# Computação Paralela e Distribuída

Exercícios Práticos – Nº 3

## Conjunto de problemas relacionados ao OpenMP

**Coordenação de Engenharia Informática**
Departamento de Engenharias e Tecnologias
Instituto Superior Politécnico de Tecnologias e Ciências

1. Espera que os seguintes programas tenham desempenho diferente? Justifique.

| | |
|---|---|
| `#pragma omp parallel for`<br>`for(i = 0; i < N; i++)`<br>`  A[i] += i;` | `#pragma omp parallel for schedule(static,1)`<br>`for(i = 0; i < N; i++)`<br>`  A[i] += i;` |

2. Forneça uma saída válida para cada um dos dois programas abaixo.
   a) Programa 1

```
#pragma omp parallel private(i) num_threads(2)
for(i = 0; i < 4; i++)
     printf("%d ",i);
```

   b) Programa 2

```
#pragma omp parallel for private(i) num_threads(2)
for(i = 0; i < 4; i++)
     printf("%d ",i);
```

3. Discuta a diferença entre `#pragma omp critical` e `#pragma omp atomic`.
4. Escreva o código para paralelizar esse ciclo usando comandos OpenMP:

```
for(i = 0; i < MAX; i++)
     a[i] += a[i-2] + 5;
```

5. Optimize o código a seguir e escreva uma implementação paralela eficiente em OpenMP.
   a) Código #1

```
n = 7;
for (i = 0; i < N; i++){
     n += 2;
     A[i] = f(n);
}
for (s = 0,i = 0; i < N; i++){
     s += A[i];

}
```

   b) Código #2

```
#define N 30000
for (i = 0; i < N; i++)
{
     if (i % 3 == 0) i++;
     A[i] = i*3 + A[i];

}
```

6. Forneça o conteúdo do array x após a execução do seguinte fragmento de código OpenMP, se possível, ou forneça o erro, se houver. Suponha que existam 5 threads:

```
int i, p, x[5] = {1,1,1,1,1}, y[5]={0,1,2,3,4};
#pragma omp parallel private (p)
{
     p = 5;
     #pragma omp for
     for(i = 0; i < 5; i++)
          x[i] = y[i] + p;
}
```

7. O código a seguir implementa uma ordenação topológica de um grafo direcionado G. Suponha que:
   a) a lista `nodesToProcess` contém inicialmente os nós de G sem arestas de entrada;
   b) `node->level` foi inicializado em 0 para todos os nós;
   c) `incomingEdges` e `outgoingEdges` contêm o número de arestas de entrada e saída de/para um nó, respectivamente.

```
while((node = removeQueue(nodesToProcess)) != NULL)
   for(i = 0; i < node->outgoingEdges; i++){
        child = node->out[i];
        child->level =
             MAX(child->level, node->level + 1);
        child->incomingEdges--;
        if(child->incomingEdges == 0)
             insertQueue(nodesToProcess, child);
}
```

   Escreva uma implementação paralela eficiente em OpenMP.

8. Considere a seguinte implementação paralela do algoritmo de ordenação Selection Sort.

```
void selection_sort(int* a, int N)
{
     int i;
     for (i = 0; i < N; ++i) {
          int k;
          #pragma omp parallel for
          for (k = i+1; k < N; ++k)
               #pragma omp critical
               if (a[k] < a[i]) {
                    int tmp = a[i];
                    a[i] = a[k];
                    a[k] = tmp;
               }
     }
}
```

   a) A implementação acima é muito ineficiente. Explique porquê.

b) Reescreva o código acima para torná-lo o mais eficiente possível.

9. Considere o seguinte código OpenMP, assuma que foi executado em um sistema com 4 threads (OMP_NUM_THREADS=4):

```
#define M 16;
#pragma omp parallel for private(j)
  for (i = 0; i < M; i++ ){
    for ( j = M - (i+1); j < M ; j++ ) {
      // Esta função tem um tempo de computação de 2s
      f(i, j, ...) ;
  }
}
```

**a)** Preencha a tabela a seguir com uma possível alocação de threads das primeiras iterações do ciclo (índice i), assumindo um escalonamento estático definido pela directiva OpenMP schedule(static). Indique qual thread executa cada iteração e quanto tempo essa iteração leva.

Determine o tempo de execução aproximado por thread e o tempo total de execução.

| #iter | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Thread | | | | | | | | | | | | | | | | |
| Tempo/iter (s) | | | | | | | | | | | | | | | | |

| | Thread 0 | Thread 1 | Thread 2 | Thread 3 |
|---|---|---|---|---|
| Tempo de execução de thread individual | | | | |
| Tempo de execução total | | | | |

**b)** Responda a questão anterior assumindo um escalonamento dinâmico definido pela directiva OpenMP `schedule(dynamic,2)`.

| #iter | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Thread | | | | | | | | | | | | | | | | |
| Tempo/iter (s) | | | | | | | | | | | | | | | | |

| | Thread 0 | Thread 1 | Thread 2 | Thread 3 |
|---|---|---|---|---|
| Tempo de execução de thread individual | | | | |
| Tempo de execução total | | | | |

C) Justifique qual dos escalonamentos anteriores seria melhor para um caso genérico (número variável de iterações e threads).

10. Um problema com sistemas de memória partilhada é a consistência da memória, um problema que surge no exemplo a seguir. Suponha que as rotinas direita e esquerda estejam a ser executadas por duas threads diferentes e as variáveis A e B estejam a ser usadas como bloqueios, de modo a garantir que as regiões de código L1 e L2 sejam mutuamente exclusivas (ou seja, a thread esquerda não pode entrar em L1 se a thread direita thread está a executar L2 e vice-versa).

```
T1:    A = 0;              T2:    B = 0;
       ⋮                          ⋮
       A = 1;                     B = 1;
       if(B == 0)                 if(A == 0)
L1:    ⋮                   L2:    ⋮
```

**É sempre garantido que este programa se comporte conforme o esperado? O que pode falhar? O que o sistema precisa implementar para evitar falhas? Existe uma modificação no código que ofereceria a mesma garantia?**

11. Considere o seguinte código:

```
int i,j;
#pragma omp parallel for
for(i = 0; i < 10; i++){
     for(j = 0; j < 10; j++){
          array[i] += buffer[i*10 + j];
          array[j] -= buffer[i*10 + j];
     }
}
```

a) Explique o significado de uma race condition.
b) Explique porquê o código acima tem uma race condition.
c) A seguinte alteração foi introduzida para remover a race condition:

```
int i,j;
#pragma omp parallel for
for(i = 0; i < 10; i++){
     for(j = 0; j < 10; j++){
          array[i] += buffer[i*10 + j];
          #pragma omp critical
          {
               array[j] -= buffer[i*10 + j];
          }
     }
}
```

Porquê essa mudança resolveu (ou não resolveu) o problema? Justifique.