# SPM 2014–2015: Final project

Version 0.1

December 9, 2014

The project is assigned to individual students or, exceptionally, 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. 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.

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

[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.

implementation. The implementation may be written using C, C++ or Java and must run on POSIX (Linux) workstations. Students may consider to use the "core" FastFlow components to implement the skeletons in addition to the POSIX/C++ framework. Multi core architectures should be targeted while implementing the project. The reference target architectures, that is the ones where the project will be run during the final demo, are

1. a symmetric multicore for the projects preparing the project using Java/POSIX (`andromeda.di.unipi.it`, an 8 core Nehalem machine) and
2. a machine equipped with a Xeon PHI board, for the students developing the project with C/C++/POSIX (frameworks) (`r720-phi.itc.unipi.it`)

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

---

**MapReduce** skeleton. 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.

The mapreduce skeleton to be implemented is the "Google" version of the mapreduce. 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 $k_j$.

Therefore, given an input collection of data $X = \langle x_1, \ldots, x_m \rangle$ if the computation of map$(f)(X)$ gives a set of $\langle k_j, v_i \rangle$ pairs $FX$ with keys in $K = \{k_1, \ldots, k_k\}$ the final result should be

$$\text{mapreduce}(f, \oplus)(X) = \langle \bigoplus_{\langle k_1, v_j \rangle \in FX} v_j, \ldots, \bigoplus_{\langle k_k, v_j \rangle \in FX} v_j \rangle$$

**Multipool** skeleton. The skeleton models "multi population" genetic algorithms computing the evolution of a pool of individuals, subject to "mutations". The multipool skeleton must implement, in parallel, the following iterative algorithm:

```
MultiPool(termination, split, merge, select, evolve, filter, Pop)
begin
    {P_0, ..., P_n} = split(Pop)
    while(not(termination(Pop))
        foreach P_i in Pops
            repeat $k$ times
                N = evolve(select(P_i))
                P_i = P_i ∪ filter(N, P_i)
            end repeat
        {P_0, ..., P_n} = merge(P_0, ..., P_n)
        end foreach
    end while
end
```

where *termination* is a boolean function checking if the current population is the "good" (final) one, *split* splits the current population into subpopulations, *merge* redistribute individuals from subpopulations to subpopulations according to a given policy, *select* selects (part of the) individuals to be mutated from the current population, *evolve* mutates and individual (or a pair of individuals, in case of crossover) and $filter$ selects which one of the mutated individuals must be inserted in the current population. At least the $evolve$ function must be implemented in parallel. Students must consider the possibility to evaluate in parallel also the other functions parameter of the pool skeleton.

**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 |
| the comparison among the predicted and measures performances (completion time and/or speedup/scalability) |

## 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[3]: FastFlow, Skandium and SkePU (data parallel projects only). The reference architectures for the application project will be the same ones listed in the skeleton project sections: the 8 core (16 contexts) Nehalem machine `andromeda` and the Xeon PHI machine `r720-phi`.

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

---

**Parallel Prefix** Given a vector of floats $x_1, \ldots, x_n$ and an associative function $\oplus$ compute in parallel the vector whose $i-th$ element $y_i$ is the "sum" (with $\oplus$) of all the elements from $x_1$ up to $x_i$, that is:

$$y_i = x_1 \oplus x_2 \oplus \ldots \oplus x_i$$

Students should consider different functions as $\oplus$, ranging from simple arithmetic operators (e.g. $+$) up to longer operations (e.g. $+$ computed with an (active) delay of $O(10)$ to $O(100)$ $\mu$secs.

**Fish and sharks** An initial population of fishes and sharks evolves according to the following set of rules:

1. Fish breeding age starts at 2, shark breeding age starts at 3
2. Fish live up to 10, then they die, sharks live up to 20, then they die
3. An empty cell with $>= 4$ fish (shark) neighbours and $>= 3$ of them in breeding age and $< 4$ shark (fish) neighbours is filled by a new fish (shark) individual with age 1
4. A fish in a cell
   - dies if $>= 5$ neighbours are sharks (Shark food)
   - dies if $>= 8$ neighbours are fishes (overpopulation)
   - otherwise its age increases
5. A shark in a cell
   - dies if $>= 6$ neighbours are sharks and =0 neighbours are fishes (starvation)

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

**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, excluding Christmas holidays) or by email (simple questions only) (via email) to both course professors (Danelutto and Torquati).

- dies with $\frac{1}{32}$ probability by random causes
- otherwise its age increases

The application computes the evolution of an initial population is a mesh of $N \times N$ cells (25% sharks, 50% fishes and 25% empty cells) for a number $M$ of iterations. A library call will be provided to display a matrix of short integer values on the screen, that will run on the host processor where the Xeon PHI co-processor is attached.

**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 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[4] and the student(s) may start working. The acknowledgment may be implicitly substituted by the pubblication of the assignment on the project web page.

## 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

[4]http://didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spm/spm1314pro

- 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) |

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 `andromeda.di.unipi.it` or on `r720-phi.itc.unipi.it`.

**Project validity**

The project described in this document is valid for all the exam terms in Academic Year 2014-2015, that is from January 2015 to September 2015 exam sessions. 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 2015-2016 course. The start of next year course "resets" any pending project work.