ASSIGNMENT 1

CSC3021 Concurrent Programming 2023-2024

Submit by 5:59 pm on Monday 10th October 2023

Total 40 marks. Counts towards 30% of marks on CSC3021.

Question 1: Process Interleaving: Straight-Line Code [15 marks]

Consider the following program:

```
integer n := 1, k := 1
```

```
Process P
P1:  n := k + 1
end P;
```

```
Process Q
Q1:  k := n - 1
end Q;
```

Analyse the above code and answer following questions:

- Consider whether all statements are atomic and transform the program to load/store architecture if needed.
- Identify the conflicting pairs of atomic instructions.
- State an upper bound on the possible number of outcomes for this program based on the number of processes and atomic instructions in this program.
- State an upper bound on the possible number of outcomes for this program based on the number of conflicting pairs found.
- What are the possible final values of **'n'** and **'k'**? Justify your answer.

Question 2: Process Interleaving with Control Flow [10 marks]

Consider the following program:

```
integer n := 1
boolean flag := true
```

```
Process P
P1:  while( n == 1 ) {
          ; // nothing
     }
P2:  flag := false
end P;
```

```
Process Q
Q1:  while( flag == true ) {
Q2:       n := 1 - n
     }
end Q;
```

Answer following questions:

(i) Consider whether all statements are atomic and transform the program to load/store architecture if needed.
(ii) Identify all conflicting pairs of atomic instructions.

(iii) Construct an interleaving of instructions of the processes `'P'` and `'Q'` for which the program terminates.
(iv) What are the possible values of `'n'` when the program terminates? Can you argue that other values than those you have listed for `'n'` are impossible?
(v) Does the program terminate for all ways to interleave instructions assuming a fair scheduler? Does the program also terminate for execution scenarios that are unique to an unfair scheduler? Justify your answers. (Reminder: with a fair scheduler no process is deferred forever).

Question 3: Graph Processing - Introduction [15 marks]

This question lays the foundation for the remaining assessments on the implementation of concurrent graph processing operations. In this assignment you will complete the implementation of a graph processing program. A skeleton program is available in a git repository. [1] You will extend the code by completing the implementation of key data structures to represent graphs.

The code models graph analytics applications as a repeated graph traversal. Each traversal operates like a higher-order *map* method: all edges of the graph are visited and a user-supplied method is called for each edge. This method performs a function specific to the graph analytics. For instance, in PageRank it consists of a floating-point addition, while the label propagation algorithm performs a 'minimum' operation on integers. The actual operation performed is not of immediate interest to us. We will focus our attention on completing the graph data structures. An accompanying document, available on canvas, gives additional information on the PageRank problem. It is useful to familiarise yourself with the PageRank problem for debugging and validation purposes.

The code may operate using one of three graph data structures (CSR, CSC or COO, defined in the slides). Implement the omitted code for calculating out-degrees (an auxiliary method required by PageRank) and for the edgemap method for each of the COO, CSR, and CSC sparse matrix formats. You only need to modify the files SparseMatrix{COO|CSR|CSC}.java. Keep the command-line interface of the Driver.java program as is and keep the damping factor at 0.85. Test your code thoroughly as you will extend it in the next question. Confirm that all three versions calculate the same PageRank values, up to an error tolerance of 10^{-7}.

Perform following experiments using your implementation and the orkut_undir[2] graph and briefly answer these questions:

(i) Record the residual error (printed to the console by the PageRank algorithm) at the end of each iteration of the power method for each of the three matrix formats. Plot these values in a chart (consider log-scale). Summarise your observations.

[1] https://hpdc-gitlab.eeecs.qub.ac.uk/hvandierendonck/csc3021_assignments_2324
You cannot login to this git server; the repository is publicly accessible. You can clone this repository and work off it. Check the README.md file for instructions.
[2] http://www.eeecs.qub.ac.uk/~H.Vandierendonck/CSC3021/graphs

(ii) The memory systems in modern computers are highly optimised towards common patterns in computer programs. Most data is accessed with high locality (a small set of data is repeatedly accessed before the program moves on to other data) which can be captured by hardware caches. Sequential accesses (e.g. accessing array elements a[0], a[1], a[2], …, a[n-1] in a large array) can be efficiently read-ahead by hardware prefetchers such that the main memory latency is hidden. "Random" access patterns where there is no apparent relation between subsequently accessed memory locations is very challenging and a markedly lower execution rate is achieved on those.

Analyse your code and describe for each array in each of the CSR, CSC and COO data structures, as well as the pagerank arrays with old and new values, which array is accessed using a sequential access pattern, and which are apparently random.

Why do you think the distinction between these access patterns matters for concurrent programs?

CSR	Pattern
index	
destination	
x	
y	

CSC	Pattern
index	
source	
x	
y	

COO	Pattern
source	
destination	
x	
y	

(iii) Record the execution time taken by the PageRank algorithm for each of the three graph data structures (CSR, CSC and COO). Plot these in a chart and describe your observations. Can you explain why some formats result in lower execution time than others?

(iv) Modify your code to use single-precision floating-point numbers instead of double-precision floating-point numbers for **one** of the three sparse matrix formats (e.g., perform a search-replace operation in the PageRank.java and Driver.java files). Record the residual error at the end of each power iteration step and the average execution time when using single-precision vs. double-precision floating-point numbers. Report these numbers using charts. Can you explain this result?

Submission

Your submission will consist of three parts:

- **Write-up:** A write-up containing your responses to Q1 and Q2; and a discussion of your findings for Q3; to be submitted through Canvas. When plotting charts in response to Q3, take care to present them carefully, clearly label all components, scale them properly so they fit on the page, and adjust font sizes and colours to be readable. Unfortunately, neither Microsoft Word nor Excel come with good default settings.
  Make a declaration at the start of the document on your use of Generative AI, or state you haven't used it.

- **Java source code:** Your completed code for Q3 submitted to a **private** git repository on gitlab2.eeecs.qub.ac.uk.
  Provide access to the users h.vandierendonck@qub.ac.uk, yun.wu@qub.ac.uk, stauhidi01@qub.ac.uk, vskouroliakou01@qub.ac.uk, b.dandurand@qub.ac.uk, mdantonio01@qub.ac.uk. When you add these users, they will be emailed about the details of the repository.
  Make sure to make frequent commits to the repository as the commit history can inform the marking in case your program is not fully functioning.
- **Online testing:** The same code submitted to an online testing and evaluation system available at http://domjudge.hpdc.eeecs.qub.ac.uk/ using the contest '2324_A1'. This system will test the correctness of your code in a few sample situations by checking the correctness of the computed values. The relative ranking of your solution compared to others is irrelevant for this assignment. The site will be available soon and you will be informed of your login credentials for this site in due time.
  The system uses the following version of Java:
  ```
  openjdk 11.0.18 2023-01-17
  OpenJDK Runtime Environment (build 11.0.18+10-post-
  Debian-1deb11u1)
  OpenJDK 64-Bit Server VM (build 11.0.18+10-post-
  Debian-1deb11u1, mixed mode, sharing)
  ```