The program below is a GPU kernel, implemented in a variation of C for cuda. This kernel computes the averages of the elements stored in each column of a matrix `m`. These averages are stored in the cells of an array `v`. This program is meant to run in parallel, by threads in lock step. So, the threads always process the same instruction, albeit on different data. The work of each thread is given by the thread identifier, the special variable Tid. This is a common pattern: each thread uses its identifier to define the work that it will do.

```
__global__ void
avgSquare(float* m, float* v, int c) {
  if (Tid < c) {
    int d = 0;
    float sum = .0F;
    int N = Tid + c * c;
    for (int i = Tid; i < N; i += c) {
      sum += m[i];
      d += 1;
    }
    v[Tid] = sum/d;
  }
}
```

Figure 1: A GPU kernel that computes averages of columns of a matrix.

1. Consider that this program will process an 8x8 matrix. Mark below the cells that will be read by thread Tid == 3, assuming that `c` == 8

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

UFMG

DCC

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Compilers Laboratory

2. All the threads will process the same function `avgSquare`. They will operate on a stack with a location for every variable. In other words, they will all see the same names of variables. However, some of these variables might have different values. For instance, variable `Tid` always has a different value per thread. Variable `d`, however, always contains the same value for all the threads. Which variables, in the program above, are guaranteed to always hold the same values for all the threads?

3. Can you design a static analysis to pinpoint the variables that are "uniform", that is, that are always seen by all the threads with the same value?

To know more: Diogo Sampaio, Rafael Martins de Souza, Caroline Collange, Fernando Magno Quintão Pereira: *Divergence analysis*. ACM Trans. Program. Lang. Syst. 35(4): 13:1-13:36 (2013)