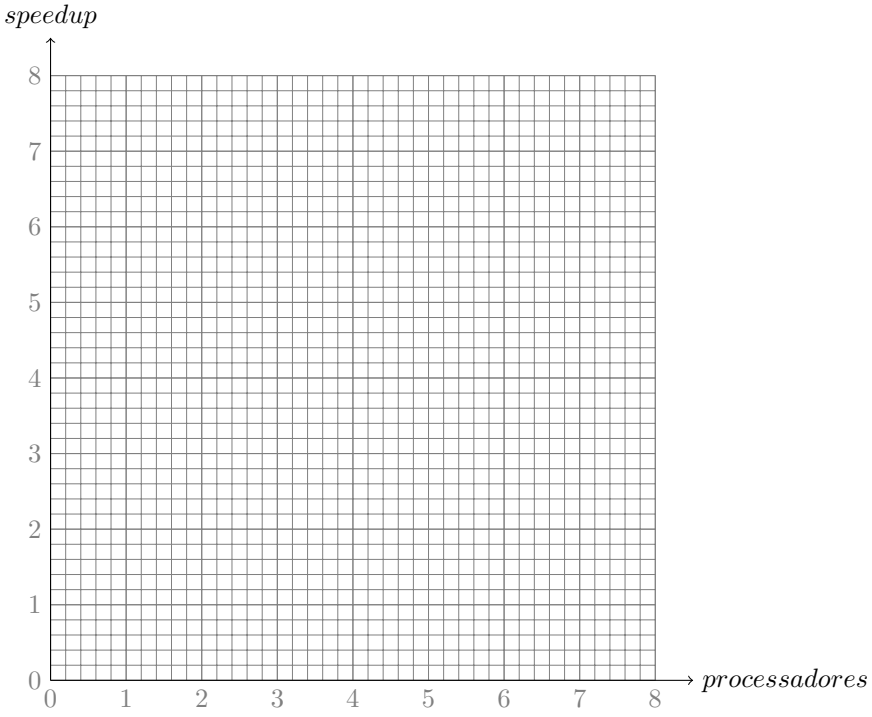
Ciência da Computação **Paralelismo**
Escola Politécnica da PUCPR **Lista de Exercícios**
Prof. Alcides Calsavara **2016–2019**

Questão 1 .....

Um algoritmo paralelo `p` foi implementado e testado sobre um computador com 8 processadores, de forma que os seguintes tempos médios (em milissegundos) de execução foram obtidos para cada quantidade de processadores usada, respectivamente para 1, 2, 4 e 8 processadores: 1800, 1000, 600, 450. Construa o gráfico de *speedup* para a implementação de `p`. Qual a eficiência dessa implementação para 2, 4 e 8 processadores?

Cálculo do speedup para 2, 4 e 8 processadores:

Gráfico de speedup:



Cálculo da eficiência para 2, 4 e 8 processadores:

Questão 2

Considere o algoritmo paralelo `p` da Questão 1. De acordo com a Lei de Amdahl, se esse algoritmo é parte de um processamento maior `K`, sendo que essa parte corresponde a 60% do tempo de execução sequencial de `K`, isto é, com apenas um processador à disposição, qual o limite do *speedup* de `K` para 8 processadores?

Questão 3

Considere uma sequência $X = \{x_0, x_1, \ldots, x_n, x_{n+1}\}$, onde $x_i \in \mathbb{R}$ e $n = 2^k, k \in \mathbb{N}$, e a função $f(x_i)$ assim definida:

$$f(x_i) = \sqrt{\frac{x_{i-1} - x_{i+1}}{1 + x_i}}$$

Escreva o pseudocódigo de uma aplicação paralela que gere a sequência $Y = \{y_1, \ldots, y_n\}$, onde $y_i = f(x_i), 1 \leq i \leq n$, e calcule M assim definido:

$$M = \frac{\sum_{i=1}^{n} y_i}{n}$$

A aplicação deve fazer o máximo uso de processadores disponíveis, limitado a n. Toda a sincronização entre threads concorrentes deve, necessária e somente, ser feita com uso de semáforos. Não é necessário escrever o pseudocódigo da função $f(x_i)$, isto é, a aplicação pode simplesmente assumir a sua existência e chamá-la.

Questão 4

Escreva o pseudocódigo de uma aplicação (composta por um ou mais tipos de threads) que recebe uma matriz $A_{m,n}$ (m linhas e n colunas) de valores reais maiores que zero e gera uma matriz $B_{m,n}$ na qual cada elemento é o valor normalizado com respeito à sua coluna tal que, sendo $MIN(j)$ o menor valor presente na coluna j da matriz A, $B[i, j] = A[i, j]/MIN(j)$. Depois de finalizados todos os cálculos para obtenção da matriz B, a aplicação deve imprimir o maior valor presente em B. A matriz B deve ser calculada em paralelo, fazendo-se o máximo uso de processadores disponíveis, limitado ao número de colunas da matriz. Toda a sincronização entre threads concorrentes deve, necessária e somente, ser feita com uso de semáforos.

Questão 5 .....

Considere o procedimento `p` definido pelo pseudo-código abaixo, o qual modifica os elementos de uma matriz $A_{m,n}$ de valores inteiros passada como parâmetro.

```
procedimento p(A, m, n)
{
   int temp[n];

   para j de 0 a n-1 faça
      temp[j] = A[0,j];

   para i de 0 a m-2 faça
      para j de 0 a n-1 faça
         A[i,j] = A[i,j] + A[i+1,j];

   para j de 0 a n-1 faça
      A[m-1,j] = A[m-1,j] + temp[j];
}
```

Escreva uma versão paralela do procedimento `p` que faça uso de k processadores.

Questão 6 .....

Considere a função `Fib` (*Fibonacci*), definida da seguinte forma:

```
Fib(0) = 0
Fib(1) = 1
Fib(n) = Fib(n-1) + Fib(n-2)
```

Escreva uma versão paralela da função `Fib` que faça uso de k processadores.

Questão 7 .....

O *Problema das 8 Rainhas* consiste em colocar 8 rainhas nas casas de um tabuleiro de xadrez de forma que nenhuma rainha seja ameaçada por outra, lembrando que um tabuleiro tem 64 casas organizadas em 8 linhas e 8 colunas e que uma rainha pode movimentar-se livremente na horizontal, na vertical e nas diagonais. Existem apenas 92 soluções para o problema, entre os 4.426.165.368 possíveis arranjos das rainhas. Obviamente, em uma solução, não podem existir duas rainhas na mesma linha, tampouco na mesma coluna. Por outro lado, necessariamente, uma solução deve ter uma rainha em cada linha e em cada coluna. A Figura 1 ilustra uma solução na qual as casas ocupadas por rainhas são: [1,1], [2,7], [3,4], [4,6], [5,8], [6,2], [7,5] e [8,3].

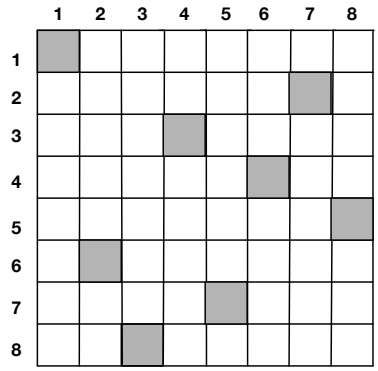


Figura 1: Uma solução para o Problema das 8 Rainhas

O algoritmo definido pelas funções `rainhas` e `tente`, conforme o pseudo-código abaixo, encontra uma, e somente uma, dessas soluções. A função `tente` é uma função recursiva que aplica a estratégia de *backtracking*: é feita uma tentativa de movimento (colocação de rainha) e, caso não seja possível encontrar uma solução a partir desse movimento, o mesmo é desfeito. A função assume que as linhas e colunas do tabuleiro são numeradas de 1 a 8. O parâmetro `i` corresponde à linha do tabuleiro que se deseja tentar colocar a correspondente rainha. Por isso, na primeira chamada da função `tente` (feita pela função `rainhas`), o parâmetro `i` tem o valor 1; posteriormente, a função chama a si própria com `i+1`.

Como existe uma solução com a rainha da primeira linha na primeira coluna (Figura 1), o algoritmo nem tenta encontrar soluções para essa rainha nas demais colunas da primeira linha. Modifique o algoritmo de forma que seja procurada uma solução – caso exista – para cada coluna da primeira linha, sendo que as 8 procuras ocorram em paralelo (cada procura deve executar em um processador).

```
01  int tabuleiro[8,8]; // todos os valores são iniciados com zero
02
03  boolean tente(int i) // 1 <= i <= 8
04  {
05     boolean OK = false;
06     int j = 0;
07     repita
08        j++;
09        tabuleiro[i,j] = 1; // coloque a i-ésima rainha na casa [i,j] do tabuleiro
10        se a i-ésima rainha está ameaçada
11           então
12              tabuleiro[i,j] = 0; // remova a i-ésima rainha do tabuleiro
13           senão
14              se i == 8
15                 então // a última rainha foi colocada com sucesso
16                    OK = true;
17                 senão // tente a próxima rainha
18                    OK = tente( i+1 );
19                    se OK == false
20                       então // não há solução com a i-ésima rainha na coluna atual
21                          tabuleiro[i,j] = 0; // remova a i-ésima rainha do tabuleiro
22     até OK ou j == 8;
23     retorne OK;
24  }
25
26  procedimento rainhas( )
27  {
28     boolean sucesso = tente( 1 );
29     se sucesso então imprima o tabuleiro;
30  }
```