Nota: uma resposta errada nas questões 1 a 6 desconta 0.33 valores

1. [1,0 valores] - Considere código C + OpenMP apresentado abaixo:

```
#pragma omp parallel
{  int i, first=false, tid = omp_get_thread_num ();
   double T;
   printf ("Thread %d starting\n", tid);
   T = omp_get_time ();
   #pragma omp for
     for (i=0; i < 300000 ; i++) do_work(i);
   #pragma omp single
   {
     first = true;
     printf ("Thread %d work done\n", tid);
   }
   if (first) printf ("1st finished in %.0lf us\n", (omp_get_wtime()-T)*1e6);
   printf ("Thread %d finishing\n", tid);
}
```

Para uma execução com 3 *threads* indique qual dos *outputs* abaixo é possível.

☐	Thread 1 starting Thread 0 starting Thread 0 work done 1st finished in 7 us Thread 2 starting Thread 2 finishing Thread 1 finishing Thread 0 finishing		☐	Thread 1 starting Thread 0 starting Thread 2 starting Thread 0 work done Thread 2 finishing 1st finished in 7 us Thread 1 finishing Thread 0 finishing
☐	Thread 1 starting Thread 0 starting Thread 2 starting Thread 2 finishing Thread 0 work done 1st finished in 7 us Thread 1 finishing Thread 0 finishing		☐	Thread 1 starting Thread 0 starting Thread 2 starting Thread 0 work done Thread 0 finishing 1st finished in 7 us Thread 1 finishing Thread 2 finishing

2. [1,0 valores] - Complete a afirmação abaixo :

"O ganho de desempenho obtido com a vectorização de código, relativamente à respectiva versão escalar, deve-se

- ☐ à diminuição do número médio de ciclos por instrução (CPI)."
- ☐ à diminuição do número total de operações matemáticas executadas sobre os dados."
- ☐ a acessos mais rápidos à memória, devidos à maior localidade espacial imposta pelas intruções de `mov` vectoriais."
- ☐ à diminuição do número total de instruções executadas (#I)."

Nome: _____ Número:_____

3. [1,0 valores] - Considere os dois excertos de código apresentados abaixo. Implementam o mesmo algoritmo, sendo que o código da esquerda usa apenas instruções escalares, enquanto o da direita usa algumas instruções vectoriais. Note que as instruções com sufixo 'ss' são escalares e realizam uma operação em vírgula flutuante precisão simples (SPFP); as instruções com sufixo 'ps' são vectoriais e realizam 16 operações em SPFP (AVX512). O registo %ecx tem inicialmente o valor 0. As instruções escalares de acesso à memória têm CPI=3; as instruções vectoriais de acesso à memória têm CPI=6; todas as restantes instruções têm CPI=1.

Escalar	Vectorial

`Ciclo:`
`  vmovass (%ebx, %ecx,4), %zmm1`
`  vaddss %zmm1, %zmm0, %zmm1`
`  vmovass %zmm1, (%ebx, %ecx,4)`
`  addl $1, %ecx`
`  cmpl $16000, %ecx`
`  jne Ciclo`
	`Ciclo:`
	`  vmovaps (%ebx, %ecx,4), %zmm1`
	`  vaddps %zmm1, %zmm0, %zmm1`
	`  vmovass %zmm1, (%ebx, %ecx,4)`
	`  addl $16, %ecx`
	`  cmpl $16000, %ecx`
	`  jne Ciclo`

Qual o ganho da versão vectorial relativamente à versão escalar, isto é, T_{exec_esc} / T_{exec_vec} ?

☐	5.0	☐	7.5
☐	10.0	☐	20.0

4. [1,0 valores] - O código "`for (i=1 ; i<S ; i+=1) a[i]=a[i-1] / (i>10 ? 2.:3.);`" não vectoriza devido…

   ☐ a uma dependência de dados RAW entre iterações.

   ☐ aos dados processados não se encontrarem em posições consecutivas de memória.

   ☐ à possibilidade de *aliasing* entre `a[i]` e `a[i-1]`.

   ☐ à estrutura condicional incluída dentro do ciclo `for`.

5. [1,0 valores] – Um programa P, escrito em OpenMP, executado com uma única *thread* numa máquina com uma frequência de 2 GHz, apresenta um CPI de 1.5 e um tempo de execução de 1.5 segundos.
   O mesmo programa executado em 10 núcleos (com 10 *threads*) executa 10% mais instruções e exibe um *speed up* de 7.5. O $CPI_{percepcionado}$ é

☐	0.75	☐	0.182
☐	0.212	☐	0.15

6. [1,0 valores] – A eficiência exibida na alínea anterior é:

☐	7.5	☐	10%	☐	75%	☐	50%

7. Considere o código apresentado abaixo:

```
#define SIZE 1000000
float a[SIZE];

main () {
   int i;

   for (i=0 ; i < SIZE-1 ; i++) {
          a[i] = powf (a[i], 2) + 10.0 / a[i] – a[i-1];
}  }
```

NOTA: `powf (a, b)` é uma função que calcula a potência a^b

1. [1,0 valores] – Identifique duas razões pelas quais este código não vectoriza.

2. [0,5 valores] – Apresenta, sob a forma de código e justificando, uma solução para uma das razões apontadas acima, mesmo que isso implique apenas uma vectorização parcial do código.

3. [0,5 valores] – Apresenta, sob a forma de código e justificando, uma solução para a outra razão apontada acima, mesmo que isso implique apenas uma vectorização parcial do código.

Nome: _____ Número:_____

8. [2,0 valores] - O código apresentado abaixo, que explora *Thread Level Parallelism* recorrendo ao OpenMP, pretende calcular a soma de alguns elementos de cada linha `i` de uma matriz (elementos das colunas `1` a `i`) e armazenar o resultado no primeiro elemento dessa linha (`a[i][0]`) :

```
#define W 400000
int a[W][W];
int sum, i, j;

...
```

```
#pragma omp parallel for
for (i=0 ; i < W ; i++) {
    sum = 0;
    for (j=1 ; j <= i ; j++) sum += a[i][j];
    a[i][0] = sum;

}
```

a) O resultado da execução deste programa com múltiplas *threads* é indeterminado, pois contém alguns erros semânticos. Identifique esses erros e diga como os corrigiria.

NOTA: Os erros semânticos não estão relacionados com o desempenho, mas sim com a correcção do programa.

b) Associada à directiva `#pragma omp parallel for` usaria para o escalonamento a cláusula `schedule(static)` ou `schedule(dynamic)` ? Justifique