I.

a) $P + S = 1 \Leftrightarrow S = 1 - P$

Amdahl: $speedup = \frac{1}{\frac{P}{N} + S} = \frac{1}{\frac{P}{N} + (1-P)} \Leftrightarrow 6 = \frac{1}{0.1P + 1 - P} \Leftrightarrow 6 = \frac{1}{1 - 0.9P} \Leftrightarrow 1 - 0.9P = \frac{1}{6}$

$\Leftrightarrow 1 - \frac{1}{6} = \frac{9}{10} P \Leftrightarrow \frac{10}{9}\left(1 - \frac{1}{6}\right) = P \approx 0.93$

$S = 1 - 0.93 = 0.07$

b)

$$speedup = \frac{1}{1 + \frac{P}{N} - P} = \frac{1}{1 + 0.93 \times 0.05 - 0.93} \approx 8.6$$

II

a) A variável sum não pode ser partilhada entre threads, pois isso leva a valores errados serem escritos nela, por isso tem de ser uma variável privada do tipo firstprivate, apenas acessível por cada thread. Para além disso não está a ser efectuado o processo de redução para juntar os valores sum calculados por cada thread numa variável global. Para isso poder-se-ia usar a cláusula reduction (+: sum).

b)

```
sum = 0.0;
#pragma omp parallel for reduction (+: sum)
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        sum += m[i][j];
```

Dentro do ambiente paralelo em que o programa está a correr, é feita uma cópia local de sum inicializada com o valor correspondente à operação "+" da cláusula reduction, que neste caso é 0. As actualizações de sum em cada thread ocorrem na cópia local de sum, assim no final cada thread vai ter um sum com a soma dos elementos de todas a linhas e colunas pelas quais ficou responsável, sendo depois os sum de todos os threads somados entre si e com o sum inicial sum = 0.0, a fim de determinar a soma total global dos elementos da matriz m.

III.

a) paddd → non-saturating addition of double words (32-bit)

```
  BB B0 FF 00 56 78 A5 70
+ 0A 0B 51 06 A0 88 7B 13
-----------------------
  C5 BC 50 06 F7 01 20 83
```

b) paddb → non-saturating addition of byte integers (8-bit)

```
  BB BO FF 00 56 78 A5 70
+ 0A 0B 51 06 A0 88 7B 13
-----------------------
  C5 BC 50 06 F7 01 20 83
```

IV.

a) 512 × 64 pixéis

A = 64
B = 1
C = 512
D = 1
gridDim.x = 64
gridDim.y = 1
blockDim.x = 512
blockDim.y = 1

b) Como a grelha só tem uma dimensão e os blocos também, os threads vão mapear os pixéis a uma dimensão, assim a expressão fica:
pixel = threadIdx.x + blockDim.x * blockIdx.x

c) Um warp é um conjunto de 32 threads que executam em simultâneo numa kernel. Este mecanismo é importante pois é como a kernel processa conjuntos de instruções.