Introduction

The purpose of this project is to gain experience in parallel programming using OpenMP and MPI programing models in combination. For this assignment, you are going to write three parallel implementations to solve an optimization problem, assessing the effects of your design decisions on the application performance. In particular, the Speedup and Efficiency metrics will be assessed in relation to the problem size.

Problem Description

Once upon a time, in a distant country, a king in his kingdom possessed a small and valuable collection of trees, which he inherited from the travels of his ancestors to other distant realms. To protect trees from thieves, the king ordered to build a fence around it. This task fell to the wizard of the kingdom, a wise man.

Unfortunately, the magician realized with deep regret that the only available materials to build the fence were wood from the trees themselves. In other words, it was necessary to cut some trees to build the fence around the remaining ones. Logically, in order to avoid the anger of the king and lose his job and his head in the effort, the magician wanted to minimize the value of the trees that should be cut. He retired to his tower and stayed there until he found the best possible solution to the problem. The fence was built and everyone lived happily. In this practice, you must develop a parallel and efficient application to solve the problem of the magician.

Solution Approach

The resolution of this problem can be addressed by following a simple strategy consist on generating and evaluating all combinations of *n* elements taken of *r* in *r* (*r* being the number of trees to be cut, $1 \leq r \leq n-1$). Then, the program must calculate the length of fence required to close the remaining n-r trees and verify if this is a feasible solution. It means that the wood of the *r* cut trees is enough to build the fence. For each feasible solution, it evaluates whether or not it minimizes the value of the cut trees in relation to the best solution found so far. In this last case, it is saved.

The generation of combinations is one of those classic examples solved by a recursive algorithm. To calculate the length of the fence is necessary to compute the convex hull algorithm of a set of points Q (defined as the smallest convex polygon P for which each point in Q is on one side of the polygon P or in its interior). Different algorithms to compute convex hulls are described in [1], [2]. With the materials of this project you will find an implementation of the convex hulls algorithm.

Project tasks

Part I – OpenMP implementation (15th of April)
Taking as a base the serial version provided, write an OpenMP implementation to the *optimal fence problem* using the same input/output definitions than the serial. Name this source code ***optimalfence-omp.c***.

The aim of this part is to be familiar with the different clauses provided by OpenMP to exploit fully the API to get the maximum performance. Be particularly careful with synchronization and load balancing.

Part 2 – MPI Implementation (13th of May)
Taking as a base the serial version provided, write an MPI implementation to the *optimal fence problem* using the same input/output definitions than the serial. Name this source code ***optimalfence-mpi.c***.

The aim of this part is to be familiar with the different clauses provided by MPI to exploit fully the API to get the maximum performance. Be particularly careful with synchronization, communication cost and load balancing.

Part 3 – Hybrid Implementation (30th of May)
Taking as a base your previous implementations and the feedback obtained from classmates and teachers, write a hybrid MPI+OpenMP implementation to the *optimal fence problem* using the same input/output definitions than the serial. Name this source code *optimalfence-hybrid.c*.

The aim of this part is to become aware about the possibilities and limitations of every paradigm and apply them correctly to develop parallel applications that get efficiently the maximum performance. Be particularly careful with synchronization, communication cost and load balancing.

Part 4 – Peer solutions assessment (17th of June)
This part is optional. You can choose to do this part in substitution of the "Writing Work Supercomputers - Cloud Computing". This activity consist in the assessment of the quality of your solutions with respect other solutions made by your classmates. You must identify the best solution in relation to the solutions proposed by your classmates and justify why. Indicate whether is it possible to further improvements and how. The aim of this exercise is to promote critical thinking and help to identify the strengths and weaknesses of your own work.

What to turn in, and when…

Documentation to deliver
Execute your solution to each part using (at least) the facilitated test-bed for different node configurations.

Submit in "Assignments" Section on the Virtual campus a short report (maximum 4 pages) with the following contents:

- URL to the repository (github, Bitbucket, etc) with the source code of your parallel implementation correctly named (please use file names recommendations in the statement).
- Description of the process followed to launch your experiments including makefile, scripts, etc.
- Justify your work/data decomposition decisions.
- Justify your synchronization considerations.
- Show the results and discuss the obtained performance. Are the results what you expected? Explain the experimental study done and represent the results using tables, graphics, etc.
- Final Conclusions

Important Considerations
In the MPI/Hybrid implementations consider the use of collective communication functions. (Bcast, Scatter, Gather, etc.), whenever it is required, but take care about communications efficiency, saturation issues, load balancing, etc. Justify your decisions. Likewise, consider the use of derived types and/or data packing.

It is really important that you justify all your implementation decisions. The only way to show that you understand the problem and acquired the concepts is that your decisions are good justified.

It is really important to describe the experiments, and overall, you correctly interpret and explain the obtained results.

To make the report understandable and easy to assess and review by any reader, you must follow the rules of good practices to prepare a report. The activity should be correctly identified (Title, authors, requirements, etc.). Use editing techniques properly: tables, figures, cross-references, table of contents, etc. You can insert some code pieces for helping to explain your solutions. The figures must be correctly formatted considering aspects such as titles, axis labels, legend, type of lines, symbols, colours, etc. The text in the figures, labels and axis must be legible. The axis must be correctly dimensioned. The type of chart must be selected correctly depending on that you want to explain. Figures must be accompanied by a brief explanation. The report should be complete, justifying the design decisions, making a proper analysis of the performance and the comparison of the effects of different alternatives to reach the more appropriate decision.

It is really important for students in a master degree create good reports and to expose and clearly state your ideas or decisions and this will be taken into account in your evaluation.

Deadlines for the deliveries and Percentage of the final mark:

- Part 1 (OpenMP Program): 15th of April (25%)
- Part 2 (MPI Program): 13th of May (25%)
- Part 3 (Hybrid Program): 30th of May (15%)
- Part 4 (Comparison of the results): 17th of June (15%)

Project materials

In virtual campus section "Resources->Project" you can find a zipped file (hpc-project1718.tgz) with the file materials for this project. Copy this zipped file to your account in the teaching cluster (moore.udl.net) and decompress it.

```
$scp hpc-project1718.tgz <user>@moore.udl.net:/home/<user>
$tar xvfz hpc-project1718.tgz
```

In the 'sourcecode' folder you will find the source code for a serial version of the optimization code.

In the 'testbed' folder you will find the text files with different problem definitions for test-bed purpose. For testing your solutions you can create your own test-bed. However, in order to compare performance and results obtained from the different classroom groups, we provided a minimum test-bed that you have to pass for providing your results, perform the analysis and elaborate the reports. Fell free to increase the test-bed if necessary.

Source code compilation and execution

The source code of a serial version to solve the optimal fence problem is provided. For source code compilation follow the following instructions:

```
$gcc -o optfence_serial optimalfence_serial.c
```

Remember that **is totally forbidden to execute the code in the front-end** of the cluster. Doing this you can collapse the server or produce a server downfall. By this, it is important to use the front-end queuing system even for serial programs.

For queuing and execute serial processes follow the next instructions:

```
$qlogin -pe smp 4
```

```
user@compute-0-X$
```
(*Notice that your prompt has changed to the assigned machine to* ***compute-0-X.***)

```
user@compute-0-X$ cd hpc-project1718
user@compute-0-X hpc]$ ./optfence_serial <test_file>
```

Problem configuration (input file)

The problem description is described in a input text file composed by the following information:

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

The first line contains an integer number, which will be denoted as ***n***, $2 \leq n \leq 30$, that correspond to the total number of trees in the forest. The following *n* lines contains 4 integer values, x_i, y_i, v_i, l_i, describing a particular tree (every tree should be identified with a correlative number from 1 to n). The first two values define the coordinates of the tree *i-th* (x_i, y_i), next is the tree value v_i, and finally the length of the fence that you can built with such a tree l_i. The values v_i and l_i are in the range [0,10000].

Output

The program should calculate the subset of trees that allow making a single fence to protect the non-felled trees. The goal is to find the subset with a lower cost. If there is more than one subset with the minimum cost, we must choose the one with the fewest trees. For simplicity, we consider that the trees have diameter zero.

The output of the program must identify the trees to cut, the missing Wood from the cut trees, the value of the cut trees and the value of the remaining trees.

```
Cutting   3 trees            :   2   5   4
Fence long./missing Wood     :       7.84 (  3.16)
Value of the cutted Trees    :       9.00
Value of the Remaining Trees:      24.00
```

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. **"**Introduction to Algorithms**"**, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0262032937.

[2] Convex Hull. Dan Sunday. 2012 http://softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm. March 2018.