Lista 1 - Daniel Machado

1️⃣ No programa de multiplicação de matrizes mostramos (Cap 2 - texto) uma forma de paralelizar o algoritmo de multiplicação de matrizes criando um fluxo de execução independente para calcular cada um dos elementos de matriz de saída. Proponha outra solução onde a tarefa de cada fluxo de execução seja calcular uma linha inteira da matriz de saída.

```
#define N 1000 //N igual à dimensão da matriz
float a[N][N], b[N][N], c[N][N];
void calculaElementoMatriz(int dim, int i) {
  int k, j, soma = 0.0;
  for(k = 0; k < dim; k++) {
    for(j = 0; j < dim; j++) {
      soma = soma + a[i][j] * b[j][k];
    }
  }
    c[i][k] = soma;
    soma = 0;
}

void main() {
  int i, j;
  //inicializa as matrizes a e b (...)
  //faz C = A * B
  for(i = 0; i < N; i++) {
    //dispara um fluxo de execução f para executar:
    //calculaElementoMatriz(N, i);
    }
  }
}
```

2️⃣ Para arquiteturas de hardware com poucas unidades de processamento (como é o caso das CPUs multicores) geralmente é melhor criar uma quantidade de fluxos de execução igual ao número de unidades de processamento. Altere a solução do exercício anterior fixando o número de fluxos de execução e dividindo o cálculo das linhas da matriz de saída entre eles.

```
#define N 1000 //N igual à dimensão da matriz
#define N_THREADS
float a[N][N], b[N][N], c[N][N];
void calculaElementoMatriz(int dim, int i) {
  int k, j, soma = 0.0;
  //divide o cálculo das linhas em pedaços para cada thread.
  for(int l = i; i < dim; l += N_THREADS){
    for(k = 0; k < dim; k++) {
      for(j = 0; j < dim; j++) {
        soma = soma + a[i][j] * b[j][k];
      }
    }
      c[i][k] = soma;
      soma = 0;
```

```
  }
}

void main() {
  int i, j;
  //inicializa as matrizes a e b (...)
  //faz C = A * B
  for(i = 0; i < N_THREADS; i++) {
    //dispara um fluxo de execução f para executar:
    //calculaElementoMatriz(N, i);
    }
  }
}
```

3️⃣ A série mostrada abaixo pode ser usada para estimar o valor da constante π. A função piSequencial() implementa o cálculo dessa série de forma sequencial. Proponha um algoritmo concorrente para resolver esse problema dividindo a tarefa de estimar o valor de π entre M fluxos de execução independentes.

$$\pi - 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ...)$$

```
double piSequencial (long long n) {
  double soma = 0.0, fator = 1.0;
  long long i;
  for(i = 0; i < n; i++) {
    soma = soma + fator / (2 * i + 1);
    fator = - fator;
  }
  return 4.0 * soma;
}
```

```
#define N_THREADS //quantidade de threads

double global_sum = 0.0; //variável global responsável por armazenar o resultado da função pi_concorrente
long long int N = 0; //valor de N termos;

void *pi_concorrente(void *i_thread){
  double local_sum = 0.0;
  int factor = 1;
  long long int k;
  int i = (int) i_thread;
  for(k = i; k < N; k += N_THREADS){
    if(i % 2 == 0){
      local_sum += (factor / (2 * k + 1));
      continue;
    }
    local_sum -= (factor / (2 * k + 1));
  }
  //Início SC
  global_sum += local_sum
  //Fim SC
}

int main(){
  double pi = 0;
  //inicializa valores de M e n (...)
  //aproximação de pi
  for(int i=0; i<N_THREADS; i++) {
    //dispara um fluxo de execução f para executar:
    //pi_concorrente((void *) i);
```

```
    double pi_estimado = 4 * global_sum;
}
}
```

4️⃣ A série infinita mostrada abaixo estima o valor de $\log(1+x)$, $(-1 < x < 1)$.

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - ...)$$

Dois programas foram implementados para calcular o valor dessa série (um programa sequencial e outro concorrente) usando N termos. Após a implementação, foram realizadas execuções dos dois programas, obtendo as medidas de tempo apresentadas na Tabela 1. A coluna N informa o número de elementos da série, a coluna thread informa o número de threads, e as colunas T_s e T_c informam os tempos de execução do programa sequencial e do programa concorrente, respectivamente.

N	$threads$	T_s (s)	T_c (s)	A
1×10^6	1	0,88	0,89	
1×10^6	2	0,88	0,50	
1×10^7	1	8,11	8,34	
1×10^7	2	8,11	4,44	
2×10^7	1	16,21	16,41	
2×10^7	2	16,21	8,84	

1. **Complete a coluna A com os valores de aceleração.**

   De acordo com a seção ***4.2 Aceleração*** do capítulo 2 do livro texto, a aceleração de um programa concorrente em relação à um programa sequencial pode ser calculada de acordo com a seguinte fórmula:

   $$A(n,p) = \frac{T_s(n)}{T_c(n,p)}$$

   Onde n corresponde ao tamanho do problema, p corresponde ao número de processadores, $T_s(n)$ corresponde ao tempo do programa sequencial para resolver o problema de tamanho n, $T_c(n,p)$ corresponde ao tempo do programa concorrente para resolver o problema de tamanho n usando p processadores. Desta maneira, efetuando todos os cálculos necessários temos:

LINHA	**N**	THREADS	Ts	Tc	A
1	1 x 10^6	1	0,88	0,89	0.98
2	1 x 10^6	2	0,88	0,50	1,76
3	1 x 10^7	1	8,11	8,34	0,97
4	1 x 10^7	2	8,11	4,44	1,82

LINHA	**N**	THREADS	Ts	Tc	A
5	2 x 10^7	1	16,21	16,41	0,98
6	2 x 10^7	2	16,21	8,84	1,83

2. **Avalie os resultados obtidos para essa métrica. Considere os casos em que a carga de dados aumenta junto com o numero de processadores e os casos isolados onde apenas a carga de trabalho ou o numero de processadores aumenta.**

   Quando a carga de trabalho aumenta junto com o número de processadores (comparando linhas 1 e 4, por exemplo) notamos uma melhora drástica no desempenho do programa, quase diminuindo pela metade o tempo de execução deste.

   O mesmo ocorre para quando o número de processadores aumenta e a carga de trabalho se mantém a mesma (linhas 3 e 4), o desempenho do programa quase dobra.

   Quando a carga de trabalho aumenta, e o número de processadores se mantém o mesmo (linhas 1 e 3), podemos perceber que não há um ganho significativo que justifique a implementação do programa concorrente.

> 5️⃣ Considere uma aplicação na qual 20% do tempo total de execução é comprometido com tarefas sequenciais e o restante, 80%, pode ser executado de forma concorrente.

1. Se dispusermos de uma maquina com 4 processadores, qual será a aceleração teórica (de acordo com a lei de Amdahl) que poderá ser alcançada em uma versão concorrente da aplicação?

   A aceleração teórica calculada de acordo com a lei de Amdahl pode ser calculada através da seguinte fórmula:

$$A_t = \frac{1}{(1-p) + (\frac{p}{n})}$$

   Onde p corresponde a fração do programa que pode ser paralelizada, e n corresponde ao número de processadores. Desta maneira, partindo das informações obtidas através do enunciado podemos calcular estimar a aceleração teórica da seguinte maneira:

$$A_t = \frac{1}{(1-\frac{4}{5}) + (\frac{\frac{4}{5}}{4})}$$

$$A_t = \frac{1}{(\frac{5}{5}-\frac{4}{5}) + (\frac{1}{5})}$$

$$A_t = \frac{1}{\frac{1}{5}+\frac{1}{5}} = \frac{1}{\frac{2}{5}}$$

$$A_t = \frac{5}{2} = 2,5$$

   Desta maneira, podemos concluir que haverá uma aceleração teórica de $2,5$.

2. Se apenas 50% das atividades pudessem ser executadas em paralelo, qual seria a aceleração teórica considerando novamente uma máquina com 4 processadores?

Utilizando a mesma fórmula do exercício anterior, no entanto para $p = \frac{1}{2}$, temos:

$$A_t = \frac{1}{(1 - \frac{1}{2}) + (\frac{\frac{1}{2}}{4})}$$

$$A_t = \frac{1}{(\frac{2}{2} - \frac{1}{2}) + (\frac{1}{8})}$$

$$A_t = \frac{1}{\frac{1}{2} + \frac{1}{8}} = \frac{1}{\frac{4}{8} + \frac{1}{8}}$$

$$A_t = \frac{1}{\frac{5}{8}} = \frac{8}{5} = 1,6$$

Sendo assim, podemos concluir que haverá uma aceleração teórica de $1,6$.