Computação de Alto Desempenho 2016/17
1st Test – 8/4/2017
Duration: 2h

Closed book test; possible doubts about the contents should be solved by the student; please include the assumptions you made in your answers.

1. When writing parallel programs, there are two standard ways of work division: data parallelism and task parallelism. Highlight their differences and why it is necessary to consider carefully how cores will coordinate their work at execution time (consider coordination in the dimensions of communication, load balancing, and synchronization).

2. Assume that you want to evaluate the scability of a program in the domain of big data (the data to be processed is large-scale). Why is it important to make such evaluation and to consider the difference between how the Amdahl law and the Gustafson-Barsis law evaluate a program's scalability? Justify your answers.

3. Recall the parallel matrix-vector multiplication example, $M_{(i,j)}$ x V_j = R_i, presented in class and extracted from the book "An introduction to Parallel Programming" by Peter Pacheco, whose performance results where the following:

Threads	Matrix Dimension					
	$8,000,000 \times 8$		8000×8000		$8 \times 8,000,000$	
	Time	Eff.	Time	Eff.	Time	Eff.
1	0.322	1.000	0.264	1.000	0.333	1.000
2	0.219	0.735	0.189	0.698	0.300	0.555
4	0.141	0.571	0.119	0.555	0.303	0.275

Why did the multi-threaded version of the program with a data organization in a Matrix of 8 lines x 8000000 columns, and V a column vector with 8000000 lines, presented the worst performance/efficiency results? Justify your answer.

4. Consider the problem of calculating the Mandelbrot's figure using the following function:

```
#define max_iterations 255
int compute_point(double ci, double cr) {
   int iterations = 0;
   double zi = 0;
   double zr = 0;
   while ((zr*zr + zi*zi < 4) && (iterations < max_iterations))
   {
        double nr, ni;
        nr = zr*zr - zi*zi + cr; ni = 2*zr*zi + ci;
        zi = ni; zr = nr;
        iterations ++;
   }
   return iterations;
 }
```

This function is to be used in a program to generate an image with size equal to L*H (L is total number of lines, and H is the total number of columns). Moreveor, *after* generating the Mandelbrot figure, the program is supposed to build the so called Mandelbrot's histogram. This histogram is the result of determining how many points have a number of iterations equal to one, how many points have a number of iterations equal to two, and so on, until 255.
Assuming that you have to parallelize the sequential version of such a program with N threads ,

a) discuss how you would apply the Foster methodology to parallelise the program, considering that the mapping is done to a shared memory multiprocessor with *N* CPUs;
b) discuss also, in case of using the API C/OpenMP, which concerns would be necessary to take account of, so that to your solution may be the most efficient as possible.

Justify your options in both questions.

5. Imagine that you were asked to implement a simulation program in order to help health authorities predict the propagation of the *Hepatitis A* disease, due to a recent surge on new cases. Although the virus may infect everyone (i.e. the domain is large), there are groups of people who are more prone to be infected. Health scientists have meanwhile developed a formula for estimating a possible contamination based on groups of population, their age, living habits, location of residence, etc, and that may be used in you program.

Due to the emergency situation, the simulation provides a rough view of the evolution of a possible spreading of the disease and in a local area (e.g. Lisbon). This area, and the groups of people living or working in it, are represented as a matrix with MxN dimensions (M: number of lines; N: number of columns). Each cell in the matrix represents subjects under evaluation (either sick people or people prone to be infected) and their neighbours represent people related with them in some way (e.g. colleagues at school, etc). Each cell may have one of three colors:

--*red*, if the cell represents a contaminated subject;
--*yellow*, if the cell represents a subject with a high probability of becoming contamined;
--*black*, if the cell represents a non-contaminated subject.

Moreover, the simulation considers a limited time frame of 100 days and has to produce the state of the possible propagation, at every 10 days. This means that your program will consider the evolution along 100 states, i.e. time steps take values from *t1* to *t100*, where the t0 state is the current/initial situation in terms of the disease cases. At every 10 time steps, the status of the propagation is to be written to a file with the prefix "LisbonHA" (e.g. "LisbonHA10", "LisbonHA20", etc).
Consider also that the following functions are available to be used by your program:

```
// Reads the initial state
readCurrentState(int matrixLx[M][N], char *filename);
```

```
/* Writes the current state of the matrix. The file name is composed as
strcat(filename, atoi(tag)), e.g. in order to produce different file
names for the different iterations.
*/
writeStateLx(int matrixLx[M][N], int tag, char *filename);

/* Calculates, for a particular cell (i,j) in matrixLx[M][N], its next
value. This function evaluates the cell (i,j) and its relevant
neighbours and produces the cell's next state (i.e. if it will become
red, yellow, or if it remains black).
*/
int nxtCellStatus(int i, int j, int matrixLx[M][N]);
```

Using these functions, complete the following code skeleton with your parallel code (*or pseudo-code*) that uses the *OpenMP C API*, assuming that the execution is done is a shared memory architecture with *T available cores* and enough RAM space for the matrix.

```
#include <stdio.h>
#include <stdlib.h>

#define M  …
#define N  …
int currStLx[M][N], nextStLx[M][N];

int num_threads;

int main (int argc, char *argv[])
{
   // Reads initial state
   readCurrentState(currStLx, "LisbonHA");
   num_threads =


   // your code




   return 0;
 }
```