Assignment #1 – Functional Programming for Parallel Processing

Introduction

This assignment aims to consolidate your understanding of several topics taught in this course, such as applied functional programming and parallel /distributed processing. You are required to build and evaluate several parallel versions of a simple but fundamental algorithm, Longest Common Subsequence (LCS). This algorithm is very important, e.g. in bio-informatics, and efficient parallel versions are still sought and proposed. For our purpose, we only require the LCS length, not the longest sequences.

Overview

The assignment has two required parts and an optional part (for a possible bonus).

1. Required: three basic LCS versions, based on a space-aware dynamic programming (DP) model.

1.1 Sequential implementation (SEQ), used as a baseline to evaluate the parallel speedup. Long input strings are expected, so you need to implement a diagonal wavefront, instead of the textbook array approach.

1.2 Parallel implementations (PAR), following a coarse granularity model. The conceptual array is divided into rectangular sub-arrays, further allocated to actors (possibly on different machines).

Each actor works sequentially, using exactly the sequential implementation SEQ. Thus, each actors evolves on its own internal diagonal wavefront. There is also one global diagonal wavefront which activates the actors. There are two versions of this global activation:

1.2.1 Parallel synchronous (PAR-SYNC): a strict multi-phase algorithm, with a barrier which ensures that all actors on the same diagonal complete before the next set of actors starts.

1.2.1 Parallel asynchronous (PAR-ASYNC): a relaxed algorithm, where any actor can start as soon as he receives the needed boundary data (no need for barriers or any other form of synchronisation).

The figures (1, 2, and 3) schematically illustrate these three cases. Green areas indicate active cells, currently on a diagonal wavefront. Yellow areas indicate already processed cells. White areas indicate unprocessed cells. Red lines indicate boundaries between actors.

2. Required: A report with literature evaluation and empirical evaluations of your results (more details below)

3. Optional (OPT): suggestions:

3.1 A Multiple LCS (MLCS) version, for the LCS of 3 strings (or more); or

3.2 An advanced LCS version, at your choice, with a faster parallel implementation; or

3.3 A container-based implementation of your required parallel versions, where each actor is allocated its own container.

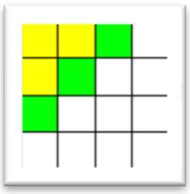

Figure 1: SEQ

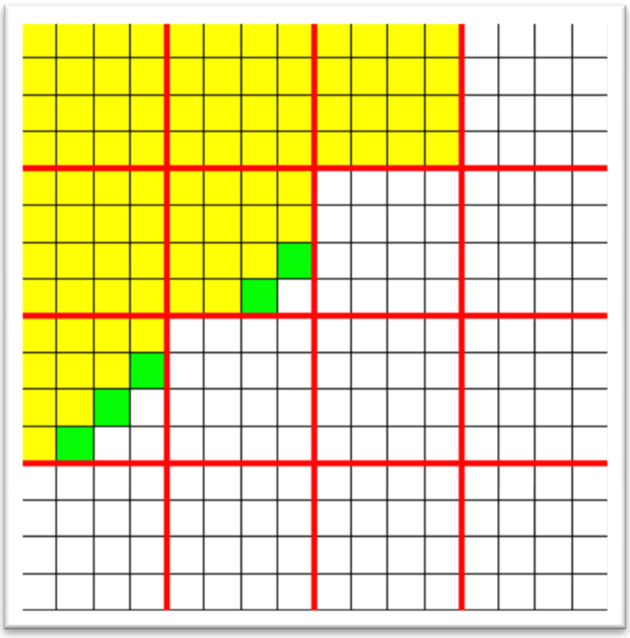

Figure 2: PAR-SYNC

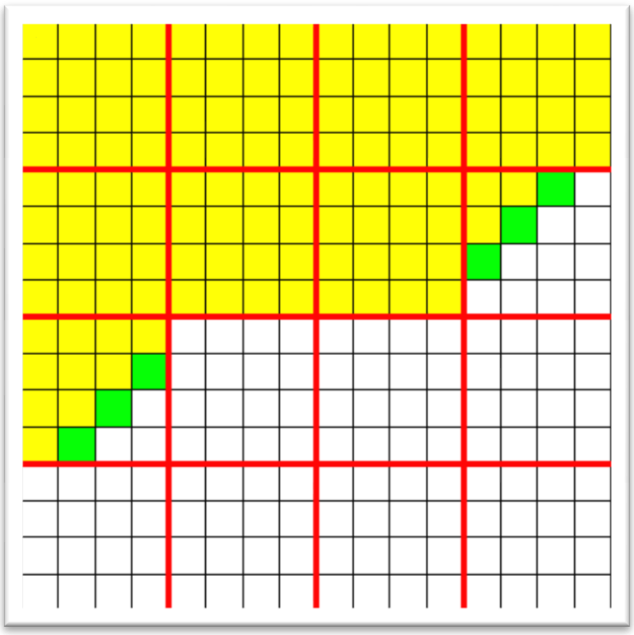

Figure 3: PAR-ASYNC

Development Tools

For all implementations, use F# and Akka.NET actors. You can develop your program with Visual Studio, if you wish so. However, your submission must only include F# .FS source files plus a batch file to compile your program, which should be called A1.EXE (for consistency with the marking tools).

For the optional containers part, use Docker containers and images based on F# and Akka.NET.

Running the Program

Please strictly follow these conventions (for consistency with the marking tools)!

1. Invoking the program – one of the following lines:

```
A1.EXE  /S1:path-to-file1  /S2:path-to-file2  /SEQ

A1.EXE  /S1:path-to-file1  /S2:path-to-file2  /PAR-SYNC:b1,b2

A1.EXE  /S1:path-to-file1  /S2:path-to-file2  /PAR-ASYNC:b1,b2
```

Where

- **file1** and **file2** are .TXT files using the UTF8 encoding and containing the two input strings, of lengths **n1**, **n2**, respectively, s.t.: 1 <= **n1**, **n2** <= 100000
- **b1** and **b2** are strictly positive numbers (1 <= **b1**, **b2** <= 100) which indicate the required number of horizontal and vertical bands, respectively
  - these requirements and the bands' sizes may need to be adjusted in a good balanced way, if the lengths of the input strings are not exactly divisible by these numbers
  - **n1** = 6, **b1** = 3 => 3 horizontal bands of lengths [2; 2; 2]
  - **n1** = 7, **b1** = 3 => 3 horizontal bands of lengths [3; 2; 2]
  - **n1** = 8, **b1** = 3 => 3 horizontal bands of lengths [3; 3; 2]
  - **n1** = 9, **b1** = 3 => 3 horizontal bands of lengths [3; 3; 3]
  - **n1** = 3, **b1** = 3 => 3 horizontal bands of lengths [1; 1; 1]
  - **n1** = 2, **b1** = 3 => 2 horizontal bands of lengths [1; 1]
  - **n1** = 1, **b1** = 3 => 1 horizontal band of lengths [1]
  - **n1** = 7, **b1** = 1 => 1 horizontal band of length [7]
  - most of the assessments will use

2. Standard output: one single integer, the LCS length – only this number!

3. Error output: may contain any additional traces which you want. However, these may affect the runtime, so you should disable these for the final submission.

Report

Your report should be structured and written like a research article. It should contain: title, author, abstract, introduction, good and updated literature overview, brief descriptions of your algorithms, clear empirical evaluations of the runtimes of your implementations (with tables and plots), conclusions, and bibliography.

As indicated by the highlights, the focus should be on the literature evaluation and your own empirical evaluation. Note that you can start working on your report from day one: only the empirical evaluation and conclusions need to wait for your results.

Suggestion to use a good and "standard" format for scientific publications, such as the LNCS article style, either in LaTeX (preferable) or Word.

Deliverables and Submission

Submit electronically, to the new COMPSCI web dropbox, an upi.7z archive containing an upi folder with:

- your report as PDF;
- your F# source file(s);
- a _COMPILE.BAT batch file to compile your file(s) into an A1.EXE executable.

Please keep your electronic dropbox receipt and follow the instructions given in our Assignments web page (please read these carefully, including our policy on plagiarism):

http://www.cs.auckland.ac.nz/courses/compsci734s1c/assignments/

Deadline

Monday 25 April 2016, 18:00! Please do not leave it for the last minute. Remember that you can resubmit and, by default, we only consider your last submission.

After this deadline, submissions will be still accepted for 5 more days, with a gradually increasing penalty of 0.5% for each hour late. For example:

Monday 25 April 2016, 20:00:	-1%
Tuesday 26 April 2016, 18:00:	-12%
Wednesday 27 April 2016, 18:00:	-24%
Thursday 28 April 2016, 18:00:	-36%
Friday 29 April 2016, 18:00:	-48%
Saturday 30 April 2016, 18:00:	-60%

After this, no more submissions are accepted!

Appendices

A#1-LCS-Demo.xls: A simple intuitive Excel demo (recall that Excel is a functional language)

A1 Tests: Correctness tests with several small and medium sized samples, using a simple SEQ 2D implementation

Barrier: A sample which implements several versions of an async barrier and compares it against the standard library sync Barrier

GridActors: A simple model of a rectangular grid of F# Mailbox actors, with a grid dataflow running from N to S and W to E

GridDiag: A sample which demonstrates a diagonal wave sweeping over a rectangular array

Model performance: To be posted, only as a rough guideline

Assement criteria

- **Functional correctness**: [**black box** style] SEQ and PAR will be assessed on small and medium sized samples (similar to the samples give in **A1 Tests**)
- **Performance medium**: [**black box** style] SEQ runtime and PAR speedup, on medium sized samples
- **Performance large**: [**black box** style] SEQ runtime and PAR speedup, on large sized samples
- **Special cases**: [**black box** style] incorrect command lines (exceptions must be caught and displayed on standard error), band numbers = 1, string sizes not exactly divisible by band numbers, bands with one cell only
- **Actors**: [**white box** style] partial marks for using MailboxProcessor actors for the main dataflow; full marks here for using Akka.NET actors
- **Report**: see the description above; ensure that your report looks readable and provides convincing discussions, on results which are reproducible on a typical lab machine (illustrate it with your own runtime results, even if the performance is not great)

Notes:

- Even when not explicitly stated, each item receives partial marks in the range from 0 up to the maximum allocated
- **Functional correctness** is critical! All following items depend on this and their scores are weighted by the functional correctness relative score!
- For **performance tests**, the full marks threshold will be based on a combination between the model results and the class results, s.t. projects may get full marks even if their performance is lower than the model solution performance
- The **black box** tests do not look into your internal implementations…
- Thus, for small and medium sized samples, there is no penalty for using 2D arrays or sub-optimal algorithms (as long as the correct result is returned within a reasonable time-out period)
- However, the large samples will be more serious stress tests…
- **Bonuses**: see the optional items described in the specs. Additionally, the scalability of the top submissions will be tested on a supercomputer with 16 cores. Please contact us, if you have in mind additional options

Draft breakdown - out of 100 assignment marks

Functional correctness	20; **FC** = actual score / total (20)
Performance medium	20 * **FC**
Performance large	10 * **FC**
Special cases	10 * **FC**
Actors	20 (10+10) * **FC**
Report	20 * **FC**