Practical Assignment

The main objective of this practical assignment is to use the capabilities of parallel programming through the memory sharing model to solve a computational calculation problem.

Problem Description

Consider the Job-Shop problem, which is the allocation of resources to carry out a job[1]. In the Job-Shop problem there are several machines that can carry out operations. In order to produce a job, it is necessary to carry out a set of operations, by sequence, on several machines. As the machines can't carry out all the operations, each job is a sequence of operations, where each operation has to be carried out on a specific machine.

In addition, because each job is different from the others, the sequence of machines for each job may not be the same as the other jobs, i.e. each job has its own sequence of machines. However, the order of the machines for each job must be respected, i.e. an operation of a job that must be carried out on a specific machine can only start when the previous operation of that same job has already finished; just as the next operation of the job can only start after the current operation has finished.

Finally, performing an operation on a machine takes time. Each operation in each job takes a specific amount of time, which can be different for each operation. A machine can only do one operation at a time, so when it starts an operation, it is only free for new operations once it has finished.

This identifies the two constraints of this problem:

1) The operations of each job must be carried out in order. The second operation of a job can only begin after the first operation of the same job has been completed, and so on for all subsequent operations of the same job;

2) a machine can only perform one operation at a time, so if there are two operations for the same machine, one can only start after the other has finished.

A characteristic of the problem, which simplifies the restrictions, is that a job does not depend on the other jobs, i.e. the jobs are independent of each other. Therefore, the operations of one job do not interfere with the operations of the other jobs, as long as the machine to which they are assigned is free.

[1] See: https://developers.google.com/optimization/scheduling/job_shop

Objective

The aim of the problem is to come up with a valid scheduling, i.e. to define a start time for each of the operations (of all the jobs) in order to fulfil the constraints, i.e. an operation only starts after the previous one has finished (for the same job), and a machine is only busy with one operation at a time.

After finding a valid solution, the second objective is to optimise (reduce) the total scheduling time. The total scheduling time, known as Makespan, is defined as the time needed to complete all the operations in the scheduling plan.

A problem can have several different solutions which are all valid and which can have a different total scheduling time. The worst possible case can be defined as the time needed to execute all the operations, one at a time, until they are all completed. The best possible time may not be easy to find; however, it is known that it can never be less than the time it takes to execute any of the jobs individually (because one operation can never start before the previous one, for the same job, but jobs are independent).

The algorithmic solution must present a better result than the "worst possible solution", but it doesn't have to be the best possible.

Input Data and Results

The application receives as input the number of machines and jobs, and a table identifying the machines capable of carrying out each of the operations for each job, as well as their duration. The format of this input can be represented by a text file with a specific format[2], for example the Fisher and Thompson 6x6 list[3]. See Annex I.

The following table shows an example of the input data for the problem presented on the web page of Google's OR-Tools application[4]. The values in the table represent the pair (<machine, duration time>) for 3 jobs with up to 3 operations each on 3 machines (identifiers start at 0).

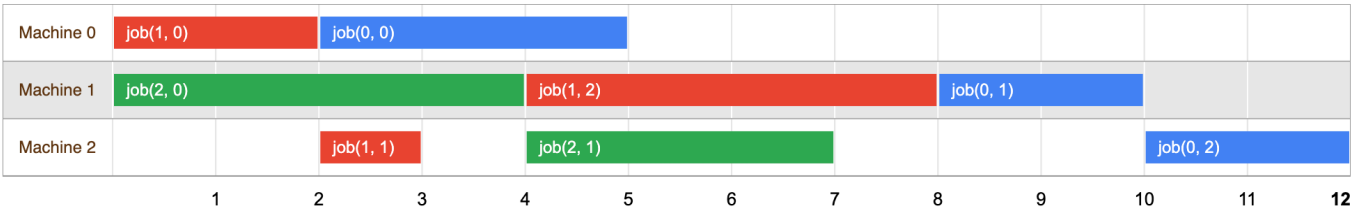
Jobs	Operation1	Operation2	Operation3
job0	(M0, 3)	(M1, 2)	(M2, 2)
job1	(M0, 2)	(M2, 1)	(M1, 4)
job2	(M1, 4)	(M2, 3)	*(M0, 1)*

[2] See: https://ptal.github.io/scheduling-data.html#job-shop-scheduling-problem
[3] See: https://github.com/ptal/kobe-rcpsp/blob/master/data/jssp/ft/ft06.jss
[4] See: https://developers.google.com/optimization/scheduling/job_shop

The application must produce as a result a list that identifies the start time of each operation, for all the operations of all the jobs. See Annex II. The following table shows the result of scheduling, where each operation of each job has a start time.

Jobs	Operation1	Operation2	Operation3
job0	2	8	10
job1	0	2	4
job2	0	4	7

This list can be grouped by machine (see Google image).



The result of scheduling is a list of the operations that will be carried out on each machine, with the start time for each one. In addition, the maximum completion time, at which all operations are completed, must be indicated in the output file. In the previous figure, the Makespan, the last operation finishes at time 12. Annex II shows the format of the output of the application.

Approach to Allocating Start Times

There are several possible approaches to this assignment. The simplest method of assigning start times is to make a sequential traversal for all the operations of all the jobs, and assign a start time to all the operations, respecting their duration, regardless of the machine or job to which they belong. This approach corresponds to the sequential case, which is also the longest possible, without gaps, in which one operation and only one operation is carried out at each instant of time.

There are other possible approaches to time allocation, which are valid as long as they take into account the constraints of the problem, such as Branch & Bound or Shifting Bottleneck.

An approach that manages to find the best optimal solution to the problem, i.e. the combination that takes the least time of all the possible ones, is called Branch & Bound[5][6]. This approach builds a search tree, where it records the various solutions found and their result. At the end of the algorithm, the node in the tree with the best solution is chosen, i.e. the one with the lowest completion time for the last operation. Each node in the tree represents a potential source of parallelism, since its calculations can be carried out independently of those of the other nodes in the tree.

[5] See Pinedo, 2012, *Job Shop Branch & Bound Slides*, https://web-static.stern.nyu.edu/om/faculty/pinedo/scheduling/shakhlevich/handout11.pdf
[6] See Brucker, 1994, *A branch and bound algorithm for the job-shop scheduling problem*, https://www.sciencedirect.com/science/article/pii/0166218X94902046

Finally, some papers have been published on parallel implementations of the Branch & Bound method[7], some of which address the job-shop problem[8][9].

These works can be considered, but their implementation is not mandatory; they should be considered as references.

Restrictions in Implementing a Solution

The choice of algorithm that the group wish to use is free. The only restriction required for implementation is **not** to use pointers on dynamic structures. The implementation of graphs or lists should be based on arrays (without pointers) and not linked lists or equivalent (with pointers). If the group so wishes, it can use a single initial pointer for an array, but it is not permitted to use internal pointers in the structure. Several arrays can be used.

Delivery

The delivery of this practical work will consist of a report in which you must present an explanation for each of the following topics, as well as an extract from the relevant code for that topic. The report must be in **PDF** format.

All the files (report + source code of the two applications + auxiliary files) must be sent in a **ZIP** file, with the identification of the group members, number and name.

A. Sequential Implementation (5 vals)

1) Describe the algorithm chosen to assign the start times of operations.
2) Present a sequential implementation of the approach.
    - This implementation must receive two parameters on the command line: a file with the input data; and a file to save the result.
    - The implementation must fulfil the constraints of the problem in the generated solution.

[7] See: Barreto & Bauer (2010), Parallel Branch and Bound Algorithm - A comparison between serial, OpenMP and MPI implementations
https://doi.org/10.1088/1742-6596/256/1/012018
[8] See: Perregaard & Clausen (1998), Parallel branch-and-bound methods for the job-shop scheduling problem,
https://link.springer.com/article/10.1023/A:1018903912673
[9] See: Steinhofel et al. (2002), Fast parallel heuristics for the job shop scheduling problem,
https://www.sciencedirect.com/science/article/pii/S0305054800000630

B. Parallel Implementation with Memory Sharing (11 vals)

1) Following Foster's methodology, present a partitioning proposal, a division of the problem into groups of instructions and data, taking into account the approach used.
    - The algorithm to be used should be identified and preferably be the same as the sequential implementation.
    - Identify the group of instructions that will be executed by each of the threads.
    - Describe the communication pattern used between the threads to exchange information.

2) Present a parallel implementation of the approach using shared memory.
    - This implementation should receive two parameters on the command line: a file with the input data; and a file to save the result.
    - In addition, it should receive the number of threads as an input parameter.
    - Compare and comment on the results of the simulation between the parallel version and the sequential version, in particular the start times of the operations and the completion time of the last operation performed.
    - The report should identify:
        i. the data structure used in the program;
        ii. initial start-up and final code for the program's threads;
        iii. code executed by the various threads;

3) Identify the following characteristics of the parallel program:
    - which global variables are shared (read-only and read/write), and the local (private) variables for each thread;
    - the critical sections (sections of code in the parallel program) which can create race condition situations, and the variables involved;
    - which mutual exclusion techniques are used to guarantee the correctness and determinism of the program, if any.

C. Performance Analysis (4 vals)

1) To measure the performance of the previous implementations, add a runtime measurement mechanism to the code.
    - During the running time measurement there should be no output or input to the console. All data should be kept in memory and can be load from/to an external text file before/after the run.
    - The calculation of execution time in the parallel version should include the creation and termination of threads.
    - The collection of execution time for each application should be an average or sum of several executions, configurable (for example: 10 repetitions), rather than a single execution. This repetition mechanism should be embedded into the executable.

2) Record the program's execution time in a table for the various thread numbers (consider as examples: SEQ, PC1, PC2, PC4, PC8, PC16, PC32).

    The input file should be chosen to generate a computation load with at least 1 minute of execution in the sequential version.
    - Use the same input data for the concurrent version. Compare the lists of results, in particular the order of execution of the operations in time and the longest time taken for the last operations to finish.
    - Present a graph using the number of threads on the XX axis and the execution time on the YY axis.

3) Compare the performance of concurrent execution with sequential execution by calculating the Speedup (S) value (ratio between sequential execution time T_1, and parallel execution time T_p for *p* threads).
    - Indicate the number of processors present in the machine used for the results.
    - Construct a graph showing the evolution of S as a function of the value of p (for the YY and XX axes respectively).
    - Comment on the variation in the value of S and give possible reasons.

Annex I – Input File Example

Filename: gg03.jss

```
3 3
0  3  1  2  2  2
0  2  2  1  1  4
1  4  2  3  0  1
```

Filename: ft06.jss

```
6 6
2  1  0  3  1  6  3  7  5  3  4  6
1  8  2  5  4 10  5 10  0 10  3  4
2  5  3  4  5  8  0  9  1  1  4  7
1  5  0  5  2  5  3  3  4  8  5  9
2  9  1  3  4  5  5  4  0  3  3  1
1  3  3  3  5  9  0 10  4  4  2  1
```

Annex II – Output File Example

Filename: gg03.output

```
12
2  8  10
0  2  4
0  4  7
```