Parallel Programming and Distributed Computing - Assignment 2

Dr. Tim Conrad
Freie Universität Berlin
Sommersemester 2013

1 Cover sheet

Student details

Matrikel Nr: Full Name:

Submission date and extensions

Due date: 21.04.2013, 23:59

Date submitted Signature

All work must be submitted by the due date. If an extension of work is granted this must be authorised on this form with the signature of the lecturer or tutor. Extension granted until:

Lecturer/tutor Signature

Plagiarism and collusion

Intentional plagiarism amounts to cheating.

Plagiarism - Plagiarism means to take and use another person's ideas and/or manner of expressing themselves and to pass these off as one's own, failing to give appropriate acknowledgement. This includes material from any source, staff, students or the internet - published and unpublished works.

Collusion - Collusion is unauthorised collaboration with another person or persons.

Penalties - If there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Chief Examiner, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Manager.
Student statement and signature

- This assignment is original and has not previously been submitted as part of another unit/subject/course
- I have taken proper care of safeguarding this work and made all reasonable effort to ensure it could not be copied
- I acknowledge that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and provide it to another member of faculty
- I understand the consequences for engaging in plagiarism as described in section 2.8.
- I certify that I have not plagiarised the work of others or participated in unauthorised collusion when preparing this assignment.

Student’s name / Date ______________________ Signature ______________________

Please note that it is your responsibility to retain a copy of your assignment.

Privacy Statement

The information on this form is collected for the primary purpose of assessing your assignment. Other purposes of collection include recording your plagiarism and collusion declaration, attending to administrative matters, and statistical analyses. If you choose not to complete all the questions on this form we may disallow the submission of your assignment.
2 Assignment

2.1 Aim

The aim of this assignment is to write parallel programs and use Java with the MPI package. All programs will tackle the same problem, finding a shortest path to a cookie in a maze. Mazes provided to your program can have varying sizes and layout. The maze is represented by a grid of cells. Each cell of the grid can be the entrance, the cookie, part of a wall or open space that could form part of a path. There is only one entrance in the outside wall (and it can’t be on a corner). There is only one cookie. Walls cannot be climbed or looked around. There are no diagonal "moves", only north, south, east and west. You are aiming to get as much parallelism out of the cells as possible but still to have an efficient algorithm in terms of overall parallel time.

Your program reads the maze definition from standard input and should output the maze definition and a shortest path on standard output.

There is a tcl script, maze.tcl to view progress of your maze search and help debug your program.

2.2 Solution idea

In graphics, the inside of a 2-D object can be found by flooding the inside with colour and seeing which of the cells are coloured. You can adapt this method to determine shortest paths to the cookie. You may represent colour flooding by increasing integer values in cells representing to shortest distance to the entrance discovered by flooding so far. Cells are only ever updated monotonically i.e. by increasing or leaving the value as is. You may adapt this idea to the problem by imagining that each cell tries to maintain a local invariant expressed by an equation defining its current value as the minimum of visible neighbouring colours plus 1. Updates could occur randomly or when neighbours change. Updating can end when no flooding occurs any more. The color flooding could then start at the entrance with color 0 which is never updated.

2.3 Formal method

Monotonic updates like the above, define an update function \( f \) per cell \( X \), such that \( x_{n+1} = f(x_n) \) with \( x_0 \) the initial value of the cell and \( x_n \) its value after the \( n \)-th update. This update function can be extended to a function \( F \) on the array of cells (the maze) \( A \), such that in each step the maze is updated in exactly one cell (using the function \( f \) for that cell). In other words if \( F(A) = A' \) and \( A \) and \( A' \) differ, then they differ in one cell.

If \( f \) is monotonic, then so is \( F \) and a solution to the problem is a fixpoint of \( F \), i.e., \( F(A) = A \), where \( A \) is the maze colouring (for example the array of colour values).

A sufficient criterion for the existence of such a fixpoint, given monotonically increasing \( F \), is the existence of an upper bound on the possible cell values. Since our arrays are finite, such a bound is trivially the number of cells.

Dually monotonically decreasing functions have the same fixpoint properties.
2.4 Application of Formal Method

First you need to design your solution such that you encode possible cell values by suitable integer values (the colors). Cells would be initialised by some distinct value for 'wall', 'unknown' etc. Walls are considered larger than any possible distance to the entry. As reachability of the exit is discovered, concrete integers are used to mark cells. Cells are only updated when a shorter distance than the current one is determined (hence the update function is monotonically decreasing).

On parallel machines you can solve such fixpoint problems by assigning possibly but not necessarily overlapping cells regions to different processors (for update) and use an asynchronous update. This means, for update processes you need not synchronise unless there is an efficiency gain. However you still need to communicate changing cell values between the processor. You could block or stripe the array. Consider efficiency issues when choosing how to partition the problem. For efficiency the trick is to find a good balance between maximising parallelism, minimising communication overhead, and finally, minimising the maximum number of steps any processor executes before the program overall reaches the fixpoint \( F(X) = X \), i.e., the solution.

2.5 Tasks

Your assignment is as follows:

1. AsyncMaze.java (12 marks): Implement the maze search using asynchronous computation in Java. All threads execute the same function. Use the method above to devise a prototype solution with eight (8) threads.

2. MPIMaze.java (8 marks): Port your maze solution to the cluster and add MPI instructions to it to work with the Java MPI binding. In this part you are required to change your program to an SPMD solution using MPI communication primitives. Different designs are possible ranging from Peer-To-Peer to Collection communication. In either case you need to think about how to distribute your data to make your solution reasonably efficient.

2.5.1 Testing

Different mazes and a script to display the maze simulation are available here:

Go to the Website [http://biopatterngroup.de](http://biopatterngroup.de) → Teaching → “Parallel Computing and Distributed Computing” → “Files”

This script will also be used when we mark your assignment.

Test your solution at least with the 8 mazes provided in this file.

2.5.2 File names

Your files must be called AsyncMaze.java and MPIMaze.java. Note that case matters!

You may lose significant marks if your solution does not follow the required file name, class names and/or I/O specification. We are marking your assignments
using compile and test scripts under Linux. Filenames and their case matter. We will not mark a solution that does not compile because of file name problems. For example, name your files *.java, not *.JAVA.

If you develop under MS Windows, you must at least test your solution under Linux before you submit.

2.6 Description (5 marks)

We want you to

1. Write a short description of the search algorithm, the threads (in the async maze) and the MPI strategy (in the MPI maze) you have used and how they are connected. This description should not exceed two pages.

2. To perform benchmark tests (i.e. time your program takes to run) with 2, 4, 8 and 16 (if possible) nodes (for an example see Figure 1).

![Figure 1: Example benchmarks for 8 nodes. Note that different sections of a program run are measured.](image)

2.7 Submission

You need to submit both the source code (electronically) and the description (electronically as PDF).

2.7.1 Source Code

Only electronic submission via SVN will be marked (no eMail submission!).

2.7.2 Description

You must submit your description electronically (in PDF or RTF format) within your source code archive (see above). You also need to hand in the filled in cover sheet (see section 1) at the next seminar meeting.
2.8 Penalties

Late submissions will lose 10% of their marks for each new day beginning once the deadline has passed. A weekend counts as one day. Also be sure to include the provided header text you will find below in each source code file you submit:

```c
/*
   ********************************************************************************
   * FREIE UNIVERSITAET BERLIN, Fachbereich Mathematik und Informatik
   * Erklaerung des Studenten zur Einreichung einer Uebung / Projektes
   * im Kurs "Verteiltes Rechnen und Parallelprogrammierung" im SS13
   *
   * Author: [Name], [Vorname]
   * Student: [Matrikelnummer]
   * Email: [FB eMail Adresse]
   * Created: [Datum der Erstellung dieser Datei]
   *
   * Ich, [Name], erklare hiermit, dass diese Einreichung von mir
   * personlich erstellt worden ist und nicht (auch teilweise) aus einer
   * anderen Quelle kopiert wurde ohne darauf im Quellcode hinzuweisen.
   * Mir ist bewusst, dass Verstoesse gegen diese Regeln Konsequenzen
   * nach sich ziehen, bis hin zu einer Bewertung der Einreichung mit
   * 0 Punkten.
   ********************************************************************************
   */
```

We run random plagiarism tests in this subject. Detection of plagiarism results in 0 marks for the assignment.

3 Auxiliaries

The tcl display script will help with your debugging. It reads from standard input looking for lines to control the display as follows:

- `entrance row# column#`
- `cookie row# column#`
- `wall row# column#`
- `cell row# column# value`
- `path row# column# value`

Other lines are simply passed to standard output.

Typically, input lines given to the tcl script first display the maze outline using entrance, cookie and wall lines; then as individual cells are updated, additional lines are given as input using cell lines; and finally a path is indicated by tracing back from the cookie when it is found using path lines. For example the following command

```
./maze.tcl 30 30 0 < maze.demo
```

displays maze.demo on a 30 by 30 grid (first two arguments) with 0 delay after each step (third argument).

The required files can be found here: [http://www.msproteomics.org/index.php?option=com_docman&task=doc_download&gid=13](http://www.msproteomics.org/index.php?option=com_docman&task=doc_download&gid=13). You may have to change the hash-bang line of the tcl script depending on the location of wish on your machine.