INRIA

CONCOURS EXTERNE D'INGENIEUR DE RECHERCHE

DIS1 / DIS2 / DIS3 / DIS4 / DIS5 / DIS6

BAP E – INFORMATIQUE ET CALCUL SCIENTIFIQUE

Ouvert au titre de 2010
Arrêtés du 7 avril 2010 parus au JO du 25 avril 2010

Epreuve écrite
Note sur 20 – Coefficient 3 – Durée trois heures

le 23 juin 2010 de 9h à 12h

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

Veillez à respecter l'anonymat dans les réponses.

Qualité de la rédaction, de la présentation, orthographe et grammaire. (1 point)

Exercice 1 : (2 points)

Q 1.1 : Dans un environnement Unix, vous voulez générer la version exécutable d'un code dont la source est composée de plusieurs éléments:

- des modules de calcul et d'interface que vous avez écrits en C++, permettant de générer deux exécutables :
  - une version avec une interface graphique X, nommée vq_x, de composants vq.cc, et x.cc
  - une version ligne de commande, nommée vq_cli, de composants vq.cc, et cli.cc
- des définitions utiles aux composants des deux exécutables, se trouvent dans un fichier vq.h
- trois modules écrits en Fortran qui implémentent des fonctions mathématiques que vous utilisez, modules que vous n'avez pas écrits mais qui vous ont été fournis: vector.f, matrix.f et vquant.f

Écrivez un `Makefile` permettant de générer les deux exécutables, à défaut écrivez les commandes à exécuter pour obtenir les exécutables. Vous ferez en sorte que les 3 modules mathématiques puissent facilement être utilisables par d'autres programmes.

Q 1.2 : Pourquoi est il recommandé d'utiliser des règles de codage? (10 lignes maximum). Donner quatre exemples de règles de codage.

Exercice 2 : (2 points)

Q 2.1 : Quelque soit le modèle de développement adopté ou requis, pouvez-vous décrire, en général et selon vous, les étapes clés du développement d'un logiciel ?

Q 2.2 : Pouvez-vous préconiser quelques outils associés à chacune des étapes ?

Exercice 3 : (5 points)

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

Attention de respecter l'anonymat de votre copie

A partir des 2 documents joints en annexe

a) Expliquez en une demi page (400 mots maximum), rédigée ***en français***, la notion de couverture de code (avantages, limites).

b) En vous appuyant sur les outils évoqués en annexe, rédigez un argumentaire ***en anglais*** d'une demi page (400 mots maximum) à destination des responsables d'équipes de recherche :

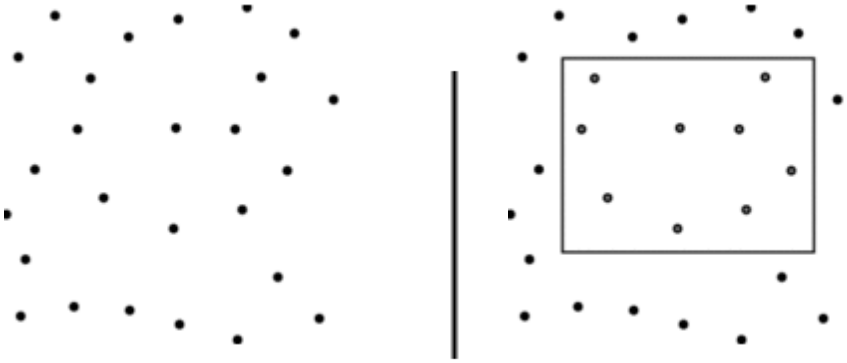
- identifiez des contextes dans lesquels cette démarche est justifiée
- détaillez la démarche.

Exercice 4 : (2 points)

Soit S un ensemble de ***N*** points dans le plan, chacun repéré par deux coordonnées. On appelle ***recherche 2D par intervalle***, le problème consistant à construire l'ensemble ***T***, contenant les points de S se trouvant à l'intérieur d'un rectangle R donné.

Sur la figure ci-dessous, l'image de gauche représente l'ensemble des points en entrée et celle de droite montre les points à retourner pour un rectangle donné.

Par exemple, S est un ensemble de villes parmi lesquelles nous souhaitons retrouver celles dont les coordonnées sont dans des intervalles donnés de latitude et de longitude.



Nous faisons l'hypothèse qu'un nombre important de requêtes (donc de rectangles) sont faites pour un même ensemble de points. Décrivez deux algorithmes et les structures de données correspondantes pour implémenter la recherche 2D par intervalle.

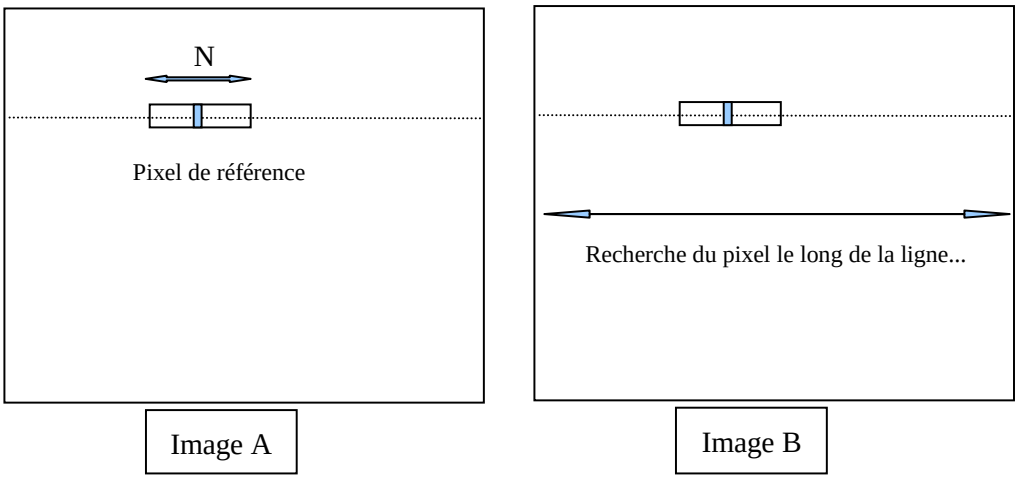
Pour chacune de vos propositions, vous expliciterez (en justifiant) le coût au pire cas de l'algorithme, coût en temps d'exécution et en mémoire. On prendra comme modèle de coût d'exécution le nombre de fois qu'un élément de vos structures de données est accédé par l'algorithme (en écriture ou en lecture).

Exercice 5 : (2 points)

On vous demande de mettre en place une plateforme expérimentale constituée de 4 têtes stéréoscopiques. Chaque tête stéréoscopique est constituée de deux caméras numériques monochromes. Chaque caméra, d'une résolution d'images de 1280x1024 pixels, dispose d'une interface Ethernet. La fréquence d'acquisition est de 25 images par seconde, la résolution par pixel est de 8 bits.

Chaque tête stéréoscopique doit permettre de réaliser, en temps réel et en connexion avec un cluster de calculateurs, une carte (matrice) de disparités entre chaque paire d'images consécutives de la séquence vidéo. En plus de ce calcul en temps réel, la plateforme doit assurer le stockage des images brutes sur disques durs.

Une carte de disparités indique pour chaque pixel la distance entre le point dans l'image A et son correspondant dans l'image B. On propose un algorithme simplifié de recherche de correspondance. Pour chaque pixel de l'image A, on recherche le pixel « le plus ressemblant » dans la même ligne de l'image B. Le critère de ressemblance est la comparaison par corrélation des 2 régions (fenêtres 1xN) autour du pixel de référence dans A et du pixel candidat dans B (cf. schéma).



Q 5.1 :

a) En allouant un disque dur par caméra, calculez le taux de transfert minimum, en mégaoctets par seconde, que doit supporter le disque pour le stockage en temps-réel d'une séquence d'images.

b) Quels sont les avantages et les inconvénients des deux configurations suivantes :

- connecter chaque caméra sur un commutateur réseau commun,
- connecter, en point à point, une caméra par PC doté d'une carte Ethernet supplémentaire.

Q 5.2 : Quelles sont les parties parallélisables du calcul des cartes de disparités ?

Q 5.3 : Indiquez brièvement (10 lignes maximum) comment paralléliser le calcul des cartes de disparité sur un cluster de PC.

Q 5.4 : Indiquez les points importants à prendre en compte pour obtenir une implémentation efficace sur accélérateur (GPU ou processeur Cell), notamment en fonction de la résolution de la caméra et du calcul de disparité utilisé pour chaque pixel.

Exercice 6 : (2 points)

Q 6.1 : On considère une fonction réalisant la multiplication de deux matrices carrées A et B de n x n éléments de type double.

a) Écrivez une version séquentielle en langage C ou en pseudo-code de cette fonction.

b) Écrivez, pour un calculateur à mémoire partagée, une version parallèle, au choix en OpenMP, en MPI (Message Passing Interface), ou avec des pseudo-fonctions.

c) Quelles techniques utilise-t-on dans le cas de matrices creuses ?

Q 6.2 :

a) Soit le système linéaire, issu d'un problème de type éléments finis ou volumes finis, $Ax=b$, où A est une matrice carrée de taille n, et b et x des vecteurs de longueur n. Citez et décrivez succinctement deux méthodes, autres que des méthodes directes, pour résoudre ce système.

b) Donnez des raisons pour lesquelles les méthodes indirectes sont utilisées.

Exercice 7 : (2 points)

On se propose de développer un simulateur de chirurgie minimalement invasive, avec retour d'effort. Le principe est de présenter un modèle réaliste de la partie du corps humain ciblée, et d'interagir avec des dispositifs à retour d'efforts mimant les instruments chirurgicaux.

Q 7.1 : Pour construire un modèle réaliste, voire personnalisé, des images médicales seront utilisées. Les données étant fournies par un partenaire hospitalier, précisez les modalités, les formats et les encodages auxquels on peut s'attendre. Décrivez ensuite plusieurs modes de visualisation d'une image médicale et indiquez leur adéquation éventuelle pour un simulateur de chirurgie

Q 7.2 : Pour réaliser la simulation, il faut disposer de modèles géométriques des organes qui seront manipulés par le simulateur, modèles géométriques qui seront représentés par des maillages. Décrivez la méthodologie pour obtenir de tels maillages à partir des images médicales, et les divers choix ou compromis qui peuvent entrer en jeu.

Q 7.3 : Quels vont être les principaux points techniques auxquels il faudra prêter attention pour le réalisme de la simulation ? Décrivez un (ou plusieurs) schéma général contrôlant un tel simulateur, donnez ses principaux composants et précisez leurs interactions.

Exercice 8 : (2 points)

Une séquence d'ADN est représentée par une chaîne de lettres prises dans {a,t,g,c}, par exemple :

```
tctttgagggcgtctctgttaaagacattctccttgacttttggggcgaatg
```

Un alignement de deux séquences d'ADN est la mise en correspondance entre les lettres des deux séquences, sans changer l'ordre des séquences mais en autorisant éventuellement des insertions ou des délétions. Notez qu'avec deux séquences on ne peut généralement pas distinguer une insertion d'une délétion car cela dépend de la séquence que l'on prend en référence. Il n'y a généralement pas de raison de prendre l'une plutôt que l'autre, aussi on considèrera de façon identique insertion et délétion.

Exemple :

```
tctttaacacttgagtacattgagggcgtctctgttaaagacattctccttgacttttggggcgaatg
|||||||||||||||||||||||||!||||||||||||--|||||||!||||||||-||||||||||||
----taacacttgagtacattgagcgcgtctctgtt--agacattttccttgac-tttggggccaa--
```

Le passage d'une séquence à l'autre peut être réalisé par une suite d'opérations d'édition (ex : *remplacement, insertion, délétion*, etc.). Minimiser la suite des opérations nécessaires permet d'obtenir un alignement optimal de deux séquences.

Chaque opération d'édition ayant un coût, la *distance d'édition* est définie comme la somme de ces coûts. La minimisation des opérations d'alignement revient donc à minimiser la distance d'édition.

Q 8.1 : Nous choisissons les opérations et les coûts suivants :

- Remplacement : coût=1
- Insertion : coût=2
- Délétion : coût=2

Donnez la distance d'édition de l'exemple.

Q 8.2 : Donnez une heuristique dont le coût est proportionnel au nombre N de symboles des séquences à aligner dans leur globalité (algorithme dont le coût est O(N)). Le coût est calculé en nombre d'accès à un symbole d'une des deux listes.

Quel est, au pire cas, la distance d'édition que vous obtenez avec votre heuristique pour une séquence de 20 symboles.

Q 8.3 : Proposez une méthode d'optimisation combinatoire permettant d'obtenir la distance minimale d'édition lors de l'alignement de deux séquences dans leur globalité. Quel est son coût d'exécution en nombre d'accès à un symbole d'une des chaînes. Justifiez.

Annexe A (2 documents)
à utiliser pour traiter l'exercice 3

Document 1
Extrait de http://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html#Gcov-Intro

10.1 Introduction to gcov

gcov is a test coverage program. Use it in concert with GCC to analyze your programs to help create more efficient, faster running code and to discover untested parts of your program. You can use gcov as a profiling tool to help discover where your optimization efforts will best affect your code. You can also use gcov along with the other profiling tool, gprof, to assess which parts of your code use the greatest amount of computing time.

Profiling tools help you analyze your code's performance. Using a profiler such as gcov or gprof, you can find out some basic performance statistics, such as:

- how often each line of code executes
- what lines of code are actually executed
- how much computing time each section of code uses

Once you know these things about how your code works when compiled, you can look at each module to see which modules should be optimized. gcov helps you determine where to work on optimization.

Software developers also use coverage testing in concert with testsuites, to make sure software is actually good enough for a release. Testsuites can verify that a program works as expected; a coverage program tests to see how much of the program is exercised by the testsuite. Developers can then determine what kinds of test cases need to be added to the testsuites to create both better testing and a better final product.

You should compile your code without optimization if you plan to use gcov because the optimization, by combining some lines of code into one function, may not give you as much information as you need to look for `hot spots' where the code is using a great deal of computer time. Likewise, because gcov accumulates statistics by line (at the lowest resolution), it works best with a programming style that places only one statement on each line. If you use complicated macros that expand to loops or to other control structures, the statistics are less helpful—they only report on the line where the macro call appears. If your complex macros behave like functions, you can replace them with inline functions to solve this problem.

gcov creates a logfile called *sourcefile*.gcov which indicates how many times each line of a source file *sourcefile*.c has executed. You can use these logfiles along with gprof to aid in fine-tuning the performance of your programs. gprof gives timing information you can use along with the information you get from gcov.

gcov works only on code compiled with GCC. It is not compatible with any other profiling or test coverage mechanism.

Document 2
Extrait de http://cobertura.sourceforge.net/introduction.html

Introduction

In a recent interview, James Gosling mused: "I don't think anybody tests enough of anything" (*A Conversation with James Gosling*).
The fact is, software today is horrendously under-tested. Software maintenance costs are spiralling because few pieces of software can be changed with any degree of confidence that new bugs aren't being introduced.
Without continual change and improvement, software is unable to continue meeting its users' needs. But without comprehensive automated test-suites, such change can also expose users to very undesirable risks: the risk that critical features no longer function as before, the risk of data loss and the risk of a system crash.
Agile software development methodologies are helping improve the quality of software. Test-driven development, where tests are written to test features *before* those features are added, ensures that every piece of software is coupled with a test-suite.

Why use a coverage tool?

No matter how good such methodologies are, and how diligently they're followed in an organisation, it isn't possible to ensure that software testing is as comprehensive as it could be. That's where tools can help.
Cobertura is a free, simple and easy-to-use tool that will complement your existing Java development practices. It helps you discover exactly where your software is being tested and, more importantly, where it *isn't*. Cobertura will help you to view your software from a number of levels, from the entire system right down to an individual line of code.

Why use Cobertura?

Sure, there are other coverage tools around, so what makes Cobertura different?
Coverage analysers work by adding instrumentation. For Java, coverage analysers fall into three categories: those that insert instrumentation into the source code, those that add instrumentation to the Java byte-code, and those that run the code in a modified JVM. Cobertura adds instrumentation directly to the bytecode. We feel this is the best approach, since it does not require a modified VM, but still retains a big speed advantage over having to compile all your source code twice.
Secondly, Cobertura is easy to integrate with Apache Ant. It comes with its own Ant task definitions for you to use. You can choose to instrument any code you wish, from a single class to an entire system.
Finally, Cobertura is completely free and not time-locked, so you can begin to use it today, without having to worry about what to do at the end of an evaluation period. What's more, we believe Cobertura is so easy to use, you'll be up and running in no time. So why not spend the next 15 minutes getting up and running with Cobertura. In that time, you'll certainly find out which parts of your code are completely tested, and where your code could do with a bit more testing.

Getting started

This chapter assumes you are already familiar with the Ant build tool from Apache. If you're unfamiliar with Ant, you can find out about it at http://ant.apache.org/.

Adding the Cobertura custom tasks to Ant

Cobertura seemlessly integrates with Ant using custom tasks. But before you can use these new tasks, you have to declare them in your Ant build file, typically named build.xml. This is done using the taskdef element, as shown below. Place this element anywhere in your Ant build file.

```
<taskdef classpath="cobertura.jar" resource="tasks.properties"/>
```

Now we're ready to start using the Cobertura tasks.

Adding instrumentation to your classes

Cobertura works by inserting instrumentation instructions directly into your compiled Java classes. When these instructions are encountered by the Java Virtual Machine, the inserted code increments various counters so that it is possible to tell which instructions have been encountered and which have not.
You instruct Ant to create instrumented versions of your classes using the instrument task. The example below assumes your classes are in the directory build/classes.

```
<cobertura-instrument todir="build/instrumented-classes">
     <fileset dir="build/classes">
       <include name="**/*.class"/>
     </fileset>
</cobertura-instrument>
```

Running an instrumented application
Once your classes have been instrumented by Cobertura, you can continue testing your application as you would normally via the JUnit task.
Simply include a classpath entry for the instrumented classes *before* any reference to the original classes. This will ensure that the instrumented classes are loaded in preference to the original classes.

```
<junit fork="yes">
  <classpath location="${build.instrumented.dir}"/>
  <classpath location="${build.classes.dir}"/>
</junit>
```

The instrumented classes found in *${build.instrumented.dir}* will be loaded before those found in *${build.classes.dir}* ensuring that the instrumented classes are used by JUnit.
An instrumented class serializes information into the file cobertura.ser. Any existing information found in this file will be merged with the current information. In this way the instrumentation for several sessions of a running program can be merged together, producing a single coverage report. For example, the instrumentation from unit and functional tests can be merged together to produce a single coverage report.

Cobertura reporting
An instrumented class serializes coverage information to the file cobertura.ser. Using the report tag, Cobertura can generate coverage reports in either HTML or XML format.

HTML coverage report
The default format for a Cobertura report is HTML.

```
<target name="coverage">
 <cobertura-report srcdir="${src.dir}" destdir="${build.coverage.dir}"/>
</target>
```

XML coverage report
The type of report generated is controlled by the format attribute of the report tag, which may be either *html* or *xml*. If the format is not supplied then it defaults to *html*.

```
<target name="coverage">
      <cobertura-report format="xml" srcdir="${src.dir}"
                                destdir="${build.coverage.dir}"/>
</target>
```

Covertures session merging
Sometimes it is necessary to merge several Cobertura sessions together, for example, to produce a single consolidated coverage report from several different test runs of an application.

Session merging with the merge tag

```
<cobertura-merge>
  <fileset dir="${basedir}">
    <include name="**/cobertura.ser"/>
```

```
      </fileset>
    </cobertura-merge>
```

The above fragment from an ANT build file, will merge together any serialised instrumentation that has been generated by Cobertura and produce a single consolidated instrumentation record.

Taking control with Cobertura and check

All too often testing is something that is left until the end of the development cycle. For example, a coverage report is run on a weekly basis over the codebase, only when the coverage metrics fall below a certain pain threshold are the developers directed to increase their test coverage and quality.
With Cobertura, development teams can choose to enforce *test driven development* on their codebase, by using the cobertura-check tag in their ANT build scripts.
After each *instrumented* unit test sequence, Cobertura can do a *health check* to ensure that the codebase is being tested to the standard that has been demanded by the team. If coverage standards fall below the criterea set by the team, cobertura-check will fail the build.
Our intention with cobertura-check is to ensure that test driven development practices are being followed by the entire development team. Inadequate testing will result in an immediate *automated* build failure, without having to study the entire coverage report.

```
<cobertura-check branchrate="80" linerate="80">
    <regex pattern="com.example.gui.*" branchrate="85" linerate="90"/>
    <regex pattern="com.example.tool.*" branchrate="55" linerate="80"/>
    <regex pattern="com.example.util.*" branchrate="85" linerate="95"/>
</cobertura-check>
```
