Parallel and distributed systems: paradigms and models

Academic year 2020-2021

Final project (Version 1.0)

The project consists of picking up one (only one) of the subjects listed below and designing a parallel application solving the proposed problem. There are two kinds of projects: application projects and framework projects. For the application projects, we would like to have two applications, one using only the C++ threads (stdlib ones) and one using FastFlow. Not necessarily the two applications would completely share the same design. The applications should be eventually used to measure performance achieved on the machines whose access has been provided to the students. For the framework projects, we would like to have two implementations of the framework, providing a similar API, one implemented using only the C++ threads and one implemented on top of FastFlow.

Project is an individual assignment. We will not accept "group" projects. This project assignment is valid for all the exam terms of the Academic year 2020—2021. Students must agree with the professor about the choice of the subject. The student should write a message to the professor (Subject: *SPM project choice*) specifying in the message text the project chosen. Before starting to work on the project, the student must receive explicit approval by email. Professor may ask the student to change his/her choice in case the choice is overbooked. A list of the project subjects will be maintained in a classroom accessible file.

Application projects

K-Nearest Neighbors (KNN)

Given a set of points in a 2D space, we require to compute in parallel for each one of the points in the set of points the set of *k closest* points. Point *i* is the point whose coordinates are listed in line *i* in the file. The input of the program is a set of floating-point coordinates (one per line, comma separated) and the output is a set of lines each hosting a point id and a list of point ids representing its KNN set ordered with respect to distance.

Graph Search

A graph is described by a set of nodes N (with an associated value) and a set of arcs (oriented pairs of nodes). The application should take a node value X, a starting node S and must return the number of occurrences in the graph of the input node X found in the graph during a bread first parallel search starting from the node S. The graph to be searched is assumed to be acyclic.

Free choice application

You can choose a different application to parallelize. In this case you need to agree the application project with the professor. Send the professor a message with the subject "SPM project choice" and describe (in a synthetic way) the application and the possibilities of parallelization in the message body. After the professor agrees on the subject, you may start preparing the project with the agreed "free" application.

Framework projects

Cellular automata

A cellular automaton consists of a regular grid of *cells*, where each cell may be in one of a finite number of states. The grid to be considered is toroidal 2D mesh, where the last row (column) must be considered adjacent to the first row (column). An initial state is determined by an assignment of a particular state for each cell. A new *generation* is created according to some fixed set of *rules* that determine the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. The rule for updating the status of cells is the same for each cell and does not change over time and is applied to the entire grid simultaneously. The computation of a cellular automaton consists of the computation of the changes in grid cell status for a given number of iterations.

A programming API must be provided that supports the definition of the grid dimensions, the rules used to compute a cell as a function of its current status and its 8-neighbor cell status, the initial state of the grid and the number of iterations to be computed and the execution of the defined simulation.

Autonomic pipeline

A framework supporting the definition of a *k* stage (stateless stages) pipeline and its run time, autonomic adaptation to the varying computations times required for the different tasks appearing onto the input stream. The framework should be able to:

a) merge adjacent states if their combined service time results smaller that the service time of the slower pipeline stages
b) split the execution of previously merged states in case their combined service time results larger that the maximum of the service times of the other stages

To test the pipeline implementation, a stream of tasks made of k+1 integer tuples <x, t_1, … t_k> can be used. The computation of a generic task onto stage *j* produces a new task whose first item is *x+1*, the rest of the items are the same as the one received in input and this computation requires t_j units of time.

Free framework choice

You can propose a different framework. The framework must provide a kind of “parallel design pattern”, that is a reusable, parametric parallelism exploitation pattern. You should send an email to professor, with subject “SPM project choice” detailing which pattern you want to implement (synthetic description). After professor’s approval, you may start implementing the “free” framework project.

Project delivery

Once completed, the project must be delivered to the professor by email, by one of the exam terms. The email must be made such that the subject is “*SPM Project submission*” and must have, as attachments, the code developed (all the files needed to recompile and rerun the project) and a report (PDF) of max 10 pages describing:

a) the main design choices
b) the expected performances
c) the actual implementation and results achieved
d) a comparison among the implementation and performances of the C++ and the FastFlow solutions
e) an analysis of the reasons of differences observed among expected and measured performance

f) the commands needed to rebuild and run the project
g) the API to be used to run a new application (only for frameworks)