Parallel and Distributed Computing

2016/2017 2nd Semester

1st Exam June 16th, 2017 Duration: 2h00

- No extra material allowed. This includes notes, scratch paper, calculator, etc.
- Give your answers in the available space after each question. You can use either Portuguese or English.
- Be sure to write your name and number on all pages, **non-identified pages will not be graded!**
- Justify all your answers.

- Do not hurry, you should have plenty of time to finish this exam. Skip questions that you find less comfortable with and come back to them later on.

I. (1 + 1 + 1,5 + 1,5 = 5 val.)

1. Consider the following parallel version of a function that computes the max on a vector of unsorted integer numbers:

```
int max(int a[], int N)
{
    int i, m = a[0];
    #pragma omp parallel for
    for (i = 0; i < N; i++)
    #pragma omp critical
        if(a[i] > m)
            m = a[i];
}
```

   a) The above implementation is very inefficient. Explain why.

b) Rewrite the function to make it as efficient as you can, having in mind its execution in a machine with a large number of cores.

2. Consider the following two code fragments:

```
#pragma omp parallel sections
{
    #pragma omp section
    f1();
    #pragma omp section
    f2();
    #pragma omp section
    f3();
}

#pragma omp parallel
{
    #pragma omp single
    {
        #pragma omp task
        f1();
        #pragma omp task
        f2();
        #pragma omp task
        f3();
    }
}
```

Explain the differences between their execution.

3. Write down a valid output produced by the code below, assuming that during its execution the value of the environment variable `OMP_NUM_THREADS` is `6`.

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

int main(int argc, char *argv[])
{
    int i, v[10];

    #pragma omp parallel for schedule(static,1)
    for(i = 0; i < 20; i++)
        v[omp_get_thread_num()] = i;

    #pragma omp parallel
    #pragma omp single
    for(i = 0; i < omp_get_num_threads(); i++)
        printf("%d\n", v[i]);

    return 0;
}
```

II. (1,5 + 1,5 + 1 + 1 = 5 val.)

1. In an optimized implementation of the MPI function `MPI_Bcast` (broadcast), how many messages does the source process need to send? Explain. (assume P represents the number of processes and n the size of the array to send)

2. The Foster's design methodology consists of four steps, the second of which is "Communication". What are the objectives of this step and in what way does it help in achieving a more efficient implementation?

3. Consider the following piece of MPI code, where: variable `id` holds the identifier of the MPI task; `P` is the number of processes; `N` is the size of array `arr`.

```
for(i = BLOCK_LOW(id, P, N); i < BLOCK_HIGH(id, P, N); i++) {
    printf("Proc %d: %d, %c\", id, i, arr[i]); fflush(stdout);
}
```

Modify the code above such that (don't worry about the syntax of any MPI routine you use, but make sure all the relevant parameters are there):

a) the indexes over all the pieces of the array are printed in order, *i.e.*, no $i + 1$ before an i.

b) each process, in order, print one position at a time, *i.e.*, process 0 prints index 0, then process 1 prints index 0, and so on until all processes have printed position 0, then process 0 prints index 1, and so on.

III. (1,5 + 1,5 + 1 + 1 = 5 val.)

1. The *Experimentally determined serial fraction* metric is given by

$$e = \frac{\sigma(n) + \kappa(n,p)}{\sigma(n) + \varphi(n)}$$

where $\sigma(n)$ are inherently sequential computations, $\varphi(n)$ are completely parallelizable computations, and $\kappa(n,p)$ represents communication / synchronization / redundant operations.

Describe how we can use this metric to efficiently optimize a parallel program.

2. Discuss the possibility of achieving an efficiency $\varepsilon > 1$ in a parallel system.

3. Consider the problem of computing the sum of each row of a large $n \times n$ matrix. A given parallel MPI implementation, divides the matrix into smaller submatrices, in a checkerboard configuration, computes the sum of each row for such submatrices, and then share this local result such that the overall sum can be computed.

   a) Derive the isoefficiency relation for this parallel implementation, assuming the number of processors to be p. Note: assume reasonable simplifications to facilitate your calculations.

   b) Is this implementation scalable? Justify using the scalability function.

IV. (1 + 1 + 1 + 1 + 1 = 5 val.)

1. a) Why are Monte Carlo methods easy to parallelize?

   b) What is the most critical issue for parallel Monte Carlo methods? Discuss how it is solved in practice.

2. Branch and bound algorithms for optimization problems keep a list of touched nodes in the search tree, ordered in terms of the most promising nodes based on a lower-bound estimate (a measure of what at least the cost function will - for a minimization problem). In a distributed implementation, each computational node keeps a local list. State what are the negative consequences of this approach and how they can be mitigated.

3. A *Maximal Independent Set* I of a graph $G(V, E)$ is a set of vertices $I \subset V$ such that no pair of vertices in I is connected via an edge in G and no other vertex in V can be added to I without violating this rule.

   Luby's algorithm provides a good parallel solution to finding I:

   1. Start with an empty set.
   2. Assign a random number to each vertex.
   3. Vertices whose random number are smaller than all of the numbers assigned to their adjacent vertices are included in the MIS.
   4. Vertices adjacent to the newly inserted vertices are removed.
   5. While graph not empty, Goto 2.

   Analyze its implementation under:

   a) shared-memory

   b) distributed-memory