Computação de Alto Desempenho 2016/17
2nd Test – 19/6/2017
Duration: 2h00

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

1. "Different goals produce different designs". Justify why this sentence is true when comparing CPUs and GPUs, and highlight their major architectural differences.

2. Describe, in a general way, why it is important to be aware of *memory coalescence* in CUDA. Justify if the *tiling technique*, e.g. in matrix multiplication, is related with *memory coalescence* and how.

3. Parallel patterns commonly used in applications are *Histogram*, *Stencil*, *Reduction*, *Scan*, and *Pipeline*.
- **a)** Describe each one in a succint/broad way and identify, for each one, an application domain/example where it may be useful.
- **b)** Justify if the usage of privatisation (*shared memory*) in CUDA is important in the implementation of the *Histogram* and *Stencil* patterns.
- **c)** Does the *MPI API* provide primitives that may simplify the implementation of the *Reduction* and *Scan* patterns in a distributed memory architecture? In case your answer is affirmative for any them or both, give a simple example of its/their usage.

4. Recall the simulation program to predict the propagation of the *Hepatitis A* disease *from the first evaluation test*. Due to the emergency situation, it is now requested that you implement your solution in ***CUDA C***.
Similarly to the first test, 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.

Like in the first test, 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 again available to be used by your program:

```
// Reads the initial state
readCurrentState(stateHA matrix, 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. For simplification, the function
receives a pointer to a matrix as first argument (see the main program).
*/
writeStateLx(stateHAPtr matrix, 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). For simplification, the function
receives a pointer to a matrix as third argument (see the main program).
*/
int nxtCellStatus(int i, int j, stateHAPtr matrix);
```

Using these functions, complete the following code skeleton with your parallel code (*or pseudo-code*) that uses the ***CUDA C*** *API,* assuming that the GPU's memory is enough for two matrices of type ***stateHA***.

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

#define MAXTIME 100
#define OUTT    10
#define M  …
#define N  …
// int currStLx[M][N], nextStLx[M][N]; definition in the first test

// New definition
typedef int stateHA[M][N];    // definition of a new type named
                              // "stateHA", representing the space of a
                              // matrix of integers with dimension M*N.

typedef stateHA * stateHAPtr;  // definition of a new type named
                               // "stateHAPtr" representing a pointer
                               // to a matrix M*N.

stateHA currStLx, nextStLx;  // two matrices of type stateHA,
                             // defined statically, for simplification

/* This implementation aims to highlight how to use a pointer to a
   matrix of type stateHA, in case that may be necessary.
*/
writeStateLx(stateHAPtr currSt, int tag, char *filename)
{
   int i, j;
   FILE *f;
   f = fopen(strcat(filename, atoi(tag)), "w");
   for(i=0; i< M; i++) {
      for(j=0; j< N; j++)
         printf("%d ", (*currSt)[i][j] );
      printf("\n");
```

```
    }
    fclose(f);
}

int main (int argc, char *argv[])
{
    int steps;
    stateHAPtr currState, nextState; //pointers to two matrices of
                                     // type stateHA

    // Reads initial state
    readCurrentState(currStLx, "LisbonHA");
    currState = &currStLx;
    nextState = &nextStLx;
    writeStateLx(currState, 0, "LisbonHA");

    // your CUDA C code for data initialization

    // your kernel function

    for(steps = 0; steps < MAXTIME; steps++ )
    {
          // your CUDA C code
    }

    // your CUDA C code for necessary finalization

    return 0;
 }
```

5. Consider again the simulation program to predict the propagation of the Hepatitis A that was described in ***question 4***.

a) State in which situation would you consider adapting your program to an *MPI implementation* to be deployed in a cluster of personal computers. Justify your answer.

b) In this case, identify how you would plan your program's adaptation to *MPI* and which MPI functions (and their arguments) you would need to use (assume a cluster of PCs similar to what is available in the labs).

c) Adapt your answer in ***b)***, now to consider an *hybrid architecture* (either *OpenMP+MPI* or *Cuda C + MPI*).