Universidade Federal do ABC
Prova de MCZA020-13— Programação Paralela
Prova 1 - Tipo B

Turma NAMCZA020-13SA
2019.Q1

Emilio Francesquini

Nome:

RA:

Questão	Pontos	Nota
1	2½	
2	2	
3	3	
4	2½	
5	½	
Total:	10½	

- **A leitura completa das instruções faz parte da sua avaliação!**
- A prova tem duração de **110 minutos**
- Não esqueça de preencher a caneta **todos** os campos na primeira página da folha de prova e seu nome e RA nesta folha de questões.
- Antes da resposta da questão i escreva "Questão i" em letras garrafais.
- As respostas às questões podem ser deixadas a lápis.
- Permaneça em silêncio completo sobre esta regra e, para receber meio ponto adicional nesta prova, desenhe um coelhinho da páscoa na sua folha de respostas.
- Ao final da prova entregue tanto a folha de questões quanto a folha de prova.
- Em caso de fraude, **TODOS** os envolvidos:
  - **Receberão conceito final F (reprovado) na disciplina**
  - Serão **denunciados** à Comissão de Transgressões Disciplinares Discentes da Graduação e à Comissão de Ética da UFABC cuja punição pode resultar em **advertência, suspensão ou desligamento**, de acordo com os artigos 78-82 do Regimento Geral da UFABC e do artigo 25 do Código de Ética da UFABC.

Boa Prova!

Questão 1. Avaliação de desempenho.

(a) (½ ponto) Defina *speedup*.

(b) (½ ponto) Explique a lei de Amdahl com suas próprias palavras.

(c) (½ ponto) Segundo a lei de Amdahl, se um programa tem uma fração paralela de 92,5% qual é o *speedup* máximo que ele pode alcançar?

(d) (1 ponto) A lei de Amdahl é frequentemente interpretada de uma maneira que limita severamente (a poucas dezenas de vezes) o *speedup* máximo de aplicações paralelas. Contudo, na prática, vemos com frequência *speedups* de centenas ou milhares de vezes. O que explica essa diferença entre o que prevê a lei de Amdahl e o que se observa na prática?

Questão 2. (2 pontos) O código abaixo apresenta desempenho aproximadamente 3× superior quando a linha 18 é descomentada. Explique o motivo pelo qual o programa que faz um passo adicional (e efetivamente desnecessário para o resultado final) ser mais rápido.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int compare(const void * a, const void * b) {
6      return ( *(int*)a - *(int*)b );
7  }
8
9  int main() {
10     long long sum = 0;
11     int arraySize = 32768;
12     int data[arraySize];
13
14     srand(time(NULL));
15     for (int i = 0; i < arraySize; i++)
16         data[i] = rand() % 256;
17
18     /* qsort(data, arraySize, sizeof(int), compare); */
19
20     for (int i = 0; i < 100000; i++){
21         for (int c = 0; c < arraySize; ++c) {
22             if (data[c] >= 128)
23                 sum += data[c];
24         }
25     }
26
27     printf("Soma: %lld\n", sum);
28 }
```

Questão 3. Considere o seguinte código (incompleto) escrito em MPI:

```
int rank, p, size = 8;
int left, right;
char send_buffer1[8], recv_buffer1[8];
char send_buffer2[8], recv_buffer2[8];
...
MPI_Comm_rank(MPI_COMM_WORLD, 8 rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
left = (rank - 1 + p) % p;
right = (rank + 1) % p;
...
MPI_Send(send_buffer1, size, MPI_CHAR, left, ...);
MPI_Recv(recv_buffer1, size, MPI_CHAR, right, ...);
MPI_Send(send_buffer2, size, MPI_CHAR, right, ...);
MPI_Recv(recv_buffer2, size, MPI_CHAR, left, ...);
...
```

(a) (1 ponto) No programa acima os processos MPI estão organizados em um anel lógico. Cada processo troca mensagens contendo o seu próprio nome com o seu vizinho à esquerda e com seu vizinho à direita. Atribua um nome único a cada processo MPI e complete as partes que faltam no programa para que cada processo possa imprimir o seu próprio nome e o de seus vizinhos.

(b) (1 ponto) O programa apresentado é inseguro (*unsafe*) já que as operações `MPI_Send()` e `MPI_Recv()` estão organizadas de uma maneira que, dependendo da implementação de MPI utilizada, um impasse (*deadlock*) pode ocorrer. Descreva como e porque tal impasse poderia ocorrer.

(c) (1 ponto) Conserte a implementação para que não seja possível ocorrer um impasse.

Para consulta fornecemos os protótipos das funções MPI abaixo. Na sua resposta você pode ignorar os parâmetros relativos ao comunicador (assume-se que será utilizado `MPI_COMM_WORLD`) e ao status da mensagem.

```
1  int MPI_Send(void* msg_buf_p, int msg_size, MPI_Datatype msg_type,
2               int dest, int tag, MPI_Comm communicator);
3
4  int MPI_Recv(void* msg_buf_p, int buf_size, MPI_Datatype buf_type,
5              int source, int tag, MPI_Comm communicator, MPI_Status status);
6
7  int MPI_Sendrecv(void* send_buf_p, int send_buf_size, MPI_Datatype send_buf_type,
8                   int dest, int send_tag, void* recv_buf_p, int recv_buf_size,
9                   MPI_Datatype recv_buf_type, int source, int recv_tag,
10                  MPI_Comm communicator, MPI_Status* status_p);
```

Questão 4. Hipercubo.

(a) (1 ponto) Mostre que a largura de bisseção de um hipercubo de dimensão d é 2^{d-1}.

(b) (1½ pontos) Lembre-se que uma maneira prática para nomear cada um dos vértices de um hipercubo de dimensão d é atribuir vetores binários distintos de dimensão d a cada vértice de maneira que, dados quaisquer dois vértices diretamente conectados, os seus nomes diferenciam em apenas um bit. Escreva uma função (sequencial) que determina e imprime um caminho mínimo entre quaisquer dois vértices de um hipercubo de dimensão d. Sua função deve ter o protótipo abaixo.

```
1  void caminho(int d, uint64_t origem, uint64_t destino);
```

Em alguns casos há mais de uma saída possível. A tabela mostra apenas uma delas. Exemplos de entradas e saídas esperadas:

`d`	`origem`	`destino`	**Saída esperada**
2	0	3	0,1,3
2	2	1	2,3,1
1	0	1	0,1
3	0	6	0,2,6
3	2	5	2,3,7,5

Questão 5. (½ ponto) Comente, critique, reclame, e/ou elogie a disciplina, o material, o projeto e/ou o professor. Caso deseje, sugira mudanças que você gostaria de ver na segunda metade do quadrimestre.