Computação de Alto Desempenho 2015/16
1st Test – 18/4/2016
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. Based on the existing forms of parallel processing hardware,
  a) define what are *shared memory systems* and *message-passing systems*, and describe which kind of applications benefit most from each type of system;
  b) what are the differences between the *MIMD* (Multiple Instruction Multiple Data) and *SIMD* (Single Instruction Multiple Data) systems?

2. A shared memory multiprocessor (*UMA – uniform memory access*) has 10 similar processors but only processor P0 can execute input/output (I/O) operations. A program is to be executed in this multiprocessor and when executed only on P0 it takes 100 seconds to complete. This time is divided into 10 seconds for reading the program's data and writing the results and 90 seconds for intensive calculation without I/O operations. Write which is the execution time of this program when it is executed on the 10 processors justifying your answer. Explain also which is the speedup that was possible to achieve.

3. Consider the laws of *Amdahl* and *Gustafson-Barsis*:
  a) explain why, when defining the speedup of a program, the Amdahl law is based on a situation where the number of operations is constant;
  b) justify how the Gustafson-Barsis' law overcomes the speedup limitations that are present in the Amdahl's law;
  c) compare both laws in terms of how they define a program's scalability (i.e. explain the way each one evaluates if a program is scalable or not).

4. It is necessary to implement a program to compare two large matrices A and B of equal dimension to identify, at each position, which elements are different, and also to calculate how many different elements were found. The goal is to build a binary matrix C (with zeros and ones) with equal dimension, identifying with the value one, at each corresponding position, if the values of A and B are different (or zero otherwise), and also to write the total number of different elements.
Assume that the space for the matrixes A, B, and C are already defined and that the values of A and B are read from a file, as shown in the following code fragment in C.

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

#define N  50000
int matrix_A[N][N], matrix_B[N][N];
int matrix_C[N][N];
int num_diff;  // contains the number of different elements
int num_threads;

// complete with your worker code

int main (int argc, char *argv[])
{
   FILE *f;
   // Matrices' reading
   f = fopen("input.bin", "r");
   fread(matrix_in1, sizeof(int), N*N, f);
   fread(matrix_in2, sizeof(int), N*N, f);
   fclose(f);
   num_threads = atoi(argv[1]);

   // complete with your code

    // Writing the results
   f= fopen("result.bin", "w");
   fwrite(matrix_C, sizeof(int), N*N, f);
   fclose(f);
   printf("Number of different elements: %d\n", num_diff);
   return 0;
 }
```

Implement a parallel version of this code by using the functions of the Pthreads library to complete the code above, and in the most efficient way possible. You may declare any variables you may find necessary, and you may assume that the number of threads to launch is two. Discuss if your solution may have problems of *false sharing*.

5. Consider the problem of calculating the Mandelbrot's figure, and assume that the following function is available:

```
#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;
 }
```

It is necessary to build the so called Mandelbrot's histogram, i.e. a program that using the function above, determines 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.

Assume that the size of the image to be generated, i.e. its number of points, is defined as lines*columns. Assume that you have N threads, i.e. each thread may run on its own CPU.

a) Apply the Foster methodology to the code above to define a parallel solution and consider that the mapping is done to a shared memory multiprocessor with *N* CPUs.
b) Implement your solution using the API C/OpenMP to be the most efficient as possible. Justify your options.

Annex

A. Some functions of the Pthreads library

`int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`
`int pthread_join (pthread_t thread, void **retval)`
`int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)`
`int pthread_mutex_lock (pthread_mutex_t *mutex)`
`int pthread_mutex_unlock (pthread_mutex_t *mutex)`
`int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr)`
`int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)`
`int pthread_cond_signal(pthread_cond_t *cond)`