UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
DIM0124 – PROGRAMAÇÃO CONCORRENTE

Trabalho Prático

Programação com *Threads*

Objetivo

O objetivo deste trabalho é verificar se é válido o senso comum de que programas concorrentes implementados utilizando threads apresentam um melhor desempenho com relação a tempo de execução quando comparados a suas respectivas versões sequenciais. Para tal, serão implementadas as versões sequencial e concorrente de um algoritmo e realizados testes para diferentes cargas (*workloads*). Através da análise do tempo de execução registrado em cada um dos testes, será possível concluir acerca da hipótese inicialmente dada.

1 Introdução

Na Matemática, uma *matriz* é um arranjo retangular de números, símbolos ou expressões, chamados de *entradas* ou *elementos*. Tais elementos são dispostos de forma horizontal em *linhas* e de forma vertical em *colunas*. O número de linhas e colunas de uma matriz determina a sua *dimensão*: se uma determinada matriz possui m linhas e n colunas ($m, n \in \mathbb{N}^+$), diz-se que ela é uma matriz m por n, ou possui dimensão $m \times n$. Por exemplo, a matriz

$$\begin{pmatrix} 5 & 13 & -8 \\ 24 & 9 & -2 \end{pmatrix}$$

é uma matriz 2 por 3 (dimensão 2×3) pelo fato de ter duas linhas e três colunas. Um dado elemento de uma matriz A de dimensão $m \times n$ é tipicamente denotado a_{ij}, referindo-se a um elemento localizado na linha i e coluna j da matriz, sendo $1 \leq i \leq m$ e $1 \leq j \leq n$.

Além da própria Matemática, matrizes podem ser encontradas em diversas áreas do mundo científico, tais como Física, Computação, Economia e Estatística. Tanto na teoria quanto na prática, matrizes são utilizadas principalmente como estruturas para a representação de fenômenos, organização de dados, relações entre variáveis, equações em sistemas lineares, etc. Em várias linguagens de programação, matrizes são representadas como *arranjos multidimensionais* em que cada linha e cada coluna é identificada por um número inteiro chamado *índice*, em umas iniciando em 0 e em outras em 1.

2 Multiplicação de matrizes

Diversas operações podem ser realizadas sobre matrizes a fim de modificar os valores de seus elementos ou mesmo as suas dimensões, tais como adição, subtração, transposição, etc. Dentre essas operações, a *multiplicação de matrizes* destaca-se pelas suas diversas aplicações teóricas e práticas na Ciência pelo fato de diversos problemas poderem ser representados e solucionados através de uma multiplicação entre matrizes. O assunto é de tamanha importância que, até o momento, encontrar o algoritmo mais rápido para realizar a multiplicação de matrizes é um problema em aberto na Ciência da Computação[1]. Diversas pesquisas têm sido desenvolvidas desde a década de 1960 no intuito de propor algoritmos que consigam computar a multiplicação de matrizes da forma mais eficiente possível, principalmente para matrizes de grandes dimensões. Com os avanços nos últimos anos em *hardware* e programação paralela e concorrente, passou-se a também investigar formas de multiplicar matrizes que fossem eficientes tanto com relação a de tempo de execução quanto a consumo de recursos computacionais.

A operação de multiplicação entre duas matrizes pode ser enunciada da seguinte forma:

> O produto de uma matriz A com dimensão $m \times p$ por outra matriz B com dimensão $p \times n$ é uma matriz C com dimensão $m \times n$ (sendo $m, n, p \in \mathbb{N}^+$) em que cada elemento c_{ij} é obtido pela soma dos produtos dos elementos correspondentes da i-ésima linha da matriz A pelos elementos da j-ésima coluna da matriz B:
>
> $$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$
>
> sendo $1 \leq i \leq m$, $1 \leq j \leq n$ e $i, j, k \in \mathbb{N}^+$.

Uma restrição importante a ser observada diz respeito ao fato que as duas matrizes A e B precisam ser necessariamente *compatíveis* entre si, isto é, o número de colunas da matriz A deve ser igual ao número de linhas da matriz B.

O procedimento tradicional para realizar a multiplicação entre duas matrizes compatíveis A e B fornecidas como entrada envolve a execução de três laços aninhados, cujas instruções são executadas de forma sequencial. O Algoritmo 1 esquematiza na forma de pseudocódigo esse procedimento, que resulta na matriz produto C.

Para exemplificar, sejam matrizes A e B, ambas de dimensão 2×2, como seguem:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} -1 & 3 \\ 4 & 2 \end{pmatrix}$$

[1] Outros problemas ainda sem solução na Ciência da Computação são listados na Wikipedia em https://en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science

Algoritmo 1 Algoritmo sequencial para multiplicação de matrizes

```
1: variável A : Matriz(m, p)                          ▷ A possui dimensão m × p
2: variável B : Matriz(p, n)                          ▷ B possui dimensão p × n
3: variável C : Matriz(m, n)                          ▷ C possui dimensão m × n
4: procedimento MultiplicaMatrizes(A, B, C : Matriz, m, n, p : Inteiro)
5:     variável i : Inteiro
6:     variável j : Inteiro
7:     variável k : Inteiro
8:     para i ← 1 até m faça
9:         para j ← 1 até n faça
10:            variável soma : Inteiro
11:            soma ← 0
12:            para k ← 1 até p faça
13:                soma ← soma + A[i, k] ∗ B[k, j]
14:            fim para
15:            C[i, j] ← soma
16:        fim para
17:    fim para
18: fim procedimento
```

Seguindo o Algoritmo 1, a computação da matriz produto C, também de dimensão 2×2, é feita da seguinte forma:

$$C = A.B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} -1 & 3 \\ 4 & 2 \end{pmatrix} = \begin{pmatrix} 1.(-1) + 2.4 & 1.3 + 2.2 \\ 3.(-1) + 4.4 & 3.3 + 4.2 \end{pmatrix} = \begin{pmatrix} 7 & 7 \\ 13 & 17 \end{pmatrix}$$

3 Tarefas

As tarefas a serem realizadas neste trabalho basicamente consistem em (i) implementar uma solução sequencial e uma concorrente para o algoritmo de multiplicação de matrizes e (ii) realizar sucessivas execuções das soluções a fim de fazer uma análise comparativa entre elas e (iii) elaborar um relatório descrevendo as atividades que foram realizadas, os resultados observados e as conclusões obtidas. As soluções poderão ser implementadas utilizando facilidades providas pelas linguagens de programação C/C++, Java **ou** Python. O desenvolvimento da solução deve de antemão visar pela busca de desenvolvimento de *software* de qualidade, isto é, funcionando correta e eficientemente, exaustivamente testado, bem documentado e com tratamento adequado de eventuais exceções.

Especificamente com relação à solução concorrente para a multiplicação entre duas matrizes compatíveis, esta deverá envolver a utilização de *threads* em que a sub-tarefa mínima para uma *thread* é o cálculo de uma linha da matriz produto. Além disso, deverão ser utilizadas *threads* **apenas** na parte da multiplicação das matrizes propriamente ditas, ou seja, a entrada e saída de dados devem ser sequenciais.

3.1 Entrada de dados e execução do programa

As operações de multiplicação serão feitas sobre duas matrizes quadradas, cada uma representada por um arquivo de texto que deverá ser lido pelo programa. Os arquivos são nomeados pelo nome da matriz seguido da sua dimensão e são estruturados da seguinte forma: (i) a primeira linha contém dois números inteiros separados por um espaço simples, representando as dimensões da matriz, e (ii) as linhas subsequentes contêm os valores dos elementos de cada linha da matriz, também separados por um espaço simples. A operação de multiplicação deverá ser feita entre as matrizes cujos nomes de arquivo iniciam com o o caractere `A` pelas matrizes cujos nomes de arquivo iniciam com o caractere `B`. Um exemplo de arquivo de entrada teria o nome `A2x2.txt` e o seguinte contéudo:

```
2 2
1 2
3 4
```

Neste trabalho, será utilizado como entrada um conjunto de vinte arquivos representando matrizes quadradas de diferentes dimensões, sendo dez para as matrizes `A` e dez para as matrizes `B`. Os arquivos de entrada estão disponíveis para *download* através da Turma Virtual do SIGAA.

O programa principal deverá ser executado **via linha de comando** da seguinte forma:

```
$ <programa> 2 S
```

em que o número inteiro seguido do nome do programa representa a dimensão das matrizes quadradas que serão tratadas pelo programa e os caracteres `S` e `C` indicam respectivamente que será utilizada a solução sequencial ou a solução concorrente. Todas as matrizes utilizadas como casos de teste para este trabalho possuem dimensões como potências de base 2, logo qualquer valor fornecido como argumento de linha de comando ao programa deve atender a essa restrição. De forma similar, os únicos caracteres válidos para especificar que tipo de solução será utilizada são `S` e `C`.

Para cada uma das dimensões de matrizes i, o programa deverá: (i) ler os arquivos de entrada `A`i`x`i`.txt` e `B`i`x`i`.txt`; (ii) realizar a multiplicação dessas matrizes, seja de forma sequencial ou concorrente; (iii) medir o tempo despendido para tal execução, e; (iv) gravar a matriz produto resultante da multiplicação em um arquivo `C`i`x`i`.txt`, que será uma das saídas a serem geradas pelo programa.

3.2 Experimentação

No intuito de tornar os experimentos representativos do ponto de vista estatístico, será necessário executar cada uma soluções em um total de vinte vezes. Os tempos de execução de cada soluções em cada uma dessas vinte execuções deverá ser registrado a fim de calcular os valores *máximo*, *mínimo* e *médio* dos tempos nas vinte execuções, além do *desvio padrão* observado. Para calcular o desvio padrão σ dos tempos de execução dessas soluções, você poderá utilizar a seguinte equação:

$$\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(t_i - t_M)^2}$$

em que t_i é o tempo registrado na i-ésima execução da solução em questão e t_M é o tempo médio das vinte execuções. Um baixo desvio padrão indica que os pontos dos dados tendem a estar próximos da média desses dados, enquanto que um alto desvio padrão indica que os pontos dos dados estão espalhados por uma ampla gama de valores.

Especificamente com relação à análise comparativa entre as versões sequencial e concorrente, é de relevante interesse determinar o ganho de desempenho (*speed-up*) eventualmente obtido com a versão concorrente. Esse cálculo pode ser realizado através da seguinte equação:

$$speedup = \frac{T_{sequencial}}{T_{concorrente}}$$

em que $T_{sequencial}$ é o tempo despendido para a multiplicação de matrizes pela solução sequencial e $T_{concorrente}$ o tempo despendido pela solução concorrente. Caso o valor do *speedup* seja superior a 1, tem-se que a solução concorrente é de fato melhor em termos de desempenho do que a sequencial.

3.3 Relato

Uma vez realizada a tarefa de implementação e experimentação, você deverá elaborar um relatório técnico simples contendo, no mínimo, as seguintes seções:

Introdução Explicar o propósito do relatório.

Metodologia Indicar o método adotado para realizar os experimentos com as soluções e analisar os resultados obtidos. Por exemplo, deverá ser apresentada a caracterização técnica do computador utilizado (processador, sistema operacional, quantidade de memória RAM), a linguagem de programação e a versão do compilador empregados, os cenários considerados, entre outras informações. Deverão também ser descritos qual o procedimento adotado para gerar os resultados, como a comparação entre as soluções foi feita, etc.

Resultados Apresentar os resultados obtidos na forma de um gráficos de linha **e** tabelas. Para cada solução deverá ser apresentada uma tabela contendo os tempos mínimo, médio e máximo obtidos para cada dimensão de matriz, além dos valores de desvio padrão. Por sua vez, os gráficos de linha deverão exibir apenas os tempos médios obtidos nas execuções de cada solução.

Conclusões Discutir os resultados, ou seja, o que foi possível concluir através dos resultados obtidos através dos experimentos, incluindo uma análise do ganho de desempenho (*speed-up*).

4 Autoria e política de colaboração

O trabalho poderá ser feito **individualmente ou em equipe composta por no máximo dois estudantes**, sendo que, neste último caso, é importante, dentro do possível, dividir as tarefas igualmente entre os integrantes da equipe. O trabalho em cooperação entre estudantes da turma é estimulado, sendo aceitável a discussão de ideias e estratégias. Contudo, tal interação não deve ser entendida como permissão para utilização de (parte de) código fonte de outras equipes, o que pode caracterizar

situação de plágio. Trabalhos copiados em todo ou em parte de outras equipes ou da Internet serão sumariamente rejeitados e receberão nota zero.

5 Entrega

Todos os códigos fonte resultantes da implementação deste trabalho, sem erros de compilação e devidamente testados e documentados, deverão ser submetidos como um único arquivo compactado no formato `.zip` **até as 23h59 do dia 19 de dezembro de 2021** através da opção *Tarefas* na Turma Virtual do SIGAA. Juntamente com os códigos fonte, o arquivo compactado deverá também conter o relatório escrito, preferencialmente em formato *Adobe Portable Format* (PDF). Se for o caso, é possível fornecer, no campo *Comentários* do formulário eletrônico de submissão da tarefa, o endereço de um repositório remoto destinado ao controle de versões, porém esta opção não exclui a necessidade de submissão dos arquivos via SIGAA. **Não é necessário anexar ao envio os arquivos utilizados como entrada nem os arquivos de saída resultantes.**

6 Avaliação

A avaliação deste trabalho será feita principalmente sobre os seguintes critérios: (i) utilização correta dos conceitos de *threads* vistos na disciplina; (ii) a corretude da execução das soluções implementadas, tanto com relação a funcionalidades quanto a concorrência; (iii) a aplicação de boas práticas de programação, incluindo legibilidade, organização e documentação de código fonte, e; (iv) qualidade do relatório produzido. O trabalho possuirá nota máxima de 10,0 (dez) pontos.