SPM (recovery course) 2017-18: Final project

Marco Danelutto

version 1.0

1 Introduction

This is the final project for the recovery course of SPM A.Y. 2017-18. The project is assigned to individual students. The project has to be prepared and sent to the teacher by one of the official dates published on the `esami.unipi.it` web site. Students may pick up any of the projects listed below (either Kernel pr Skeleton) but the choice should be agreed with the professor **before** starting to work on the project, either via email or in person, during question time.
When completed, the project sent to the professor via email must consist in:

1. a message with subject SPM 1718 project submission (please observe we need exactly this subject. The messages will be filtered automatically and if you don't use the correct subject I'll not receive your submission;
2. a PDF document, in attachment, with the project report, of max 10 pages (strict);
3. a tar.gz (or .zip) document, in attachment, with the project code, the examples, the makefiles, and all whats necessary to recompile and run the project.

Each one of the two attachments is described in detail later on in this document. After receiving all the projects relative to the exam session, the teachers will take about one week to mark them (a little bit more in case of a huge number of projects submitted) and then they will publish a calendar of oral exams for the students that submitted a project eventually ranked sufficient or higher. The oral exam is made of two parts:

1. a short demo of the project run by the student using one or more text terminals connected via SSH to the parallel machines where the project has been developed. During the demo the student will be asked to answer questions relative to the project structure, code, behaviour.

2. two/three questions relative to the topics presented and discussed in the course and covered by the teaching material (project course notes + book chapters covering the last part of the course arguments).

At the end of the oral exam, the student will get the final exam mark registered. In case, the student may submit the project at exam session i and have the exam at session $i + k$ provided it is in the same period (e.g. submit the project in June and have the oral exam in July, but not in September).

2 Kernels

2.1 Monte Carlo

The application computes the integral of a function using a Monte Carlo method: given a function $f(x)$ to be integrated in the interval $[a, b]$ we choose a number N of random points $h_{x1}, \dots, x_N\ i : xi \in [a, b]$ and we compute the integral as

$$\frac{1}{N}\sum_{i=1}^{N}(f(x_i)(b-a))$$

Given a function f the integral is computed over a stream of intervals

$$\langle\langle a_1, b_1\rangle, \dots, \langle a_m, b_m\rangle\rangle$$

2.2 Function approximation (genetic algorithm)

A random population of functions $F = \{f_0, \dots, f_f\}$ is subject to evolution to look for functions that approximate an unknown function f' implicitly defined by a set of points with values

$$P = \{\langle x_1, y_1\rangle, \dots, \langle x_m, y_m\rangle\} \quad \forall i \in [1, m] y_i = f'(x_i)$$

The functions in F are defined according to the following grammar:

```
<const> ::= 0 | 1 | 2 | ...
<var> ::= x | y | ...
<leaf> ::= <const> | <var>
<binop> ::= - | + | * | / | pow
<unop> ::= exp | sin | cos | log
<node> ::= <leaf> | <monop> <node> | <node> <binop> <node>
```

The application evaluates the approximation of function f' by minimizing the fitness function ($f \in F$):

$$E(f, P) = \sqrt{\sum_{i=1}^{m} (y_i - f(x_i))^2}$$

The functions having the better fitness value are selected for being modified by the crossover or the mutation operator. The crossover randomly selects two subtrees belonging to two different trees respectively and exchanges them. The mutation substitutes a randomly selected subtree with a new one of the same depth. In both cases, the new generated trees substitute the modified ones in the population. Thus, generation after generation, the functions exhibiting the better fitness.

2.3 Histogram Thresholding

This module performs histogram thresholding on an image. Given an integer image I and a target percentage p, it constructs a binary image B such that $B_{i,j}$ is set if no more than p percent of the pixels in I are brighter than $I_{i,j}$. The input data are therefore the matrix I and the percentage p. The thresholding must be applied to a stream of input images.

2.4 Free choice kernel/application

Students are encourage to propose and discuss with the professor different alternatives, possibly individuated because of personal interests, subjects already tackled in other courses etc.

In order to be considered valid alternatives:

- either the alternative consist in parallelizing and existing and well known application/kernel/algorithm by means of one of the structured parallel programming frameworks discussed in the course

- or the alternative consist in implementing a new (alternative) pattern using either the low level mechanisms discussed in the initial part of the course or integrating the new pattern in one

of the existing frameworks.

3 Patterns

3.1 Pipeline with load balancing

The skeleton implements a pipeline, whose stages are sequential. It manages to monitor the latencies experienced in pipeline stages and to merge S_i with S_{i+1} in a single sequential stage in case $L_i + L_{i+1} \leq \max\{L_k\}$.

3.2 Micro-mdf (MDF)

Micro-mdf is a minimalist parallel programming framework supporting execution of macro data flow graphs. In particular, it must support the following features:

- data flow instructions with instruction and graph id, pointer to the function to be computed, input token and output destination placeholders and a missing input token counter
- a graph repository, suitable to host several graphs of data flow instructions
- a streamer, generating a type stream of tokens directed to the first instruction of some data flow graph, and instantiating in the graph repository a new copy (with a new graph id) of a data flow graph with the token properly assigned to the graph first instruction
- a drainer, processing each final token, i.e. each token produced while executing the graph in the graph repository with special destination output
- an interpreter looking for the fireable instruction in the graph repository and processing them up to the point no other fireable instructions are present and the streamer already terminated its operation.

4 Submit HOWTO

These are mandatory recommendations for the project submission:

- the project report should include only major design and implementation choices, performance plots, a synthetic usage manual, etc. It should not include parts of these document nor description of the problem/skeleton parallelised/implemented
- all graphs in the report should be mandatory black and white
- the project code archive should host anything needed to run the code, including makefiles (if any), input data sets, etc.
- submission has to be performed via email at the official professor email address within the official exam dates. If at the official deadline you still have to fix something (report or code), send a message as if it where the submission of what you did so far and we'll agree some days of extension.
- Application/kernel projects should be implemented using FastFlow and C++ threads and the two implementations should be compared in terms of the achieved performances
- Skeleton projects should be implemented either using C++ with threads or with the "core" FastFlow skeletons
- all communications or material supporting the project implementation will be published on the didawiki web page