Universidade Federal do ABC
Prova de MCZA020-13— Programação Paralela
Prova 2 - Tipo B

Turma NAMCZA020-13SA
2019.Q1

Emilio Francesquini

Nome:

RA:

Questão	Pontos	Nota
1	3	
2	2½	
3	3½	
4	2	
Total:	11	

- **A leitura completa das instruções faz parte da sua avaliação!**
- 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.
- 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. Nesta disciplina exploramos em detalhes três ferramentas para a programação paralela: MPI (1), PThreads (2) e OpenMP (3). Também falamos sobre outras técnicas para programação concorrente e paralela: memória transacional (4), modelo de atores (5), Intel Thread Building Blocks (6) e MapReduce (7). Sobre essas ferramentas responda:

(a) (½ ponto) Classifique cada uma dessas 7 ferramentas quanto à sua adequação para a programação de máquinas com memória distribuída e máquinas com memória compartilhada.

(b) (1 ponto) No primeiro projeto implementamos uma versão paralela do Crivo de Eratóstenes. Essa paralelização foi feita utilizando MPI. Descreva em detalhes como essa implementação poderia ser adaptada para uma das seguintes ferramentas: *modelo de atores* ou *Intel Thread Building Blocks* ou *MapReduce*. Quais seriam as vantagens e desvantagens da sua abordagem com relação a sua implementação usando MPI?

(c) (1½ pontos) Descreva detalhadamente, inclusive utilizando ilustrações se necessário, como você implementou a paralelização do Sokoban. Não deixe de citar qual tecnologia de paralelização foi utilizada, como ela foi utilizada e como você contornou os problemas de consumo de memória.

Questão 2. Considere a seguinte implementação para um mecanismo de travas (*locks*):

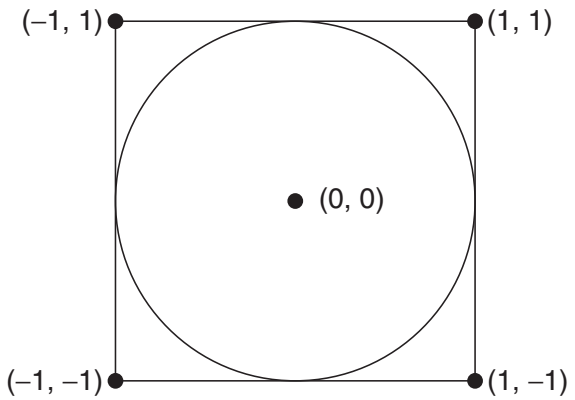
```
1  int flag[2];
2
3  int lock() {
4      int outro = 1 - meuRank();
5      flag[meuRank()] = 1;
6      while (flag[outro]); //Busy-wait (laço de espera ocupada)
7  }
8
9  int unlock() {
10     flag[meuRank()] = 0;
11 }
```

Assuma que a função `meuRank()` devolve um inteiro com o rank do thread que efetuou a chamada. Suponha também que 2 threads com ranks 0 e 1 executem este código para acessar uma estrutura de dados de maneira concorrente. Em outras palavras ambas as threads protegem a entrada da seção crítica de código efetuando uma chamada à função `lock()` e indicam a saída da seção crítica utilizando uma chamada à função `unlock()`. Responda e justifique a sua resposta para as seguintes perguntas:

(a) (1½ pontos) As chamadas às funções `lock` e `unlock` para delimitar as regiões críticas garantem exclusão mútua das threads?

(b) (1 ponto) A implementação acima pode causar um deadlock?

Questão 3. Uma maneira interessante de calcular π é utilizando o método de Monte Carlo baseado em aleatoatorização. Esta técnica funciona da seguinte maneira: suponha que haja uma circunferência inscrita em um quadrado tal qual é mostrado na figura abaixo:



Assumindo que o raio da circunferência é 1, gere uma sequencia de pontos aleatórios (x, y). Estes pontos precisam estar localizados dentro do quadrado. Do número total

de pontos aleatórios que forem gerados alguns vão estar dentro da circunferência. Em seguida estime π através do seguinte cálculo:

$\pi = 4 \times$ (número de pontos na circunferência)/(número total de pontos gerados)

O código abaixo implementa este algoritmo:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  int i;
  double x, y, d;
  unsigned long long dentro = 0;
  unsigned long long sorteios = 0;

  srand(time(NULL));

  scanf("%llu", &sorteios);

  for (i = 0; i < sorteios; i++) {
    x = rand() / (double)RAND_MAX;
    y = rand() / (double)RAND_MAX;
    d = x * x + y * y; //não é preciso tirar a raiz. Pq?
    if (d <= 1)
       dentro++;
  }

  printf("PI: %lf\n", 4 * dentro / (double)sorteios);

  return 0;
}
```

(a) (2 pontos) Escreva uma versão paralela do código acima utilizando Pthreads.

(b) (1 ponto) Escreva uma versão paralela do código acima utilizando OpenMP.

(c) (½ ponto) Critique a paralelização desenvolvida no item anterior. Dê a sua opinião (justificada) sobre o desempenho que pode-se esperar da sua paralelização.

Questão 4. Sejam x e y dois números em $\mathbb{R}^n$. Considerando uma implementação utilizando variáveis do tipo `double`, a operação DAXPY (sigla em inglês para *Double precision Alpha Times X Plus Y*) é definida como:

$$y = \alpha x + y$$

(a) (1½ pontos) Escreva um programa em PThreads ou OpenMP que efetue uma operação DAXPY entre x e y (dados como dois vetores de tamanho `n`). Seu código deve ser paralelo e acessível por uma função com o protótipo abaixo:

```c
void daxpy(unsigned int n, double *x, double *y);
```

(b) (½ ponto) Para valores grandes de n, descreva o desempenho esperado da sua implementação. Ela é escalável? Particionamentos por blocos ou cíclicos teriam diferenças de desempenho? Justifique suas respostas.