DISCIPLINA	**Computação Paralela e Distribuída**		
CURSO	**Engenharia Informática**		
DISCENTE			
Nº MATRÍCULA		TURMA:	DATA: **27/06/2023**

- Não é permitido o uso de dispositivos electrónicos, excepto calculadora com funções básicas
- A prova é sem consulta
- Responda apenas o que é perguntado
- Justifique todas as suas respostas

GRUPO I. (1 + 1 + 1.5 + 1.5 = 5 val.)

1. Considere a seguinte versão paralela de uma função que encontra o maior valor em um determinado vector de números inteiros não ordenados:

```
int max(int a[], int N)
{
      int i, m = a[0];
      #pragma omp parallel for
      for(i = 0; i < N; i++)
      #pragma omp critical
            if(a[i] > m)
                  m = a[i];
      return m;
}
```

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

b) Reescreva a função para torná-la mais eficiente, tendo em mente que será executada em uma máquina com um grande número de núcleos (unidades de processamento, i.e., cores).

2. Considere os dois fragmentos de código seguintes:

Fragmento 1

```
#pragma omp parallel sections
{
      #pragma omp section
      f1();
      #pragma omp section
      f2();
      #pragma omp section
      f3();
}
```

Fragmento 2

```
#pragma omp parallel
{
   #pragma omp single
   {
      #pragma omp task
      f1();
      #pragma omp task
      f2();
      #pragma omp task
      f3();
   }
}
```

Explique a diferença em termo da execução de cada um.

3. Escreva uma saída válida produzida pelo código abaixo, assumindo que durante sua execução o valor da variável de ambiente `OMP_NUM_THREADS` é 6.

```
#include <stdio.h>
#include <omp.h>

int main(int a, char *b[])
{
     int i, v[10];
     #pragma omp parallel for schedule(static,1)
     for(i = 0; i < 20; i++)
          v[omp_get_thread_num()] = i;
     #pragma omp parallel
     #pragma omp single
     for(i = 0; i < omp_get_num_threads(); i++)
          printf("%d\n", v[i]);
     return 0;
}
```

GRUPO II. (1.5 + 1.5 + 2 + 2 + 2 = 9 val.)

1. Em uma implementação optimizada da função `MPI_Bcast` (broadcast), quantas mensagens o processo origem precisa enviar? Explique. (assuma que **p** representa o número de processos e **n** o tamanho do vector a ser enviado).

2. A metodologia de projecto de Foster consiste de 4 etapas, a segunda etapa é a "Comunicação". Qual é o objectivo desta etapa "Comunicação" e de que forma esta ajuda a alcançar uma implementação mais eficiente?

3. Considere o seguinte código MPI, onde A e B são vectores:

```
MPI_Comm_size(comm, &p);
MPI_Comm_rank(comm, &id);
dest = (id + 1) % p;
source = (id - 1) % p;
MPI_Send(A, size, MPI_INT, dest, tag, comm);
MPI_Recv(B, size, MPI_INT, source, tag, comm, &status);
```

a) Descreva qual é o problema que este código pode causar?

b) Discuta de forma simples as possíveis alternativas de solução?

4. Considere um programa MPI a executar com 3 processos, inicialmente todos com vectores `A={1,2,3,4,5,6}` e `B={7,8,9,10,11,12}`. Mostre os valores de A e B em todos os 3 processos após a execução das seguintes rotinas (assuma que A é sempre a fonte e B o receptor):

a) broadcast (`MPI_Bcast`).

b) alltoall (`MPI_Alltoall`).

5. O `MPI_Scatter` utiliza os seguintes parâmetros:

```
MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
MPI_Comm comm)
```

a) Escreva a implementação mais trivial da função `MPI_Scatter` em C. Utilize apenas o `MPI_Send` e `MPI_Recv` como funções de comunicação.

Funções que podem ser úteis:

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int
dest, int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int
source, int tag, MPI_Comm comm, MPI_Status *status)

int MPI_Comm_size( MPI_Comm comm, int *size)

int MPI_Type_size(MPI_Datatype datatype, int *size)

void *memcpy(void *dst, const void *restrict src, size_t n)
```

GRUPO III. (1.5 + 1.5 + 2 + 1 = 6 val.)

1. A métrica **fracção serial determinada experimentalmente** representa a fracção do programa original que não pode ser paralelizado em relação ao tempo de execução sequencial e é dado por

$$e = \frac{\sigma(n) + \kappa(n,p)}{\sigma(n) + \varphi(n)}$$

Onde $\sigma(n)$ são as computações inerentemente sequenciais, $\varphi(n)$ são as computações completamente paralelizáveis, e $\kappa(n,p)$ representa a comunicação / sincronização / operações redundantes.

p	2	4	6	8
e	0,125	0,125	0,126	0,127

A tabela acima indica a fracção serial determinada experimentalmente, **e**, obtido de um dado programa em um sistema paralelo, com número variado de processadores, **p**.

Utilizando este resultado, discuta o que deve ser feito no programa paralelo para torna-lo mais eficiente.

2. Considere um algoritmo com a seguinte complexidade:

Complexidade do algoritmo sequencial: $T(n, 1) = \Theta(n^3)$

Algoritmo paralelo:

- Complexidade computacional: $\Theta(n^3/p)$
- Complexidade da comunicação: $\Theta(n^2 \log p)$

Sobrecarga paralela: $T_0(n, p) = \Theta(pn^2 \log p)$

a) Determine a relação de isoeficiência.

b) Determine a função de escalabilidade. É um bom indicador? Forneça uma justificativa cuidadosa em termos da definição da função de escalabilidade.

3. Um programa de animação por computador gera uma longa-metragem quadro a quadro. Cada quadro pode ser gerado independentemente e enviado para seu próprio ficheiro. Requer 99 segundos para renderizar um quadro e 1 segundo para exibi-lo na tela. Determine a aceleração máxima do programa se renderizarmos o filme em 100 processadores?

.

Bom trabalho!