for MPI_Test_cancelled or in the case that there is an error, in which case the MPI_ERROR field of status will be set.

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.

MPI_Comm_rank

Determines the rank of the calling process in the communicator

Synopsis

```
#include "mpi.h"
int MPI_Comm_rank ( MPI_Comm comm, int *rank )
```

Input Parameters

comm

communicator (handle)

Output Parameter

rank

rank of the calling process in group of `comm` (integer)

Notes for Fortran

All MPI routines in Fortran (except for `MPI_WTIME` and `MPI_WTICK`) have an additional argument `ierr` at the end of the argument list. `ierr` is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the `call` statement.

All MPI objects (e.g., `MPI_Datatype`, `MPI_Comm`) are of type `INTEGER` in Fortran.