Parallel and distributed systems: paradigms and models
Academic Year 2018-2019 – Final project – Rev 1.1

Each student must implement only one project, chosen among the following 6 projects (4 assigned, 2 free choices). The choice must be communicated by email to the professor (mandatory ***Subject**: SPM19 project choice*) and after the project approval (by email) the student may start work. In a few cases, the professor can ask to change the choice, for different reasons. Projects are divided into two classes: applications projects and framework projects. The application projects aim at parallelizing a given application. The framework projects aim at providing a parallel implementation of a pattern that will be tested using a simple, possibly synthetic, application.

Application projects

The application projects require:

1. to properly design and refine the parallel structure of the application, using the methodology discussed in the SPM course
2. to provide a reference sequential version, to be used to compute T_{seq}
3. to provide a performance model of the (expected) parallel performances
4. to provide two parallel implementations, one using C++ only (threads and relative mechanisms) and one using FastFlow
5. to perform (and report) experiments showing actual performances achieved on the Xeon PHI KNL compared with the expected values from the model.

Swarm particle optimization

The target is to develop an application finding the *minimum of a function in a given interval* using swarm particle optimization. The function is given as a parameter in the code, and should have type *std::function<float(float, float)>*. The Swarm Particle Optimization (SPO) uses particles to explore the state of solutions. It is an iterative process where a set of particles, initially spread randomly across the solution space, autonomously travel the solution space driven by local and global knowledge relative to the solution explored so far.

The SPO pseudocode can be summarized as follows:

INITIALIZATION:

a. *distribute n particles across the search space (uniform distribution)*
b. *evaluate the objective function f for each particle position and assign the computed value as local optimum*
c. *assign to the global optimum the "best" computed local optimum*
d. *randomly initialize particle velocities*

ITERATION:

a. *for each particle, update position as a function of current velocity, distance from local optimum and distance from global optimum.*
b. *re-evaluate local and global optima*

The key point in the algorithm is the iteration step. Usually:

a. Velocity is updated as a linear combination of current velocity, distance from local optimum and distance from global optimum. Multiplicative coefficients in the linear combination are execution parameters (same for all particles). Distance from local/global optima are weighted with a random number R_i in [0,1]
$V(t+1) = a\ V(t) + b\ R_1\ (Pos(t) - Pos(localOpt)) + c\ R_2\ (Pos(t) – Pos(globalOpt))$

b. Position is re-computed as the current position plus current velocity (i.e. assuming a single time unit across iterations)
$Pos(t+1) = Pos(t) + V(t+1)$

The application computing the minimum of a function though SWO must provide as a result the global minimum computed after *niter* iterations.

Skyline stream application

The target is to develop an application computing the *skyline* of a stream of tuples. We consider tuples made of *k* items. Items at position *k* have type $type_k$. For each $type_k$, an order is defined by a $>_k$ relation. A tuple $t_1 = <x_1, \dots, x_k>$ dominates a a tuple $t_2 = <y1, \dots, y_k>$ if and only if at least one component of t_1 (x_j) it is better than the corresponding component of t_2 ($x_j >_k y_j$) and all the other components of t_1 (the x_j such that *i!=j*) are better or equal. (i.e. either $x_i >_k y_i$ or $x_i =_k y_i$). The skyline of a set of tuples is the set of tuples that are not dominated by any other tuple.

The skyline stream application considers a (possibly infinite) stream of tuples. For each window of size n (n consecutive tuples), the application computes the skyline and outputs the skyline onto the output stream. Consecutive windows differ by exactly k item (sliding factor). As an example, the stream:

a, b, c, d, e, f, g, h, I, j , k, l, m, n, o

with n = 4 and k =2 results in the following windows:

a,	b,	c,	d
c,	d,	e,	f
e,	f,	g,	h
g,	h,	l,	j
…			

In order to execute the tests, a random stream of integer tuples of k items each must be used, with the classical integer comparison arithmetic.

Free choice application

Students may propose any application of interest to be parallelized, but the application must be agreed with the professor. After agreeing the application subject, the work to be performed is the one listed for the two assigned applications.

Framework projects

These projects require to implement a parallel pattern, through the following steps:

1. properly design and refine the parallel structure of the pattern implementation
2. provide a performance model of the (expected) parallel performances
3. provide a parallel implementation using C++ only (threads and the relative mechanisms)
4. optionally, provide an implementation using low-level FastFlow building blocks
5. perform (and report) experiments showing actual performances achieved on the Xeon PHI KNC compared with the expected values from the model, using synthetic benchmark applications or the applications suggested in the text.

BSP pattern

The Bulk Synchronous Parallel (BSP) parallel programming can be described as follows:

- Parallel computations are performed by a sequence of super-steps
- Within each super-step, a set of independent concurrent activities is computed. Each independent activity schedules communications to other concurrent activities, which will be performed at the end of the super-step
- When all the super-step concurrent activities are terminated, all the communications are performed and finally the next super-step is started, with each concurrent activity being able to consume the data carried by the communications directed to that concurrent activity.

The goal in this case is to provide a pattern implementing a BSP computation such that the user may provide as parameters, for each one of the super-steps: 1) the code of the independent concurrent activities in each one of the super-steps, and 2) the items to be communicated during the communication phase at the end of each super-step (data to be sent, destination (index of the destination) of the receiving concurrent activity). In case of iterative algorithms, 3) a condition on the global state returning true in case termination has been reached is also to be provided. In case the condition does not hold true, the number of the super-step to be executed again must be provided.

For the sake of simplicity, we assume all items exchanged in communications must have the same type T.

The pattern must be tested executing the Tiskin BSP sorting algorithm described as follows:

- The *n* input items to be sorted are distributed among *p* concurrent activities (n/p each, n must be multiple of p)
- First super-step: each concurrent activity sorts its data portion and then selects *p+1* samples uniformly distributed in the ordered sequence and including the first and the last item in the ordered sequence. At the end of the super-step, each concurrent activity sends to the all the other concurrent activities the *p+1* samples.
- Second super-step: each concurrent activity sorts the list of all samples received, including the ones computed on its own. Then it picks up *p+1* separator items equally distributed in the sequence. Subsequently sends all the items in between separator *p* and *p+1* to concurrent activity *p*.
- Third super-step: each concurrent activity sorts the received items.

Autonomic farm pattern

The goal is to provide a farm pattern ensuring (best effort) a given service time leveraging on dynamic variation of the parallelism degree. The farm is instantiated *and* run by providing:

i. A collection of input tasks to be computed (of type *Tin*)
ii. A *function<Tout(Tin)>* computing the single task
iii. An expected service time TS_{goal}
iv. An initial parallelism degree n_w

During farm execution, autonomic farm management should increase or decrease the parallelism degree in such a way its service time is as close as possible to the expected service time TS_{goal}.

The pattern should be tested providing a collection of tasks such that the tasks in the initial, central and final part all require a different average time to be computed (e.g. 4L in the first part, L in the second part and 8L in the third part) and the task collection execution time is considerably longer than the time needed to reconfigure the farm.

Free choice pattern

Students may propose any pattern of interest to be implemented, but the pattern must be agreed with the professor. After agreeing the pattern subject the work to be performed is the one listed for the two assigned patterns.

Report

A report of maximum 10 pages must be prepared described all the points (general idea and notable implementation details) of the points listed at the beginning of Sec. “Application projects” and “Framework projects”, that is including (at least) these sections:

a. parallel architecture design
b. performance modeling
c. implementation structure and (notable) implementation details
d. experimental validation (with plots)

Any graphs included must be B&W (no colors, use different types of lines (full, dashed, etc.) in the case of multiple plots).

Exam procedure

For the term whose exam day is reported as “Day X” on the esami.unipi.it portal:

a. by “day X” send an email message (mandatory Subject: SPM19 project submission) with your name, enrollment number in the body, and with the PDF report and archive of the source files (zip or tar/tgz only) needed to recompile and run the project in attachment. Professor will read messages the subsequent morning, therefore you can send messages any time before 8am of the Day X+1, indeed.
b. professor will take some days to “mark” the projects (depends on how much are submitted to the term and on the other exams in the period, but usually the time will be a few days up to one week)

then he will send you an email (to the address you used for the submission) with the time of the oral exam

c. the oral exam will consist in
    a. discussion of the project, with demo through a text only terminal provided by the professor connected to the Xeon PHI machine, and
    b. some questions relative to the material discussed during the course.
d. The project discussion may require running the tests, to make small modification to the code or to show in the code where and how certain features are implemented. You may be asked to come to the oral exam also in case the project submission has problems (that is, it is not enough to pass the exams). In this case during the oral the professor will only ask you questions about project implementation and tell you what you should do to improve the project.
e. if by the project submission deadline you are still missing something that can be fixed in some more days (a few, no more than a week), you may send the project “as is” by the deadline and ask an extension in the message text, that will be automatically given, unless particular schedules of the professor activity.