NOTA – as questões 1 a 4 são de escolha múltipla e para cada uma são apresentadas 4 respostas alternativas:

- para cada questão apenas uma das opções está correcta;
- uma resposta correcta vale 1,5 valores;
- uma resposta incorrecta desconta 0,5 valores.

1. [1,5 valores] - Considere código C + OpenMP apresentado abaixo. Note que:

- a cláusula `#pragma omp single` garante que o bloco que se segue é executado apenas por **uma *thread***, podendo esta ser qualquer *thread* do *team*; note ainda que **esta cláusula implica uma barreira no fim**, isto é as *threads* só prosseguem para as instruções seguintes quando todas as *threads* atingirem o fim deste bloco.

```
#pragma omp parallel
{  int i, first=false, tid = omp_get_thread_num ();
   double T;
   printf ("Thread %d starting\n", tid);
   #pragma omp for
     for (i=0; i < 300000 ; i++) do_work(i);
   #pragma omp single
   {  T = omp_get_time ();
     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. [1,5 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. [1,5 valores] - Complete a afirmação abaixo:
"As unidades de processamento gráfico minimizam o impacto dos acessos à memória no desempenho dos programas…"

- ☐ usando memórias e barramentos tão rápidos que estes acessos são satisfeitos sem qualquer impacto no desempenho."
- ☐ explorando essencialmente a hierarquia de memória, pois a localidade exibida nos acessos a memória é normalmente extremamente elevada."
- ☐ comutando rapidamente entre grupos de *threads*, sobrepondo o tempo de acesso de algumas *threads* com a execução de outras."
- ☐ recorrendo a técnicas sofisticadas de previsão dos acessos a dados e iniciando as leituras de memória antecipadamente (*memory prefetching*)."

4. [1,5 valores] – O *loop unrolling* tem potencial para disponibilizar mais instruções para execução em paralelo num contexto de superescalaridade. Para o código abaixo seleccione a opção de *unrolling* que disponibiliza potencialmente mais *instruction level parallelism*.

```
int a[SIZE], i, sum=0;
for (i=0; i < SIZE ; i++) sum +=a[i];
```

☐	`int a[SIZE], i, sum=0;`
	`for (i=0; i < SIZE ; i+=2)`
	`{ sum +=a[i];`
	`  sum +=a[i+1]; }`
			☐	`int a[SIZE], i, sum=0, sum_a=0;`
				`for (i=0; i < SIZE ; i+=2)`
				`{ sum_a +=a[i];`
				`  sum +=a[i+1]; }`
				`sum += sum_a;`

☐	`int a[SIZE], i, sum=0;`
	`for (i=0; i < SIZE ; i+=4)`
	`{ sum +=a[i];`
	`  sum +=a[i+1];`
	`  sum +=a[i+2];`
	`  sum +=a[i+3]; }`
			☐	`int a[SIZE], i, sum=0;`
				`for (i=0; i < SIZE ; i+=4)`
				`{ sum += a[i] + a[i+1];`
				`  sum += a[i+2] + a[i+3]; }`

Nome: _____ Número:_____

5. [2,0 valores] - Considere um processador superescalar com 2 unidades funcionais (UF):

UF1 (Op) – realiza operações lógicas e aritméticas sobre inteiros;

UF2 (LS + B) – realiza acessos à memória (*Load/Store*) e saltos (*branches*).

Considere que cada uma destas unidades funcionais executa **uma instrução por ciclo do relógio** (isto é, não há nenhuma operação que exija mais do que um ciclo do relógio na respectiva UF). Considere ainda o seguinte excerto de código:

```
I1:   movl (%ebx, %edx, 4), %esi
I2:   addl %esi, %eax
I3:   incl %edx
I4:   decl %ecx
I5:   jnz I1
```

Preencha, para a primeira iteração do ciclo, a tabela 1 considerando um escalonamento *static in-order scheduling* e a tabela 2 considerando um escalonamento *dynamic out-of-order*.

Para preencher as tabelas indique, para cada ciclo do relógio, qual a instrução que executa em cada uma das unidades funcionais (use a etiqueta da instrução, isto é, `I1`, `I2`, …). Calcule também o CPI exibido por essa primeira iteração.

Tabela 1 - *static in-order*	
UF1 (Op)	**UF2 (L/S + B)**
CPI =	

	Tabela 2 - *dynamic out-of-order*	
	UF1 (Op)	**UF2 (L/S + B)**
ciclo 1		
ciclo 2		
ciclo 3		
ciclo 4		
ciclo 5		
	CPI =	

Use o espaço abaixo para alguma justificação que lhe pareça pertinente.

Nome: _____ Número:_____

1. 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.

Nome: _____ Número:_____