I.

a)

Pela lei de Amdahl: $speedup = \frac{1}{1-0.6} = 2.5$

b)

$$speedup = \frac{1}{\frac{0.6}{40} + 0.4} = 2.4096$$

~~III.~~

IV.

a)

A = 256
B = 1
C = 1024
D = 1
gridDim.x = 256
gridDim.y = 1
blockDim.x = 1024
blockDim.y = 1

b)

~~pixel = threadIdx.x + blockIdx.x * blockDim.x~~

Como a grelha só tem uma dimensão e os blocos também, os threads vão mapear os pixeis a uma dimensão, assim a expressão fica:

pixel = threadIdx.x + blockIdx.x * blockDim.x

c) As memórias disponíveis em CUDA e as respetivas características principais são:

Global Memory: - Acessível pelo host e device, mas de acesso lento.

Texture Memory: - Optimizada para acessos a 2D.

Constant Memory: - Aqui guardam-se os argumentos das kernels e constantes.

Shared Memory: - Memória apenas acessível pelo device e pode ser acedida pelos threads das kernels e é uma memória com acessos mais rápidos, sendo também possível partilhar dados entre threads através dela.

Local Memory: - Usada para guardar dados que não cabem nos registos.

Registers: - Memória muito rápida acessível localmente em cada thread.

III.

a) 0 0 0 0 1 10001 20001 30001 40001 50001 60001 70001

b) igual ↑

II.

a) O problema é não ter usado firstprivate(i)

b)

```
# pragma omp parallel por firstprivate(i) ...
  :
# pragma omp parallel for reduction (=: suma)
  for (i=1; i<N; i++)
    #pragma omp critical
      sumx = s[i];
```