SPM 2011–2012: Final project

Version 0.9

May 3, 2012

The project is assigned to individual students or to groups of max 2 students. The project has to be prepared and sent to the teacher by one of the deadlines (exam dates) published on the course web site[1] (and on secretary web site). One deadline per exam session will be given. At the moment being, the first three deadlines (relative to the Summer session exams) are fixed on June 4th and 15th and on July 16th, respectively. The last deadline of the term (last deadline in June-July, September deadline and the last deadline in January-February) may be extended (one week) provided the student sends an email to the professor with the work made so far by the official exam deadline. The project sent to the professor via email *must* consist in:

a message with subject "SPM project submission"
a PDF document, in attachment, with the project report, of max 10 pages
a `tar.gz` document, in attachment, with the project code, the examples, the makefiles, and all what's necessary to recompile and run it

Each one of the two attachments is described in detail later on in this document (see Sec. 3). After receiving all the projects relative to the exam session, the teacher will take about one week to mark them (a little bit more in case of a huge number of projects submitted) and then he 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:

- a short demo of the project run by the student using one or more text (only) 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 and execution behaviour. Possibly, the student(s) may be asked to implement small changes in the project code[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).

1 Project subjects

The student can choose one of the two projects listed below. Both projects are somehow "underspecified": part of the project has to be defined by the student. As an example, the "skeleton" projects do not specify any use case to be used to test the implementation. It is up to the student to figure out proper tests/simple applications, in this case. The complete definition of the project out of the project schema presented here is part of the project itself and it will be evaluated during the exam.

1.1 "Skeleton" projects

The final goal is the implementation of a simple run time/library for one of the structured parallel programming patterns discussed in the course. In other words, the student must provide some code that can be used to implement a generic application exploiting the parallel pattern subject of his/her implementation. The implementation may be written using C, C++ or Java and must

[1] `http://didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spm/start`

[2] usual Linux text only editors will be available: vi, emacs, pico, etc.

run on POSIX (Linux) workstations. Depending on the skeleton, COW/NOW, multi cores or even network of multi core architectures should be targeted. The reference target architectures, that is the ones where the project will be run during the final demo, are

1. the Linux PC in Aula H (or aula M or aula I)
2. a symmetric multicore [3] which will be made available by the teacher to the students targeting multicore only architectures.

This year we will consider the following skeletons (that is the project consists in implementing one of the following skeletons):

Divide&Conquer skeleton. The student must implement a divide and conquer skeleton.The skeleton takes four "code" parameters:

- a `divide` function, splitting input data into a collection of data of the same type
- a `ifbasecase` function, returning true if the result relative to the input data may be directly computed
- a `basecase` function, computing the result out of the input data when the input data represents a base case
- a `conquer` function, computing a result out of a collection of partial results.

The divide and conquer computation may be defined as follows:

```
d&c(div,ifbc,bcs,conq,input) {
  if(ifbc(input))
  then return bcs(input);
  else {
    parts = div(input);
    ress  = map(d&c(div,ifbc,bcs,conq),
                parts);
    return conq(ress);
  }
}
```

It is worth taking into account that not necessarily *ifbasecase* should return true for the "algorithmic" base case. It may be assumed that under a given "threshold" divide and conquer should apply a different–sequential–algorithm rather than going up splitting input data up to the base case. As an example, consider the quicksort algorithm. Rather than splitting up to the base case (lists of no more than 2 items), we can decide that when lists are shorter than k elements we may sequentially apply a `bubblesort`. In this case `ifbasecase` will simply return true when `input.lenght()<k` and `basecase` will simply call `bubblesort`.

[3]most likely `ottavinareale.di.unipi.it` or other multicores possibly available

MapReduce skeleton. The student must implement a map-reduce skeleton. Two versions of the skeleton are required. The *first one* is the classical map reduce. In this case the skeleton takes two code parameters:

- a `map` function f to be applied on the items of the input collection, and
- a commutative and associative `reduce` function $\oplus$ to be used to "sum up" all the items in the input collection after the application of the map function.

Therefore, given an input collection of data

$$\langle x_1, x_2, \ldots, x_m \rangle$$

the final result should be

$$f(x_1) \oplus f(x_2) \oplus \ldots \oplus f(x_m)$$

The *second version* is the "Google" variant of the map reduce. In this case, the skeleton takes the same code parameters as before. However, the function f is such that $f(x_i)$ returns a key–value pair $\langle k_i, v_i \rangle$, and the reduce function should be computed on all the pairs relative to the same key. That is, the result should be a collection of values $\langle y_1, \ldots, y_k \rangle$ where each y_j is the result of "sum" (through the $\oplus$ operator) of the v_i relative to the pairs with key equal to k_j.

```
mapreduce(f,g,input) {
  foreach x in input
    <ki, vi> = f(x);
  keys = set of distinct ki;
  foreach k in keys {
    values = set of values with key k
    res = reduce(g, values)
    add res to result
  }
  return result
```

Parameter sweeping skeleton. The student must implement a parameter sweeping skeleton. The parameter sweeping skeleton takes as parameters a function f, a comparison function `comp` (a binary function returning a boolean, comp(a,b) = true iff a is "better" than b) and a set of input data sets $s_1, \ldots, s_k$, and returns the input data set s_i such that $\forall j \neq i$ comp(s_i, s_j) = true. The student must consider the possibility f is given as an application (executable reading input parameters from standard input and writing results to the standard output). In this case we may assume the "result" of the application is an integer, and the comp simply evaluates $a > b$.

```
PS(applname, inputs) {
  foreach input in inputs {
    res = exec(applname(input));
    add <res, input> to results;
  }
  res = null;
  foreach r in results
    if comp(r,res)=true then res = r
  return res;
}
```

Domain specific skeleton . The student may propose a new/different skeleton, that is a new/different parallelism exploitation pattern which is efficient, resusable and parametric, as other skeletons are. The new skeleton has to be discussed with (and approved by) the teacher *before* actually starting the project.

In all cases, the project report *must include*:

a description of the concurrent activity graph and the implementation graph
the proper performance models related to the skeleton (abstract model) and to the implementation (concrete model)
the code implementing the run time support

and the skeleton should be implemented

- targeting either a single multi core or a COW/NOW, in case the implementation/project is developed by a single student, or

- targeting a network of multi core workstations, in case the implementation/project is developed by a group of two students.

A comparison of the expected performance vs. the achieved performance figures must be included in the project report.

1.2 "Application" projects

The final goal is the implementation of an application using a skeleton based structured programming environment. The candidate programming environments are those introduced and discussed during the course, that include[4]:

Env.	Host lang.	Target hw
Muesli	C++	multicore COW/NOW
FastFlow	C++	multicore
Skandium	Java	multicore
SkeTo	C++	COW/NOW (data parallel only)

Depending on the framework chosen, three kind of architectures may be targeted: multi cores, COW/NOW or network of multi core workstations. The reference architectures will be `ottavinareale` in the first case and the machines in Aula H, M and I, in the second and third case.

This year we consider the following applications (that is the project consist in implementing one of these applications, using a structured parallel programming framework):

MonteCarlo 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 $\langle x_1, \ldots, x_N \rangle : x_i \in [a, b]$ 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 $\langle\langle a_1, b_1\rangle, \ldots, \langle a_m, b_m\rangle\rangle$.

Bucket sort Given a stream of lenght m of floating point vectors of lenght n, produce the stream of sorted vectors using a bucket sort. Bucket sort divides the input vector into k buckets. Each bucket B_i hosts vector items that are smaller than the items in buckets $B_j : \forall j > i$ and larger that the items in buckets $B_j : \forall j < i$. Each bucket is sorted sequentially and the sorted vector is built out of the sorted buckets.

Game of life Given a blackboard of $N \times N$ positions, M iterations of the "Game of life" have

[4]but are not limited to. Students may consider other structured parallel programming environments (to be approved by the teacher).

to be computed. The initial blackboard is randomly filled (each position is either "alive" (full) or "dead" (empty)). At each iteration, position i, j evolves according to the following rules:

Current state	Alive neighbours	Next state
Alive	≤ 1	Dead
Alive	≥ 4	Dead
Alive	3	Alive
Dead	3	Alive

The blackboard is to be considered a mesh. Position $i, M-1$ is adjacent to position $i, 0$ and position $M-1, j$ is adjacent to position $0, j$.

Free application The student can pick up an application in his/her favorite domain and implement the application using a skeleton framework. The application has to be discussed with (and approved by) the teacher *before* actually starting the project.

In all cases, the project submission should eventually include:

The design of the implementation with the available skeletons, including an estimate of the performances
The application implementation
A comparison of the performances achieved with the ones expected

2 Project assignment

When a student (group of two students) decides to start working on the project he/she should first "negotiate" the project with the professor. He/she communicates to the professor the project chosen sending an email (subject "SPM project choice"). The email text should give an idea of i) which project subject has been chosen and ii) of the unspecified parameters of the project. In case of "free application", as an example, the application chosen should be introduced; in case of any application, the framework chosen for the implementation is to be specified; etc. The professor, in a short amount of time, returns an acknowledge message possibly including more detailed requirements and/or modifications of the choices made by the student/s. In particular cases, the professor may ask the student(s) to discuss the project assignment during question time. After the acknowledge, the assignment is registered on the project web page[5] and the student(s) may start working. The acknowledgment may be implicitly substituted by the pubblication of the assignment on the project web page.

Help Desk

Any question relative to the project (design, coding, debugging) can be posed during the lesson breaks, during the question time (question time will be active even in periods without lessons) or by email (simple questions only). Questions relative to the programming environments may also be sent to P. Dazzi (via email).

3 Submission details

3.1 Project report attachment

The project report is a short report (no more than 10 pages, excluding code). It must include:

- the specification of the project chosen, as agreed with the professor
- the general design of the work done
- the peculiarities tackled when implementing the project
- the comparison among performance predicted with the performance models and the one observed during the experiments
- a one page "user manual" explaining how the code may be compiled and run

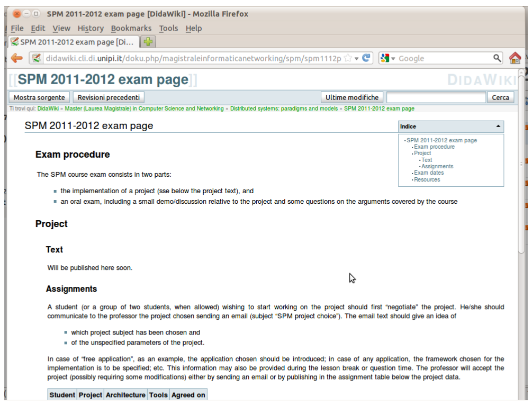
Please do not copy&paste parts of the project text, of the SPM notes book, etc. Conciseness of the project report is appreciated. The ability to report only important facts is also evaluated.

3.2 Project sources

The whole sources of the project, including

(full) code
sample code/data used to exercise the run time support or to run the application
makefiles or ant files used to compile the project
scripts used to run the project (if any)

[5] `http://didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spm/spm1112proj`


SPM 2011-2012 exam page [DidaWiki] - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SPM 2011-2012 exam page [Di...
didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spm/spm1112p
Google
[[SPM 2011-2012 exam page]]
DidaWiki
Mostra sorgente
Revisioni precedenti
Ultime modifiche
Cerca

Ti trovi qui: DidaWiki » Master (Laurea Magistrale) in Computer Science and Networking » Distributed systems: paradigms and models » SPM 2011-2012 exam page

SPM 2011-2012 exam page

Indice
- SPM 2011-2012 exam page
  - Exam procedure
  - Project
    - Text
    - Assignments
  - Exam dates
  - Resources

Exam procedure

The SPM course exam consists in two parts:

- the implementation of a project (sse below the project text), and
- an oral exam, including a small demo/discussion relative to the project and some questions on the arguments covered by the course

Project

Text

Will be published here soon.

Assignments

A student (or a group of two students, when allowed) wishing to start working on the project should first "negotiate" the project. He/she should communicate to the professor the project chosen sending an email (subject "SPM project choice"). The email text should give an idea of

- which project subject has been chosen and
- of the unspecified parameters of the project.

In case of "free application", as an example, the application chosen should be introduced; in case of any application, the framework chosen for the implementation is to be specified; etc. This information may also be provided during the lesson break or question time. The professor will accept the project (possibly requiring some modifications) either by sending an email or by publishing in the assignment table below the project data.

Student	Project	Architecture	Tools	Agreed on

Figure 1: Didawiki page relative to the project available at http://didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spm/spm1112proj

must be sent as a tar, gzipped file. Following the instruction in the project report the teacher should be able to run the code with proper inputs and to observe the results described in the project report.

The source code file should be named as follows: `NameFamilyname.enrollmentNumber.tar.gz` As an example, I would submit a file with name `MarcoDanelutto12345.tar.gz` In case the project is prepared by two students, the project file should be named using both names and enrollment numbers.

The code should be decently commented. In case of usage of tools such as `javadoc` or `doxygen`, the commands necessary to generate the documentation should be included in the proper `makefile` or `ant` files.

The code may be developed on any machine, including student notebooks or other machines they have access to, but as far as the evaluation process is concerned, the code must run either on the Aula H/I/M machines or on `ottavinareale.di.unipi.it`.

Project validity

The project described in this document is valid for all the exam terms in Academic Year 2011-2012, that is from June 2012 to February 2013. The assignment may be required at any time since project publication on. The assignment is valid up to moment a new, different assignment is required by the student(s) or up to start of the 2012-2013 course. The start of next year course "resets" any pending project work.