

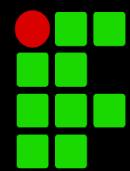
Instituto Federal de Educação, Ciência e Tecnologia Baiano

Campus Guanambi

Análise de desempenho de algoritmos de multiplicação de matrizes considerando versões sequencial, concorrente e distribuída

Adauto Benevides Couto

Projeto de Conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (ADS)



INSTITUTO FEDERAL
Baiano
Campus Guanambi

TRABALHO DE CONCLUSÃO DE CURSO

Woquiton Lima Fernandes

Data de Entrega: _____

Assinatura do Aluno: _____

BANCA EXAMINADORA

Msc. George Gabriel Mendes Dourado
(Orientador)

Instituto Federal Baiano - Campus Guanambi

Dr. Woquiton Lima Fernandes (Membro)
Instituto Federal Baiano - Campus Guanambi

Dr. Naidson Clayr Santos Ferreira (Membro)
Instituto Federal Baiano - Campus Guanambi

Adauto Benevides Couto

Análise de desempenho de algoritmos de multiplicação de matrizes considerando versões sequencial, concorrente e distribuída

Projeto de Conclusão de Curso apresentado ao , ligado ao Ministério da Educação como parte dos requisitos para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Área de Concentração: Análise e Desenvolvimento de Sistemas

Orientador: Prof. MSc. George Gabriel Mendes Dourado

**Guanambi - BA
Julho de 2023**

Adauto Benevides Couto

Performance analysis of matrix multiplication algorithms
considering sequential, concurrent, and distributed versions.

Completion of course work submitted to the , Ministry
of Education as part of the requirements for obtaining
the title of Technologist in Systems Analysis and
Development.

Concentration Area: Systems Analysis and
Development

Advisor: Prof. MSc. George Gabriel Mendes Dourado

Guanambi - BA
July 2023

AGRADECIMENTOS

Primeiramente, quero expressar minha gratidão a Deus, pois sem Ele, eu não teria alcançado este ponto. Em cada momento de dificuldade ao longo deste processo, encontrei forças Nele. Agradeço a Deus por ser meu refúgio, fortaleza e fonte de confiança, assim como Davi descreve no Salmo 91.

Agradeço ao meu orientador, Msc. George Gabriel Mendes Dourado, que me guiou durante todo o processo de escrita e desenvolvimento deste trabalho. Sua orientação foi fundamental, e suas sugestões e refinamentos contribuíram significativamente para que meu trabalho alcançasse a melhor forma possível.

Agradeço também aos membros da banca do pré-projeto, Dr. Naidson Clayr e Msc. João Paulo Barbosa, por suas considerações e conselhos. Suas observações foram essenciais para o aprimoramento e conclusão deste trabalho.

Um agradecimento especial à minha família, cujo apoio foi de suma importância para que eu chegassem até aqui. Mesmo nos momentos mais difíceis, vocês estiveram ao meu lado, oferecendo força e encorajamento. Em particular, quero homenagear meu pai, que já não está mais conosco, mas que sempre me ensinou a nunca desistir. Ele foi uma grande influência para me tornar quem sou hoje, e é impossível não expressar minha gratidão a ele neste trabalho.

Finalmente, sou grato aos meus amigos e colegas que, ao longo deste processo, me apoiaram e me incentivaram a continuar, mesmo quando as coisas não estavam fáceis. Cada um de vocês teve um papel especial durante os desafios e dificuldades do curso e deste projeto.

RESUMO

COUTO, A. B. . **Análise de desempenho de algoritmos de multiplicação de matrizes considerando versões sequencial, concorrente e distribuída** . 2023. 76 p. Projeto de Conclusão de Curso (Projeto de Conclusão em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia Baiano, Guanambi – BA, 2023.

A multiplicação de matrizes é uma operação de grande importância em diversas áreas da sociedade, como por exemplo, na matemática, física, engenharia, processamento de linguagem natural, entre outros. A computação tem auxiliado melhorando o desempenho de realização destes cálculos, reduzindo o tempo total para se obter uma resposta, entretanto, a depender da abordagem computacional utilizada no algoritmo de multiplicação de matrizes (sequencial, concorrente ou distribuída), este tempo de resposta pode variar. Desta forma, o objetivo deste trabalho foi o de implementar as três abordagens e posteriormente realizar uma análise de desempenho utilizando como métrica o tempo de resposta, buscando identificar qual algoritmo apresenta melhores resultados em cenários distintos de tamanhos de matrizes (pequenas, médias e grandes). Os dados foram coletados executando os algoritmos no Laboratório de Informática I, do Instituto Federal de Educação, Ciência e Tecnologia Baiano Campus Guanambi, onde foram separadas cinco máquinas com especificações semelhantes para as execuções. Os resultados demonstraram que, os cenários distintos de tamanhos de matriz influenciam no desempenho dos algoritmos, apontando que para cenários onde as matrizes são menores, o algoritmo sequencial apresenta melhor desempenho, enquanto para matrizes médias o algoritmo concorrente obtém um desempenho mais elevado, e considerando matrizes de tamanhos grandes, o algoritmo distribuído possui os melhores resultados. Esse estudo buscou contribuir para uma melhor compreensão das vantagens e desvantagens de cada abordagem, oferecendo informações para a escolha do algoritmo adequado para cada cenário.

Palavras-chave: Multiplicação de matrizes, Algoritmo sequencial, Algoritmo concorrente, Algoritmo distribuído, Análise de desempenho.

ABSTRACT

COUTO, A. B. . **Performance analysis of matrix multiplication algorithms considering sequential, concurrent, and distributed versions.**. 2023. 76 p. Projeto de Conclusão de Curso (Projeto de Conclusão em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia Baiano, Guanambi – BA, 2023.

Matrix multiplication is an operation of great importance in various areas of society, such as mathematics, physics, engineering, natural language processing, among others. Computing has helped by improving the performance of these calculations, reducing the total time to obtain an answer. However, depending on the computational approach used in the matrix multiplication algorithm (sequential, concurrent or distributed), this response time can vary. Therefore, the aim of this work was to implement the three approaches and then carry out a performance analysis using response time as a metric, seeking to identify which algorithm performs best in different matrix size scenarios (small, medium and large). The data was collected by running the algorithms in the Computer Laboratory I of the Federal Institute of Education, Science and Technology of Bahia, Guanambi Campus, where five machines with similar specifications were separated for the runs. The results showed that the different matrix size scenarios influence the performance of the algorithms, pointing out that for scenarios where the matrices are smaller, the sequential algorithm performs better, while for medium matrices the concurrent algorithm achieves higher performance, and considering large matrices, the distributed algorithm has the best results. This study sought to contribute to a better understanding of the advantages and disadvantages of each approach, providing information for choosing the appropriate algorithm for each scenario.

Keywords: Matrix multiplication, Sequential algorithm, Concurrent algorithm, Distributed algorithm, Performance analysis.

LISTA DE ILUSTRAÇÕES

Figura 1 – Nomenclatura de uma matriz	19
Figura 2 – Exemplo de matrizes retangular (a) e quadrada (b)	20
Figura 3 – Exemplo de matriz linha (a), matriz coluna (b) e matriz nula (c)	20
Figura 4 – Exemplo de diagonal principal e secundária (a), matriz diagonal (b) e matriz identidade (c)	21
Figura 5 – Imagem do Laboratório 1 de informática	33
Figura 6 – Exemplo do funcionamento da função para uma matriz de tamanho 2x2, onde na primeira parte é mostrado o processo padrão da multiplicação, enquanto na segunda é mostrado o processo dos elementos no algoritmo, sendo os elementos da matriz “a” representados pela cor azul, os elementos da matriz “b” pela cor amarela, e os elementos da matriz “c” pela cor vermelha	42
Figura 7 – Fluxograma do código sequencial que mostra os passos realizados pelo algoritmo para concluir a tarefa da multiplicação	43
Figura 8 – Diagrama representando o fluxo de procedimentos realizados para o cálculo de uma matriz de ordem 32x32 no algoritmo concorrente	45
Figura 9 – Demonstração do comando de execução do nó trabalhador, onde o comando java é utilizado seguido do nome da classe e dos argumentos: o primeiro é o número IP da máquina mestre, e o segundo é a porta em que o nó mestre está sendo executado.	51
Figura 10 – Fluxograma que mostra as etapas e procedimentos realizados pelo nó mestre e nós trabalhadores no algoritmo distribuído	53
Figura 11 – Fluxograma apresentando um exemplo de execução do algoritmo distribuído considerando uma matriz de ordem 64x64	54
Figura 12 – Gráfico 1: Gráfico em barras dos resultados das execuções dos algoritmos .	55
Figura 13 – Gráfico 2: Gráfico em linhas dos resultados das execuções dos algoritmos .	56
Figura 14 – Gráfico 3: gráfico em barras dos resultados das execuções de matrizes pequenas	57
Figura 15 – Gráfico 4: Gráfico em linhas dos resultados das execuções de matrizes pequenas	57
Figura 16 – Gráfico 5: Gráfico em barras dos resultados das execuções de matrizes médias	59
Figura 17 – Gráfico 6: Gráfico em linhas dos resultados das execuções de matrizes média	59
Figura 18 – Gráfico 7: Gráfico em barras dos resultados das execuções de matrizes grandes	60
Figura 19 – Gráfico 8: Gráfico em linhas dos resultados das execuções de matrizes grandes	61
Figura 20 – Gráfico 9: Gráfico em barras da comparação de desempenho para cada ordem, realizadas nas máquinas do laboratório	62

Figura 21 – Gráfico 10: Gráfico em barras da comparação de desempenho para cada ordem, realizadas no computador pessoal 62

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – loop aninhado	40
Código-fonte 2 – Função de multiplicação da matriz	41
Código-fonte 3 – lógica da criação de threads	44
Código-fonte 4 – Espera das threads por meio do método join	45
Código-fonte 5 – Inicialização de variáveis	47
Código-fonte 6 – Criação de conexão	47
Código-fonte 7 – Criação da classe de conexão	48
Código-fonte 8 – Divisão de tarefa para cada nó trabalhador	48
Código-fonte 9 – Recebe e monta a matriz resultante	49
Código-fonte 10 – Obtenção dos parâmetros por meio de argumentos	50
Código-fonte 11 – Recebimento das matrizes para o cálculo	50
Código-fonte 12 – Lógica da criação das threads e cálculo da multiplicação das matrizes	51
Código-fonte 13 – Comando para execução do algoritmo sequencial	75
Código-fonte 14 – Comando para execução do algoritmo sequencial	75
Código-fonte 15 – Comando para execução do algoritmo concorrente	76
Código-fonte 16 – Comando para execução do algoritmo distribuído: Nô Mestre	76
Código-fonte 17 – Comando para execução do algoritmo distribuído: Nós Trabalhadores	76

LISTA DE TABELAS

Tabela 1 – Especificações da máquina GBI0227405CMP do laboratório	34
Tabela 2 – Especificações da máquina GBI0227410CMP do laboratório	34
Tabela 3 – Especificações da máquina GBI0227402CMP do laboratório	35
Tabela 4 – Especificações da máquina GBI0227401CMP do laboratório	35
Tabela 5 – Especificações da máquina GBI0227407CMP do laboratório	35
Tabela 6 – Especificações do computador pessoal LAPTOP-HAKTI0UT	36
Tabela 7 – Quantidade total de <i>threads</i> usadas no algoritmo concorrente para cada ordem de matriz. O cálculo foi dado pela quantidade de <i>threads</i> criadas no algoritmo mais a <i>thread main</i>	46
Tabela 8 – Informações de configuração para execuções distribuídas. A tabela mostra a quantidade total de <i>threads</i> , calculadas considerando as <i>threads main</i> dos nós trabalhadores, mais a quantidade de <i>threads</i> criadas para a execução, é apresentado também a quantidade de nós trabalhadores usados para cada tamanho da matriz.	53
Tabela 9 – Tempos de execução no algoritmo sequencial realizados na máquina 227401-A	68
Tabela 10 – Tempos de execução no algoritmo sequencial realizados na máquina 227402-A	69
Tabela 11 – Tempos de execução no algoritmo sequencial realizados na máquina 227405-A	69
Tabela 12 – Tempos de execução no algoritmo sequencial realizados na máquina 227407-A	69
Tabela 13 – Tempos de execução no algoritmo sequencial realizados na máquina 227410-A	70
Tabela 14 – Média feita entre os tempos de execução do algoritmo sequencial nas cinco máquinas	70
Tabela 15 – Tempos de execução no algoritmo sequencial realizados no computador pessoal	70
Tabela 16 – Tempos de execução no algoritmo concorrente realizados na máquina 227401-A	71
Tabela 17 – Tempos de execução no algoritmo concorrente realizados na máquina 227402-A	71
Tabela 18 – Tempos de execução no algoritmo concorrente realizados na máquina 227405-A	72
Tabela 19 – Tempos de execução no algoritmo concorrente realizados na máquina 227407-A	72
Tabela 20 – Tempos de execução no algoritmo concorrente realizados na máquina 227410-A	72
Tabela 21 – Média feita entre os tempos de execução do algoritmo concorrente nas cinco máquinas	73
Tabela 22 – Tempos de execução no algoritmo concorrente realizados no computador pessoal	73
Tabela 23 – Tempos de execução do algoritmo distribuído realizado nas cinco máquinas do laboratório	74

LISTA DE ABREVIATURAS E SIGLAS

CAT 5e	<i>Category 5 enhanced</i>
CAT 6	<i>Category 6</i>
CPU	<i>Central Processing Unit</i>
IDE	<i>Integrated Development Environment</i>
IP	<i>Internet Protocol</i>
IPC	<i>Inter-Process Communication</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>

SUMÁRIO

1	INTRODUÇÃO	15
2	OBJETIVOS	17
2.1	Objetivo Geral	17
2.2	Objetivos Específicos	17
3	JUSTIFICATIVA	18
4	REVISÃO BIBLIOGRÁFICA/TEÓRICA	19
4.1	Matrizes	19
4.1.1	<i>O que são Matrizes ?</i>	19
4.1.2	<i>Aplicação das matrizes</i>	21
4.1.3	<i>Multiplicação de matrizes</i>	22
4.1.4	<i>Tamanhos de matrizes</i>	23
4.2	Algoritmos	23
4.2.1	<i>O que são algoritmos?</i>	23
4.2.2	<i>Forma sequencial</i>	23
4.2.3	<i>Forma concorrente</i>	23
4.2.4	<i>Forma distribuída</i>	24
4.2.5	<i>Algoritmos multiplicadores de matrizes</i>	25
4.3	Programação sequencial, concorrente e distribuída em Java	26
4.3.1	<i>Primitivas utilizadas:</i>	26
5	METODOLOGIA	32
5.1	Etapa 1: Definição do escopo do projeto	32
5.2	Etapa 2: Definição do local e ambiente de desenvolvimento	33
5.2.1	<i>Especificações de Hardware:</i>	33
5.2.2	<i>Ambiente de desenvolvimento:</i>	36
5.3	Etapa 3: Estudo das matrizes e algoritmos	36
5.4	Etapa 4: Codificação dos algoritmos	37
5.4.1	<i>Codificação do algoritmo concorrente</i>	37
5.4.2	<i>Codificação do algoritmo sequencial</i>	37
5.4.3	<i>Codificação do algoritmo distribuído</i>	37
5.5	Etapa 5: Definição dos parâmetros para os testes	38

5.6	Etapa 6: Geração dos gráficos e análise dos dados	39
6	ALGORITMOS DESENVOLVIDOS	40
6.1	Explicação dos algoritmos	40
6.1.1	<i>Algoritmo Sequencial</i>	40
6.1.2	<i>Algoritmo Concorrente</i>	43
6.1.3	<i>Algoritmo Distribuído</i>	46
7	RESULTADOS E DISCUSSÃO	55
7.1	Análise dos resultados	55
7.1.1	<i>Comparação geral</i>	55
7.1.2	<i>Comparação entre os resultados das matrizes pequenas</i>	57
7.1.3	<i>Comparação entre os resultados das matrizes médias</i>	59
7.1.4	<i>Comparação entre os resultados das matrizes grandes</i>	60
7.1.5	<i>Comparação entre os resultados do laboratório e testes realizados no computador pessoal</i>	62
8	CONCLUSÃO	64
8.1	Trabalhos Futuros	65
REFERÊNCIAS		66
A	TABELAS DE TEMPOS DE EXECUÇÃO DOS ALGORITMOS . .	68
A.1	Resultados Sequenciais	68
A.1.1	<i>Máquina : 227401-A</i>	68
A.1.2	<i>Máquina : 227402-A</i>	69
A.1.3	<i>Máquina : 227405-A</i>	69
A.1.4	<i>Máquina : 227407-A</i>	69
A.1.5	<i>Máquina : 227410-A</i>	70
A.1.6	<i>Média dos tempos de execução do algoritmo sequencial nas cinco máquinas</i>	70
A.1.7	<i>Computador Pessoal</i>	70
A.2	Resultados Concorrentes	71
A.2.1	<i>Máquina : 227401-A</i>	71
A.2.2	<i>Máquina : 227402-A</i>	71
A.2.3	<i>Máquina : 227405-A</i>	72
A.2.4	<i>Máquina : 227407-A</i>	72
A.2.5	<i>Máquina : 227410-A</i>	72
A.2.6	<i>Média dos tempos de execução do algoritmo concorrente nas cinco máquinas</i>	73

A.2.7	<i>Computador Pessoal</i>	73
A.3	Resultados Distribuídos	73
A.3.1	<i>Resultados no laboratório</i>	74
A.3.2	<i>Resultados no Computador Pessoal</i>	74
B	COMANDOS PARA EXECUÇÃO DOS ALGORITMOS	75
B.1	Comando de compilação Java	75
B.2	Execução do algoritmo sequencial	75
B.3	Execução do algoritmo Concorrente	76
B.4	Execução do algoritmo Distribuído	76



INTRODUÇÃO

As matrizes são estruturas de elementos organizados em linhas e colunas usadas para representar e manipular dados, possuindo extrema importância em áreas como matemática, estatística e diversos outros campos de estudo, inclusive na computação, podendo ser aplicadas em setores como aprendizado de máquina e processamento de imagem (GONZALEZ; WOODS, 2000; GOODFELLOW; BENGIO; COURVILLE, 2016).

A multiplicação de matrizes, objeto de estudo mais aprofundado, possui diversas soluções para o cálculo. Diversos estudiosos propuseram formas de multiplicá-las, desde soluções mais simples para matrizes menores até soluções mais complexas para reduzir o esforço computacional em matrizes muito grandes. Gene H. Golub e Charles F. Van Loan são matemáticos renomados e se tornaram referência estudando as matrizes e a álgebra linear, seu livro *Matrix Computations* discute sobre as matrizes e formas de multiplicá-las (GOLUB; LOAN, 1996).

Considerando essas estruturas é essencial abordar a influência dos algoritmos. Programadores desenvolveram códigos para realizar essas operações e otimizar o desempenho de aplicações que as utilizam, como aprendizado de máquina e processamento de linguagem natural, que incorporam de forma significativa essa operação (GOODFELLOW; BENGIO; COURVILLE, 2016).

Esses algoritmos podem assumir as formas sequencial, concorrente e distribuída, trazendo diferentes abordagens de programação para se adaptarem a cenários distintos, como o exemplo da programação distribuída, que opera com vários computadores para resolver um problema complexo e proporcionar um melhor desempenho (no contexto do tempo de resposta do trabalho) para aplicações em larga escala. Enquanto isso, aplicações mais simples podem se beneficiar da abordagem sequencial, buscando uma maior velocidade na execução.

Nesse contexto, o trabalho propõe analisar o desempenho de um conjunto de dados de entrada com matrizes de diferentes tamanhos, considerando o tempo de resposta como fator de avaliação. Foram exploradas abordagens sequenciais, com códigos básicos e instruções

simples, computação concorrente, na qual tarefas são realizadas simultaneamente e computação distribuída, com tarefas divididas entre diferentes computadores trabalhando em conjunto, apresentando o algoritmo mais adequado para cada situação estudada.

A sequência deste trabalho está organizada da seguinte forma: O capítulo dois apresenta os objetivos geral e específicos, que descrevem os objetivos a serem alcançados durante o desenvolvimento. No capítulo três, apresenta-se a justificativa, que explica o motivo da realização do trabalho. O capítulo quatro descreve a revisão bibliográfica, onde está o embasamento teórico necessário para a construção do trabalho. O capítulo cinco define quais materiais foram utilizados e como o trabalho foi desenvolvido. O capítulo seis discorre sobre os algoritmos desenvolvidos durante a etapa de desenvolvimento. O capítulo sete apresenta uma análise detalhada dos dados coletados e discute os resultados. No capítulo oito é abordada a conclusão do trabalho. É possível também acessar no apêndice A tabelas com os dados brutos coletados, detalhando os tempos de cada uma das execuções dos algoritmos.



OBJETIVOS

2.1 Objetivo Geral

Implementar e avaliar o desempenho de algoritmos multiplicadores de matrizes em suas versões sequencial, concorrente e distribuída, visando identificar quais os melhores resultados em cenários distintos de tamanho de matrizes.

2.2 Objetivos Específicos

- Implementar os algoritmos multiplicadores de matrizes em suas diferentes versões, utilizando a linguagem de programação Java;
- Avaliar o desempenho das diferentes versões dos algoritmos de multiplicação de matrizes, considerando tamanhos variados de matrizes e utilizando o tempo de resposta como métrica;
- Apresentar os resultados da avaliação de desempenho dos algoritmos, em forma de tabelas e gráficos, a fim de facilitar a observação dos dados;
- Comparar os resultados alcançados pelas diferentes formas de multiplicação de matrizes;
- Recomendar a melhor versão do algoritmo para cada situação estudada, com base nos resultados obtidos.



JUSTIFICATIVA

A multiplicação de matrizes desempenha um papel de grande relevância, sendo encontrada em aplicações em diversos âmbitos, como economia, engenharia, física e também em campos modernos como processamento de linguagem natural e processamento de imagens (STÜLP; FOCHEZATTO, 2009; GONZALEZ; WOODS, 2000; TARANTINO; MONICA; BERGENTI, 2018). A análise de desempenho é uma técnica fundamental na área da ciência da computação que está ligada à medição e avaliação de desempenho de algoritmos, sistemas e programas de computador. No contexto da multiplicação de matrizes é um tópico relevante e pode ser avaliado em termos de tempo de execução, consumo de memória e eficiência.

Essa variedade de grandezas, levam a estudar diferentes algoritmos com tamanhos distintos, fazendo com que em um determinado ambiente seja possível analisar qual algoritmo e forma (sequencial, concorrente e distribuída) traz melhor resultado, para situações específicas.

Com relação a aplicações em grande escala, ou até mesmo áreas que em partes usem esse recurso, é possível que o uso de um algoritmo menos eficiente possa trazer gastos, de tempo, energia, ou recursos computacionais como processamento de memória, e até mesmo levar a resultados que não trazem grande precisão, o que seria um problema para quem o utiliza.

Sendo assim, fazer um estudo aprofundado verificando as formas sequencial, concorrente e distribuída e mostrar dados relevantes sobre algoritmos de multiplicação de matrizes, pode resultar na contribuição para que, conforme situações usadas no trabalho, se obtenha um melhor resultado em sua aplicação considerando seu tempo de resposta.



REVISÃO BIBLIOGRÁFICA/TEÓRICA

4.1 Matrizes

Este capítulo apresentará as matrizes, trazendo definição, exemplos e aplicações, mostrando também como as matrizes podem ser multiplicadas.

4.1.1 *O que são Matrizes ?*

Uma matriz pode ser definida como um conjunto de entradas, que possui em sua formação elementos que podem ser números, símbolos ou expressões. Estes elementos fazem parte de um corpo, que está alinhado entre filas horizontais que tem o nome de linhas e filas verticais que se chamam colunas (STRANG, 2016).

As matrizes possuem uma regra a ser seguida em relação ao nome que elas recebem: São representadas por letra maiúsculas e os seus elementos pela mesma letra em minúsculo e dois índices, o primeiro indicando a linha daquele elemento e o segundo indicando a coluna, como indicado na Figura 1 (MACHADO, 2005).

Figura 1 – Nomenclatura de uma matriz

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Fonte: (STRANG, 2016)

Conceitos importantes:

Quando o número de elementos da linha de uma matriz é diferente do número de elementos de sua coluna ela é classificada como uma matriz retangular, conforme mostrado na Figura 2 (a).

Já quando o número de elementos são iguais é denominada como uma matriz quadrada, assim como na Figura 2 (b).

Figura 2 – Exemplo de matrizes retangular (a) e quadrada (b)

$$A = \begin{pmatrix} 1 & 2 & 2 & 4 & 6 \\ 1 & 2 & 3 & 6 & 9 \\ 0 & 0 & 1 & 2 & 3 \end{pmatrix}$$

(a)

$$B = \begin{pmatrix} 2 & 4 & 2 \\ 0 & 4 & 4 \\ 0 & 8 & 8 \end{pmatrix}$$

(b)

Fonte: (STRANG, 2016)

Os termos a seguir podem ser encontrados no livro (MACHADO, 2005)

Matriz Linha: Ocorre quando existe uma matriz que possui uma única linha podendo haver várias colunas como diz a Figura 3 (a).

Matriz Coluna: Acontece quando ocorre o inverso da matriz linha, ou seja existe apenas uma coluna, e pode haver diversas linhas assim como mostra a Figura 3 (b).

Matriz nula: é uma matriz onde todos os elementos são iguais a zero como na Figura 3 (c).

Figura 3 – Exemplo de matriz linha (a), matriz coluna (b) e matriz nula (c)

$$y = (y_1 \quad y_2 \quad \cdots \quad y_n)$$

(a)

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

(b)

$$z = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

(c)

Fonte: (MACHADO, 2005), (LIPSCHUTZ; ABELLANAS; ONTALBA, 1992)

Quando se trata de uma matriz quadrada existem alguns outros conceitos tais como:

Diagonal principal ou diagonal de uma matriz: se refere a sequência de elementos que está disposto na mesma linha e coluna, ao exemplo da matriz da Figura 4 (a) a sequência de elementos (1, 0, 1).

Diagonal secundária de uma matriz: é a sequência de elementos que está disposto em mesma linha e coluna, mas com linhas e colunas invertidas, ao exemplo da matriz da Figura 4 (a) a sequência de elementos (0, 0, 2).

Matriz diagonal: É referente a situação em que todos os elementos fora da diagonal principal da matriz principal são iguais a 0 como é mostrado na Figura 4 (b).

Matriz identidade: Quando os elementos da diagonal são todos iguais a 1.Uma matriz identidade de ordem n é representada por In ou por I, conforme a Figura 4 (c).

Figura 4 – Exemplo de diagonal principal e secundária (a), matriz diagonal (b) e matriz identidade (c)

$$\begin{array}{l} A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 1 & 1 \end{pmatrix} \quad (a) \\ A = \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix} \quad (b) \\ A = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_n \end{pmatrix} \quad (c) \end{array}$$

Fonte: (MACHADO, 2005), (SANTOS, 2006)

4.1.2 Aplicação das matrizes

Como descrito na justificativa as matrizes são ferramentas importantes para diversas áreas e são amplamente usadas em serviços variados, a seguir será mostrado alguns exemplos de onde as matrizes podem ser aplicadas e sua usabilidade na prática.

Economia: Em seu artigo (STÜLP; FOCHEZATTO, 2009) Valter José Stülp e Adelar Focchezatto aplicam matrizes de Markov para analisar a distribuição da renda no estado do Rio Grande do Sul. Eles usam o intervalo de tempo de 1985 até 1999 para fazer uma estimativa com base em uma função de distribuição da renda *per capita*. Examinando os dados para verificar se economias menos desenvolvidas tem tendência a alcançar o nível de renda *per capita* das economias com maior desenvolvimento socioeconômico.

A matriz de Markov é usada para mostrar a evolução de um sistema ao longo do tempo. Ela

representa a possibilidade de que um estado mude com o passar do tempo e suas entradas são essas probabilidades de um estado atual convergir para um possível no futuro (ROSS, 2019).

Processamento de Linguagem Natural: As matrizes estão presentes também na área do processamento de Linguagem Natural como pode ser visto no artigo (TARANTINO; MONICA; BERGENTI, 2018) de Gianmaria Tarantino, Stefania Monica e Federico Bergenti que propõe a realização de um algoritmo de fatoração de matriz probabilística baseado em um algoritmo já existente sugerindo algumas melhorias com base nos testes realizados.

O algoritmo usa uma técnica que processa volumes muito grandes de dados em arquivos textuais fazendo um vocabulário de termos, e em seguida criando uma matriz do documento, onde as linhas são referentes aos documentos e as colunas o número de vezes que a palavra foi mencionada nele, e por fim essas matrizes são reduzidas visando encontrar dados com precisão e sem a perda de informações relevantes (TARANTINO; MONICA; BERGENTI, 2018).

4.1.3 Multiplicação de matrizes

A multiplicação de matrizes é uma operação que é recorrente e está presente de forma significativa em diversos setores da área computacional. Ela desempenha um papel importante em diferentes aplicações no ramo da ciência da computação, incluindo simulação de física, computação gráfica e aprendizado de máquina, reforçando a importância de sua utilização e exploração de algoritmos para executá-la da maneira mais eficiente possível.

Existem diversas maneiras de se multiplicar matrizes cada uma com suas vantagens e desvantagens. No livro (GOLUB; LOAN, 1996) de Gene H. Golub e Charles F. Van Loan podemos ver exemplos de algumas dessas formas:

Multiplicação de matrizes convencional: existindo duas matrizes onde o número de colunas da primeira é igual ao número de linhas da segunda, elas podem ser multiplicadas tendo como produto uma terceira matriz resultante, sendo assim para realizar essa operação considerando uma matriz A sendo multiplicada por uma matriz B sabendo que seguem a primeira regra citada, basta que seja multiplicado os elementos da primeira coluna de A com os elementos da primeira linha de B, a soma do produto desses elementos vai resultar no primeiro elemento da matriz resultante, e assim também com a segunda e as demais colunas de A pelas linhas de B .

Algoritmo de Strassen: É um algoritmo recursivo que usa uma técnica chamada de divisão e conquista para realizar a operação de multiplicar matrizes. Seu objetivo é fazer com que seja reduzido o número de multiplicações necessárias para se obter a matriz resultante, o que facilita a multiplicação quando consideramos matrizes maiores. O algoritmo inicialmente divide as matrizes em quatro submatrizes de tamanhos iguais, realizando assim sete multiplicações de matrizes e também 18 somas para obter um resultado. Por ser recursivo o algoritmo chama a si mesmo até chegar a um resultado de matriz que possa ser multiplicado pelo método convencional.

4.1.4 Tamanhos de matrizes

Para realizar a análise de desempenho em matrizes, é necessário definir previamente as entradas que serão utilizadas, ou seja, o tamanho das matrizes. Esse tamanho pode variar de acordo com a situação. Por isso, foram definidos tamanhos pequenos, médios e grandes para as matrizes utilizadas nos testes.

Este trabalho se baseará nos estudos de Pacheco (PACHECO, 2019). Em sua pesquisa, ele conduz testes com matrizes variando de 32x32 até tamanhos equivalentes a 4096x4096. O objetivo é estabelecer uma comparação entre as abordagens concorrente e sequencial. Com base neste trabalho e nas informações apresentadas anteriormente, bem como nos métodos mencionados, foi possível determinar a média para os tamanhos de matrizes utilizados nos testes.

4.2 Algoritmos

4.2.1 O que são algoritmos?

Os algoritmos podem ser definidos como sendo qualquer procedimento computacional que possui valores de entrada, e que gera como resultado uma saída, portanto pode ser definido como sendo uma série de passos computacionais que, a partir de uma entrada passa por processos e instruções que resultam em uma saída lógica. Pode ser definido também como uma ferramenta utilizada para resolver problemas de computação específicos, que estabelecem uma relação entre entrada e saída (CORMEN *et al.*, 2009).

Um exemplo de um problema lógico seria a organização de números em ordem crescente. Nesse caso, a entrada poderia ser qualquer número digitado por um usuário [5,3,2,1,4], por exemplo. O algoritmo de ordenação deve gerar a saída [1,2,3,4,5] (CORMEN *et al.*, 2009).

Um conjunto de valores em sequência passados na entrada é chamado de instância de um problema. Basicamente, trata-se dos dados necessários para se calcular uma forma de resolver o problema (CORMEN *et al.*, 2009).

4.2.2 Forma sequencial

Um algoritmo pode ser definido como sequencial, quando uma entrada de dados é definida, e então ele percorre as instruções do código uma após a outra de forma que quando uma delas é iniciada ele deve terminar para dar início a outra instrução, essa sequência de passos tem como resultado um valor ou conjunto de valores conhecido como saída (CORMEN *et al.*, 2009).

4.2.3 Forma concorrente

Algoritmo concorrente ou programa concorrente pode ser definido como um conjunto de algoritmos sequenciais, mas que se comunicam entre si, podendo sincronizar e compartilhar

recursos por meio de uma comunicação, como uma memória compartilhada, nesse contexto as *threads* executam um papel importante na execução desses algoritmos (RAYNAL, 2013).

As *threads* são processos leves, que possuem um espaço de endereçamento compartilhado, fazendo com que os processos consigam se comunicar entre si com maior facilidade. Porém ao introduzir as *threads* em um programa deve se tomar cuidado pois uma série de complicações pode ser gerado como por exemplo duas ou mais *threads* podem entrar em conflito ao tentar acessar a região crítica do programa, que é a região onde há o acesso compartilhado (RAYNAL, 2013).

Os processos de um programa podem ser executados simultaneamente ou em paralelo considerando que haja um processador por processo, quando há menos processadores do que processos ocorre o que é chamado de pseudoparalelismo, que é basicamente quando os processos disputam pelo uso da CPU, nesse caso existe um agendador responsável por atribuir para cada processo um tempo de permanência no processador chamado de *quantum* (RAYNAL, 2013).

4.2.4 Forma distribuída

Dentro desse cenário, um sistema distribuído pode ser definido como um sistema onde componentes de hardware ou de software que pertencem à dispositivos independentes, estabelecem comunicação e coordenam suas tarefas e ações por meio de troca de mensagens (COULOURIS *et al.*, 2011).

No contexto abordado, é importante destacar o papel das redes de computadores, um tema de grande relevância e que não pode faltar quando se trata de sistemas distribuídos. Elas estão por toda a parte, e um exemplo delas é a própria internet, indispensável nos dias atuais, assim como as redes que dela fazem parte, como as redes de telefones móveis, corporativas ou domésticas.

Em questões geográficas computadores conectados por um rede pode estar separados por qualquer distância, podendo estar na mesma sala ou em outro continente (COULOURIS *et al.*, 2011).

Características importantes dos sistemas distribuídos:

Concorrência de componentes: A programação concorrente em uma rede de computadores é a norma, onde cada um utiliza de seu próprio dispositivo, e quando necessário é compartilhado recursos como páginas web ou arquivos. Se adicionado mais computadores à uma rede por exemplo a capacidade do sistema de lidar com o compartilhamento de recursos aumenta, o que ajuda a definir a escalabilidade do sistema (COULOURIS *et al.*, 2011).

Falta de um relógio global: Em sistemas distribuídos cada processo possui um relógio interno, por conta disso existe um limite do quanto os computadores conectados conseguem sincronizar seus relógios, essa falta de sincronização pode gerar problemas, ao exemplo

de um comércio eletrônico que precisa determinar qual transação ocorreu primeiro para evitar a venda do mesmo produto.

Como solução para esse problema existem algoritmos que fazem essa sincronização, porém não estabelecem um relógio global perfeito (COULOURIS *et al.*, 2011).

Falhas independentes de componentes: Basicamente diz respeito a capacidade do sistema de lidar com falhas individuais de cada componente, mantendo os outros ainda em execução. Essas falhas podem ocorrer na rede, o que leva ao isolamento dos computadores conectados a ela, porém não significa que eles vão parar de funcionar, da mesma maneira um pode haver uma falha no computador, ou término não desejado de um programa no sistema que é conhecido como *crash*, de maneira que os outros componentes continuarão em execução (COULOURIS *et al.*, 2011).

4.2.5 Algoritmos multiplicadores de matrizes

Considerando a possibilidade de implementação dos algoritmos de multiplicação de matrizes em três versões, a saber, sequencial, concorrente e distribuída. Este trabalho propõe uma avaliação de desempenho, levando em conta cada uma das versões e trazendo dados a respeito de sua performance.

Sequencial: Para o desenvolvimento deste algoritmo, foi utilizada a técnica convencional descrita no capítulo 5.1.3, que considera duas matrizes como entrada e gera a matriz resultante seguindo as regras padrão de multiplicação de matrizes. O algoritmo também calcula o tempo de execução, que é usado como a métrica para a avaliação de desempenho durante o trabalho. O algoritmo sequencial desenvolvido é discutido detalhadamente no capítulo 6.1.1 deste trabalho.

Concorrente: Essa operação foi realizada com o mesmo método do primeiro algoritmo, porém as tarefas foram divididas entre *threads* que realizaram as multiplicações referentes a cada elemento da matriz resultante de forma simultânea, para que uma *thread* contendo o resultado de todas as operações feitas realize a soma para a obtenção da matriz resultante final. Foram feitos testes considerando diferentes tamanhos de matrizes, analisando seu tempo de resposta. Os detalhes de como o algoritmo concorrente foi desenvolvido encontra-se no capítulo 6.1.2.

Distribuída: Para a implementação deste algoritmo, foram utilizados os paradigmas da programação distribuída. Foram utilizadas cinco máquinas conectadas em uma rede cabeada para que o experimento fosse possível. Uma máquina atuando como nó mestre, responsável por dividir a tarefa de multiplicação, enquanto as outras quatro máquinas, como nós trabalhadores, calculando suas respectivas partes da matriz resultante. Após o cálculo, cada nó trabalhador envia sua parte de volta ao nó mestre, que então monta a matriz resultante completa. Mais detalhes sobre a implementação do algoritmo distribuído podem ser encontrados no capítulo 6.1.3.

4.3 Programação sequencial, concorrente e distribuída em Java

Esse trabalho foi implementado utilizando a linguagem Java, pois é uma linguagem que aceita programação nas formas sequencial, concorrente e distribuída, além de ser uma linguagem que é usada hoje para programação voltada a *desktop*, *web*, é uma linguagem portável, ou seja seus códigos podem ser executados em diferentes sistemas operacionais e cumpre todos os requisitos necessários para a aplicação.

4.3.1 Primitivas utilizadas:

Programação sequencial: Assim como outras linguagens o Java permite a programação sequencial com comandos bem definidos permitindo a orientação a objetos e outros paradigmas da programação. Existem alguns tipos de comandos que a linguagem apresenta que são usados durante a programação, a exemplo de:

- Comandos de atribuição: A linguagem Java permite a criação de operadores de atribuição simples usados para atribuição de um valor a uma variável, usando o comando (`=`) como no exemplo `(int numero = 10;)`, dessa forma a variável “numero” recebeu o valor 10 (Oracle Corporation, 2023).
Java permite também operadores de atribuição combinada usando comandos como (`+=`, `-=`, `*=`), ao exemplo de uma variável que precisa que os seus valores sejam somados a uma quantidade sempre que o código passar por essa instrução. Funcionaria da seguinte maneira `(int numero; numero += 10;)` (Oracle Corporation, 2023).
- Comandos de declaração: Outro tipo de comando importante dentro do Java é o de declaração, por ser uma linguagem tipada, é necessário especificar o tipo da variável no momento da sua criação trazendo maior precisão para os comandos. Essas variáveis podem ser dos tipos *String*: armazena um conjunto de caracteres , *Integer*: usada no Java como `(int)` cria uma variável do tipo numérico inteiro , *boolean*: pode assumir apenas os valores *(true)* ou *(false)*, entre outros (Oracle Corporation, 2023).
- Comandos de seleção: Permitem alterar o fluxo do caminho do programa, impondo uma condição, por exemplo caso for necessário que uma ação seja feita apenas por um usuário maior de 18 anos podemos definir a seguinte condição:

```
1      int idade = 15;
2      if (idade >= 18) {
3          //acesso permitido
4      } else {
5          //acesso negado
```

```
6 }  
7
```

Nesse código é mostrado o comando (*if*) que verifica se a condição é verdadeira. Caso seja, ele executa as linhas de código dentro do bloco delimitado pelas chaves, enquanto o comando (*else*) é executado caso a condição não seja cumprida (Oracle Corporation, 2023).

- Comandos de repetição: Os comandos de repetição na programação são usados quando se quer realizar uma ordem repetidas vezes, ou até que uma condição seja cumprida, para realizar essa tarefa o Java permite a utilização de comandos como *for*, normalmente usado quando se sabe quantas vezes determinada ação irá ocorrer, ou o *while* que geralmente permanece rodando enquanto nenhuma condição é cumprida, para além desses métodos convencionais, o Java permite também métodos como *foreach* e *do-while* (Oracle Corporation, 2023).

Programação concorrente: Existem duas unidades básicas de execução no meio da programação concorrente: processos e *threads*, a linguagem Java está mais associada a programação com *threads*, mas suporta também a criação de processos.

Processos: Um processo possui um ambiente de execução autocontido, ou seja ele possui um conjunto privado de ferramentas e recursos enquanto está em execução e também possui seu próprio espaço de memória.

O que normalmente é visto pelo usuário como um único aplicativo, pode ser vários processos em conjunto estabelecendo uma conexão entre si (Oracle Corporation, 2023). Para facilitar essa comunicação são usados recursos de Comunicação entre Processos (IPC), como pipes e soquetes. Em Java a maioria das implementações são feitas com um único processo, mas a linguagem permite criar processos usando um objeto *ProcessBuilder* (Oracle Corporation, 2023).

Threads: A linguagem Java permite que um programa tenha diversas *threads* em execução simultânea, inicialmente quando um programa é criado em Java existe apenas uma *thread*, que faz a chamada do método *main* (Oracle Corporation, 2023).

Existem duas formas no Java de criar uma *thread* de execução. A primeira delas é criando uma nova classe que herda da classe *Thread* e sobrescrevendo o seu método *run()*, permitindo assim que uma instância da subclasse possa ser criada e iniciada corretamente. (Oracle Corporation, 2023)

A segunda forma é a criação de uma classe que implementa a interface *Runnable*, o que faz com que as *threads* sejam criadas em uma classe separada, permitindo a definição do comportamento da *thread*, através da implementação do método *run()*. Assim uma instância dessa classe pode ser gerada e iniciada sendo passada como argumento para o construtor da classe *Thread* (Oracle Corporation, 2023).

Para a execução de *threads* em paralelo é necessário que haja mecanismos para o controle de sincronização, que são utilizados para limitar o acesso das *threads* à região crítica, parte do programa em que há um compartilhamento de memória entre as *threads* (COULOURIS *et al.*, 2011).

A seguir, serão apresentados três importantes mecanismos de sincronização amplamente utilizados: semáforos, barreiras e variáveis de condição.

Semáforo: O semáforo é um mecanismo de sincronização que funciona fazendo com que uma *thread* ao tentar acessar um recurso compartilhado do programa, passe por um semáforo que irá verificar se aquele espaço está disponível para ser utilizado, caso esteja disponível, irá seguir normalmente sua execução. Quando uma *thread* tenta acessar essa região mas já está sendo utilizada, ela ficará no estado de espera até que a região crítica seja liberada para seu uso (COULOURIS *et al.*, 2011).

A linguagem Java permite esse controle através dos métodos *acquire()* e *release()* da classe *Semaphore*, quando o método *acquire()* é chamado o acesso a determinada região é bloqueado e a *thread* poderá acessá-la, até que uma permissão seja acionada, enquanto o método *release()* irá liberar o acesso ao método *acquire()*, que estava bloqueado (Oracle Corporation, 2023).

Barreiras: As barreiras permitem que *threads* esperem em um ponto específico do programa podendo sincronizar, ou compartilhar os resultados, é uma ferramenta útil para ocasiões onde as *threads* precisam de informações vindas de outra *thread* para prosseguir a execução (Oracle Corporation, 2023).

No Java a criação de barreiras é feita através da classe *CyclicBarries*, ao cria-lá é possível passar dois parâmetros um que indicara a quantidade de *threads* que irão esperar na barreira até que seja liberada, e o segundo parâmetro é uma ação opcional que será sempre executada quando se passa pela barreira (Oracle Corporation, 2023).

Variáveis de condição: Essas variáveis permitem que sejam criados métodos para que uma *thread* fique em espera até que uma determinada condição seja cumprida (COULOURIS *et al.*, 2011).

Em Java é possível fazer com que uma *thread* fique bloqueada enquanto uma condição não é atendida através do método *wait* que pertence a classe *Object* do Java. Para se desbloquear uma *thread* é utilizado o método *notify()*, e para desbloquear todas as *threads* que estejam esperando naquele determinado objeto utiliza-se *notifyAll()*, que também fazem parte da classe *Object* (COULOURIS *et al.*, 2011).

Programação distribuída: Para se estabelecer uma conexão entre máquinas com sistemas distribuídos em Java, são utilizados o *socket TCP (Transmission Control Protocol)* para realizar a comunicação por meio de uma conexão orientada a conexão e o *socket UDP (User Datagram Protocol)*, que transmite os dados de forma não orientada a conexão, o que o torna mais rápido porém menos seguro, comparado com o TCP, portanto é usado em

situações onde não há problema em perder alguns pacotes como em uma transmissão de vídeo em tempo real (COULOURIS *et al.*, 2011).

Socket TCP em Java:

O TCP oferece uma garantia de confiabilidade, permitindo que todos os dados enviados pelo transmissor sejam recebidos pelo receptor, e na mesma ordem. O protocolo TCP é orientado à conexão, o que exige a comunicação entre o transmissor e o receptor antes que qualquer mensagem seja transferida (COULOURIS *et al.*, 2011).

A camada TCP implementa alguns mecanismos para que a confiabilidade na entrega dos dados sejam garantidas. São eles:

Sequenciamento: O processo transmissor divide o fluxo em frações de dados que são transmitidas como pacotes IP. Para cada uma dessas partes, é anexado um número de sequência. O receptor usa esses números para ordenar as frações dos dados antes que sejam adicionadas ao seu fluxo de entrada. Nenhuma das partes pode ser adicionada ao fluxo sem que todas as frações de dados de números inferiores já tenham sido adicionadas. Caso alguma parte chegue antes, deve ser mantida em um *buffer* até que as partes de números anteriores cheguem, garantindo assim a entrega de todos os dados na ordem correta (COULOURIS *et al.*, 2011).

Controle de fluxo: O objetivo desse mecanismo é evitar que o receptor ou os nós intermediários sejam sobrecarregados. Ele funciona da seguinte forma: primeiramente, o receptor registra o número de sequência da fração dos dados sempre que a recebe com sucesso. Em intervalos de tempo, o receptor envia para o transmissor uma confirmação contendo o último número de sequência recebido corretamente, juntamente com um tamanho de janela, que é basicamente a quantidade de dados que ainda cabem no *buffer* antes que uma nova confirmação seja enviada. Por fim, se houver fluxo reverso de dados, ou seja, o receptor enviando dados para o transmissor, a confirmação é enviada junto com uma parte dos dados, economizando assim tempo e recursos. Caso contrário, é criado um segmento contendo apenas a confirmação (COULOURIS *et al.*, 2011).

Retransmissão: Esse mecanismo garante que, se algum dado não for enviado corretamente, ele seja reenviado. O transmissor salva os números de sequência dos dados enviados. Então, quando recebe uma mensagem de confirmação contendo os números de sequência já enviados, ele os exclui dos *buffers* de saída. Caso algum segmento não seja identificado dentro de um tempo estabelecido, ele é reenviado ao receptor (COULOURIS *et al.*, 2011).

Armazenamento em buffer: É utilizado para equilibrar o fluxo de dados entre o transmissor e o receptor. Caso o transmissor esteja enviando dados mais rapidamente do que o receptor consegue receber, o *buffer* de entrada do receptor armazena esses dados até que o receptor seja capaz de processá-los, evitando assim a perda dos dados. Se o *buffer* de entrada estiver cheio, os dados são excluídos, resultando em uma retransmissão dos mesmos (COULOURIS *et al.*, 2011).

Soma de verificação: Cada fração dos dados possui uma soma de verificação que cobre o cabeçalho e os dados presentes. Quando os dados chegam ao receptor, o valor da soma de verificação é lido, e um novo cálculo é realizado pelo receptor, que compara o resultado com o cálculo recebido do transmissor. Essa técnica é utilizada para conferir os dados, garantindo que, caso algum dado seja corrompido durante a transmissão, ele possa ser reenviado (COULOURIS *et al.*, 2011).

A programação em Java para esse protocolo é realizada através das classes *ServerSocket* que é usada para criar um objeto *socket* em uma porta de um servidor para que seja recebido do cliente solicitações de conexão. Para receber essa solicitação é utilizado o método *accept()*, que verifica uma fila de conexões enviadas do cliente, e caso não haja solicitações na fila, ela é bloqueada até que uma conexão seja fornecida. Quando fornecida o método retorna a instância de um objeto *socket*, que fará a comunicação com o cliente recebendo requisições e enviando respostas (COULOURIS *et al.*, 2011; Oracle Corporation, 2023).

A classe *Socket* permite que através do construtor o cliente crie um *socket*, passando como parâmetro o nome do *host*, gerado pelo DNS (*Domain name System*) e também a porta de um servidor, fazendo com haja também uma conexão com dispositivo remoto que foi especificado e a respectiva porta (COULOURIS *et al.*, 2011; Oracle Corporation, 2023).

Socket UDP em Java: Em Java é possível criar conexões de datagramas , através das classes *DatagramSocket* e *DatagramPacket*.

DatagramPacket: É composta por um pacote de datagrama que é especificado em seu construtor, onde é criado uma instância a partir de um *array* de *bytes* que formam uma mensagem, o seu comprimento, o endereço IP (*Internet Protocol*) e o número da porta local do *socket*. Os processos podem enviar ou receber a instância do *DatagramPacket*, quando essa mensagem é recebida pode ser acessada por meio do método *getData()*. Existem também os métodos *getPort()* e *getAddress* para que a porta e o endereço IP sejam acessados (COULOURIS *et al.*, 2011; Oracle Corporation, 2023).

DatagramSocket: Essa classe fornece um construtor que tem como parâmetro um número de porta, para processos que necessitam de uma porta específica. Ela permite também a criação de um construtor vazio, o que faz com que o sistema escolha uma porta local. No entanto, se a porta estiver sendo utilizada ou for uma porta reservada, haverá um erro de envio *SocketException* (COULOURIS *et al.*, 2011; Oracle Corporation, 2023).

Alguns métodos importantes utilizados na classe são:

Send: Uma instância da classe *DatagramPacket* que possui uma mensagem e o destino (COULOURIS *et al.*, 2011).

Receive: É um *DatagramPacket* inicialmente vazio , que irá armazenar os valores da mensagem, seu comprimento e a origem transmitidas pelo *socket* através do método *send()* (COULOURIS *et al.*, 2011).

setSoTimeout: Define um tempo limite em que o método *receive()* receberá os datagramas, quando o tempo expira o método é bloqueado, e então é lançado uma exceção *InterruptedException* (COULOURIS *et al.*, 2011).

Connect: É uma função que faz conexão entre os diferentes processos utilizando uma porta remota definida, o *socket* poderá trocar mensagens apenas com esse endereço.



METODOLOGIA

Este tópico apresentará as etapas seguidas para o desenvolvimento do trabalho, desde a escolha do ambiente e suas especificações até o desenvolvimento dos algoritmos e a coleta de dados.

5.1 Etapa 1: Definição do escopo do projeto

Esta etapa teve como objetivo analisar e definir o escopo, além de estudar e verificar a melhor abordagem para a realização do projeto e dos testes.

Este trabalho tem como objetivo analisar o desempenho de algoritmos multiplicadores de matrizes em três versões: sequencial, concorrente e distribuída. Para isso, foi conduzida uma pesquisa experimental com o propósito de definir o objeto de estudo, que são os algoritmos multiplicadores de matrizes.

Além disso, foram selecionadas as variáveis e definidas as formas de controle e observação, referentes à definição das matrizes e aos tamanhos a serem utilizados, bem como à análise de seu desempenho em relação ao tempo de resposta (PRODANOV; FREITAS, 2013).

Para a realização dos testes, a linguagem de programação Java foi escolhida para a implementação dos algoritmos, juntamente com o ambiente Eclipse. Além disso, foram utilizadas ferramentas e bibliotecas adequadas para cada uma das versões.

Os dados foram coletados durante a execução dos algoritmos, onde foi registrado o tempo de resposta da execução, considerando as matrizes em seus diferentes tamanhos, juntamente com as versões do algoritmo.

A análise dos dados ocorreu por meio da comparação dos algoritmos entre as versões sequencial, concorrente e distribuída. Foi utilizada uma abordagem quantitativa, na qual técnicas estatísticas foram necessárias para efetuar comparações e apresentar as principais diferenças

entre os resultados coletados (PRODANOV; FREITAS, 2013).

5.2 Etapa 2: Definição do local e ambiente de desenvolvimento

Inicialmente, o desenvolvimento do projeto ocorreu no computador pessoal, a fim de que os algoritmos pudessem ser produzidos sem a necessidade de se locomover ao laboratório, e alguns testes iniciais pudessem ser realizados para comparação entre os algoritmos e identificação de possíveis problemas.

Após desenvolvidos os algoritmos e testes realizados no computador pessoal, a coleta dos dados aconteceram no Instituto Federal de Educação, Ciência e Tecnologia Baiano Campus Guanambi. No laboratório de informática 1, localizado no pavilhão do curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Neste laboratório estão disponíveis as máquinas necessárias para a realização dos experimentos, incluindo uma conexão de rede com cabeamento CAT 5e e o uso de *switchs* para que haja uma comunicação eficiente e segura entre as máquinas durante a execução dos algoritmos distribuídos.

Figura 5 – Imagem do Laboratório 1 de informática



Fonte: Elaboração própria (2024)

5.2.1 Especificações de Hardware:

A fim de permitir comparação entre os resultados, foram escolhidas máquinas com especificações de *Hardware* similares. O laboratório não possui cinco máquinas com especificações

iguais, por esse motivo foram utilizadas aquelas com a maior semelhança possível. Considerando isso, os testes sequenciais e concorrentes foram realizados em cinco máquinas diferentes, com os resultados coletados para cada uma delas. Para os experimentos na forma distribuída, os mesmos computadores usados nas outras duas formas foram utilizados em conjunto para o sistema distribuído. Uma máquina (GBI0227410CMP) atuou como nó mestre, responsável por enviar as partes da matriz que cada nó irá calcular e montar a matriz resultante quando todas as partes forem recebidas. As outras quatro, nós trabalhadores, que têm a tarefa de calcular sua parte da matriz resultante e enviar de volta para o nó mestre.

Cada uma das máquinas possui uma etiqueta no gabinete que possibilita a identificação no laboratório. Nessa etiqueta é apresentado um número referente ao número da máquina. Por exemplo, a máquina GBI0227405CMP tem a etiqueta 0227405. Todas as máquinas seguem este mesmo padrão de etiquetagem, facilitando sua identificação.

As especificações das máquinas seguem o descrito nas tabelas.

Tabela 1 – Especificações da máquina GBI0227405CMP do laboratório

Nome do dispositivo	GBI0227405CMP
Memória	8,0 GB
Processador	Intel Core(tm) i5-3470 CPU @ 3.20 GHz x 4
Gráficos	Mesa Intel HD Graphics 2500 (IVB GTI)
Capacidade de disco	480,1 GB
Nome do SO	Ubuntu 22.04.4 LTS
Tipo do SO	64 bits

Fonte: Elaboração própria (2024)

Tabela 2 – Especificações da máquina GBI0227410CMP do laboratório

Nome do dispositivo	GBI0227410CMP
Memória	8,0 GB
Processador	Intel Core(tm) i5-3330 CPU @ 3.00 GHz x 4
Gráficos	Mesa Intel HD Graphics 2500 (IVB GTI)
Capacidade de disco	480,1 GB
Nome do SO	Ubuntu 22.04.3 LTS
Tipo do SO	64 bits

Fonte: Elaboração própria (2024)

Tabela 3 – Especificações da máquina GBI0227402CMP do laboratório

Nome do dispositivo	GBI0227402CMP
Memória	8,0 GB
Processador	Intel Core(tm) i5-3470 CPU @ 3.20 GHz x 4
Gráficos	Mesa Intel HD Graphics 2500 (IVB GTI)
Capacidade de disco	480,1 GB
Nome do SO	Ubuntu 22.04.3 LTS
Tipo do SO	64 bits

Fonte: Elaboração própria (2024)

Tabela 4 – Especificações da máquina GBI0227401CMP do laboratório

Nome do dispositivo	GBI0227401CMP
Memória	8,0 GB
Processador	Intel Core(tm) i5-3470 CPU @ 3.20 GHz x 4
Gráficos	Mesa Intel HD Graphics 2500 (IVB GTI)
Capacidade de disco	480,1 GB
Nome do SO	Ubuntu 22.04.3 LTS
Tipo do SO	64 bits

Fonte: Elaboração própria (2024)

Tabela 5 – Especificações da máquina GBI0227407CMP do laboratório

Nome do dispositivo	GBI0227407CMP
Memória	8,0 GB
Processador	Intel Core(tm) i5-3330 CPU @ 3.00 GHz x 4
Gráficos	Mesa Intel HD Graphics 2500 (IVB GTI)
Capacidade de disco	480,1 GB
Nome do SO	Ubuntu 22.04.3 LTS
Tipo do SO	64 bits

Fonte: Elaboração própria (2024)

Tabela 6 – Especificações do computador pessoal LAPTOP-HAKTI0UT

Nome do dispositivo	LAPTOP-HAKTI0UT
Ram instalada	8,00 GB
Processador	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
Gráficos	Intel(R) UHD Graphics
Capacidade de disco	250 GB SSD
Edição do SO	Windows 11 Home Single Language
Tipo de sistema	Sistema operacional de 64 bits

Fonte: Elaboração própria (2024)

5.2.2 Ambiente de desenvolvimento:

Será apresentado nesse o tópico o ambiente de desenvolvimento utilizado, bem como seu funcionamento e especificações. Para esse trabalho foi escolhido o ambiente de desenvolvimento Eclipse.

Eclipse: O Eclipse é uma IDE - *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado) e disponibiliza uma gama de ferramentas para o desenvolvimento em Java, o que a torna uma ótima opção para esse trabalho. Ele funciona através de subsistemas implementados em um ou mais *plug-ins*(Eclipse Foundation, 2023).

Workbench: Refere-se ao ambiente de desenvolvimento de *desktop*, e tem como objetivo trazer ferramentas para criar, gerenciar e navegar entre os recursos em seu projeto Java no espaço de trabalho, uma janela *workbench* possui perspectivas que contém editores e visualizações que controlam os menus e barras de ferramentas da plataforma (Eclipse Foundation, 2023).

Para este trabalho foi utilizado o *workbench* padrão do eclipse no desenvolvimento dos algoritmos.

5.3 Etapa 3: Estudo das matrizes e algoritmos

Após a definição do ambiente de desenvolvimento, foi iniciado o processo de estudo sobre como o algoritmo seria desenvolvido. Para isso, foi necessário estudar as particularidades das matrizes, pois elas possuem diversas características que podem ser levadas em conta na hora de realizar uma análise de seu desempenho. Sabendo disto, foi estabelecida a matriz que seria utilizada (matriz quadrada) para que a multiplicação pudesse seguir um padrão, facilitando o entendimento das análises realizadas.

Após esse estudo, foi realizada uma pesquisa sobre algoritmos que multiplicam matrizes, com um exemplo fornecido pelo orientador. Nesta etapa, buscou-se entender como o algoritmo se comportava e como realizava o cálculo das matrizes. Esse algoritmo estava em sua forma

concorrente, e esse estudo foi importante para a compreensão e planejamento de como seriam feitos os algoritmos nas formas sequencial, concorrente e distribuída, focando no objetivo do trabalho.

5.4 Etapa 4: Codificação dos algoritmos

Esta etapa se refere a como foram desenvolvidos os algoritmos sequencial, concorrente e distribuído.

5.4.1 Codificação do algoritmo concorrente

Com o ambiente de desenvolvimento e a linguagem de programação já escolhidos e configurados, o primeiro algoritmo a ser desenvolvido foi o concorrente. O motivo pelo qual este algoritmo foi desenvolvido primeiro se deve à base que já existia do algoritmo estudado. Esta etapa foi crucial para o desenvolvimento do trabalho, pois nela o primeiro dos três algoritmos foi desenvolvido, passando por um processo de validação com o orientador a fim de obter um algoritmo capaz de realizar os testes necessários com a maior precisão possível.

Os detalhes do algoritmo são apresentados no capítulo 6.1.2.

5.4.2 Codificação do algoritmo sequencial

Após o desenvolvimento do algoritmo concorrente, o próximo passo foi desenvolver o algoritmo sequencial. Esta etapa foi relativamente simples e concluída rapidamente, pois o algoritmo concorrente já havia sido desenvolvido e passado por um processo de validação. Dado isto, o processo realizado foi simplificar este algoritmo a fim de transformá-lo em sua forma sequencial, retirando os elementos de paralelismo, como as *threads*, para que fosse executado apenas pela *thread* principal, tornando-o um algoritmo sequencial.

Os detalhes do desenvolvimento deste algoritmo podem ser vistos no capítulo 6.1.1.

5.4.3 Codificação do algoritmo distribuído

Para esta fase, foi necessário um estudo mais aprofundado a respeito da programação distribuída, pois inicialmente seria utilizado o protocolo UDP, mencionado no capítulo 4.3.1. Entretanto, devido à limitação de espaço que uma mensagem era capaz de enviar, não foi possível utilizar esse protocolo, já que o limite alcançado pelo *buffer* foi uma matriz de tamanho 120x120, enquanto seriam necessárias matrizes de tamanhos superiores para o experimento. Por conta disso, o protocolo TCP foi a alternativa encontrada para permitir a transferência de matrizes de tamanhos maiores. Com isso, seguiu-se com o estudo juntamente com a realização de testes para verificar a possibilidade de enviar esses dados. Assim, o algoritmo distribuído foi concluído,

utilizando o protocolo TCP.

É possível visualizar mais informações a respeito do algoritmo distribuído no capítulo 6.1.3.

5.5 Etapa 5: Definição dos parâmetros para os testes

Nesta etapa, foram definidos os procedimentos e parâmetros utilizados na realização dos experimentos, visando garantir que os dados apresentados fossem confiáveis e precisos. Os parâmetros utilizados serão exibidos e detalhados a seguir:

Tamanho das matrizes: Esse parâmetro se refere ao tamanho das matrizes que foram utilizadas no experimento. Para isso foram definidos tamanhos de matrizes conforme especificações descritas no capítulo 4.1.4. Esses tamanhos são :

Matriz pequena: Pode ser definida como matrizes de tamanho 16x16, 32x32 ou ainda 64x64.

Matriz média: Para esta categoria foi utilizada matrizes de tamanhos 128x128, 256x256, 512x512.

Matriz grande: por fim os tamanhos das matrizes grandes utilizadas foram 1024x1024, 2048x2048 e 4096x4096.

Forma de Apresentação das matrizes: Para esse trabalho, os elementos das matrizes apresentaram uma forma padronizada, fazendo com que os resultados fossem mostrados de forma simples, facilitando assim a visualização e análise dos dados.

Quantidade de testes: Para obter resultados precisos, é necessário realizar uma grande quantidade de testes, pois diversos fatores externos podem influenciar no tempo de resposta, como um processo iniciado em segundo plano ou uma atualização automática do sistema.

A quantidade de testes realizados foi definida da seguinte forma: para os algoritmos sequencial e concorrente, foram realizados dez testes para cada tamanho de matriz, sendo três tamanhos pequenos, três médios e três grandes, em cada uma das cinco máquinas. Isso resultou em um total de quatrocentos e cinquenta experimentos para cada algoritmo (10 testes * 9 tamanhos de matrizes * 5 máquinas).

No caso do algoritmo distribuído, como utiliza as cinco máquinas em conjunto, foram realizados dez testes para cada tamanho de matriz, totalizando noventa testes (10 testes * 9 tamanhos de matrizes).

Além disso, foram realizados também noventa testes para cada algoritmo na máquina pessoal, para permitir uma comparação entre diferentes máquinas e descrever as semelhanças e diferenças entre elas.

5.6 Etapa 6: Geração dos gráficos e análise dos dados

Após a realização dos testes e a coleta dos resultados, foi iniciado um procedimento estatístico para calcular as médias das execuções, com o objetivo de transformar esses dados em gráficos, facilitando a visualização e compreensão dos resultados. As médias dos algoritmos sequenciais e concorrentes foram calculadas em duas etapas:

1. Primeiramente, para cada tamanho de matriz, foram realizadas dez execuções em cada uma das cinco máquinas, e a média dessas dez execuções foi calculada para cada máquina individualmente.

2. Em seguida, foi calculada a média das médias obtidas de cada máquina para cada tamanho de matriz. Por exemplo, para o tamanho 16x16, foi tirada a média das dez execuções em cada máquina e, posteriormente, calculada a média dessas médias entre as cinco máquinas.

Este processo permitiu obter uma média representativa do desempenho de cada algoritmo em diferentes tamanhos de matrizes, facilitando a comparação e análise dos resultados. O cálculo da média para o algoritmo distribuído foi realizado de forma mais direta, pegando as dez execuções de cada tamanho de matriz e dividindo pelo total de execuções.

Após a geração dos gráficos com os resultados obtidos, foi realizada uma análise dos dados, visando a coleta de informações relevantes entre os resultados, como a busca por padrões e relações entre os dados obtidos. Os dados foram interpretados, buscando entender o contexto em que os dados estão inseridos e a adicionando um significado para esses resultados.

Esta etapa teve por finalidade, fazer com que as informações coletadas fossem transformadas em um conhecimento útil para os leitores e passível de execução no ambiente envolvido. Trazendo uma conclusão e resultados de forma clara e bem estruturada.



ALGORITMOS DESENVOLVIDOS

6.1 Explicação dos algoritmos

Este tópico irá explicar de forma detalhada os algoritmos produzidos durante o processo de desenvolvimento do projeto. Os algoritmos podem ser encontrados no GitHub, no seguinte link: github.com/AdautoBenevides/Algoritmos-multiplicadores-matizes.

Os comandos para execução dos algoritmos estão descritos no apêndice B.

6.1.1 Algoritmo Sequencial

Este algoritmo utiliza uma abordagem simples e direta, onde os cálculos são realizados sequencialmente, sem nenhum tipo de concorrência. O código conta com algumas etapas principais:

Inicialização das matrizes: Nessa parte do código, as matrizes a, b e c são inicializadas, e as matrizes a e b são preenchidas com valores padrão.

Lógica de multiplicação: Aqui é apresentada a lógica da multiplicação, que ocorre de forma sequencial e segue a abordagem tradicional de multiplicação de matrizes. Cada elemento da matriz resultante c é obtido multiplicando os elementos correspondentes das linhas da matriz a pelos elementos correspondentes das colunas da matriz b, sendo o resultado a soma desses produtos. Este cálculo é realizado pela função *multiplyRowAndColumn*, que recebe como parâmetros as matrizes a e b, além das posições atuais da linha da matriz a e coluna da matriz b, bem como a ordem da matriz.

Código-fonte 1 – loop aninhado

```

1   for (int i = 0; i < order; i++) {
2       for (int j = 0; j < order; j++) {

```

```

3           c[ i ][ j ] = multiplyRowAndColumn( a , b , i , j ,
4           order );
5       }
6   }
```

O trecho do Código-fonte 1 é responsável pela lógica da multiplicação, que é feita através de laços aninhados e de uma função auxiliar.

O laço externo percorre as linhas da matriz resultante “c”. A variável “i” é o índice que aponta para a linha atual.

O laço interno percorre as colunas da matriz resultante “c” para a linha “i” atual. A variável “j” é o índice que aponta para a coluna atual.

A função *multiplyRowAndColumn* é chamada para calcular o elemento na posição “c[i][j]” da matriz “c”.

Código-fonte 2 – Função de multiplicação da matriz

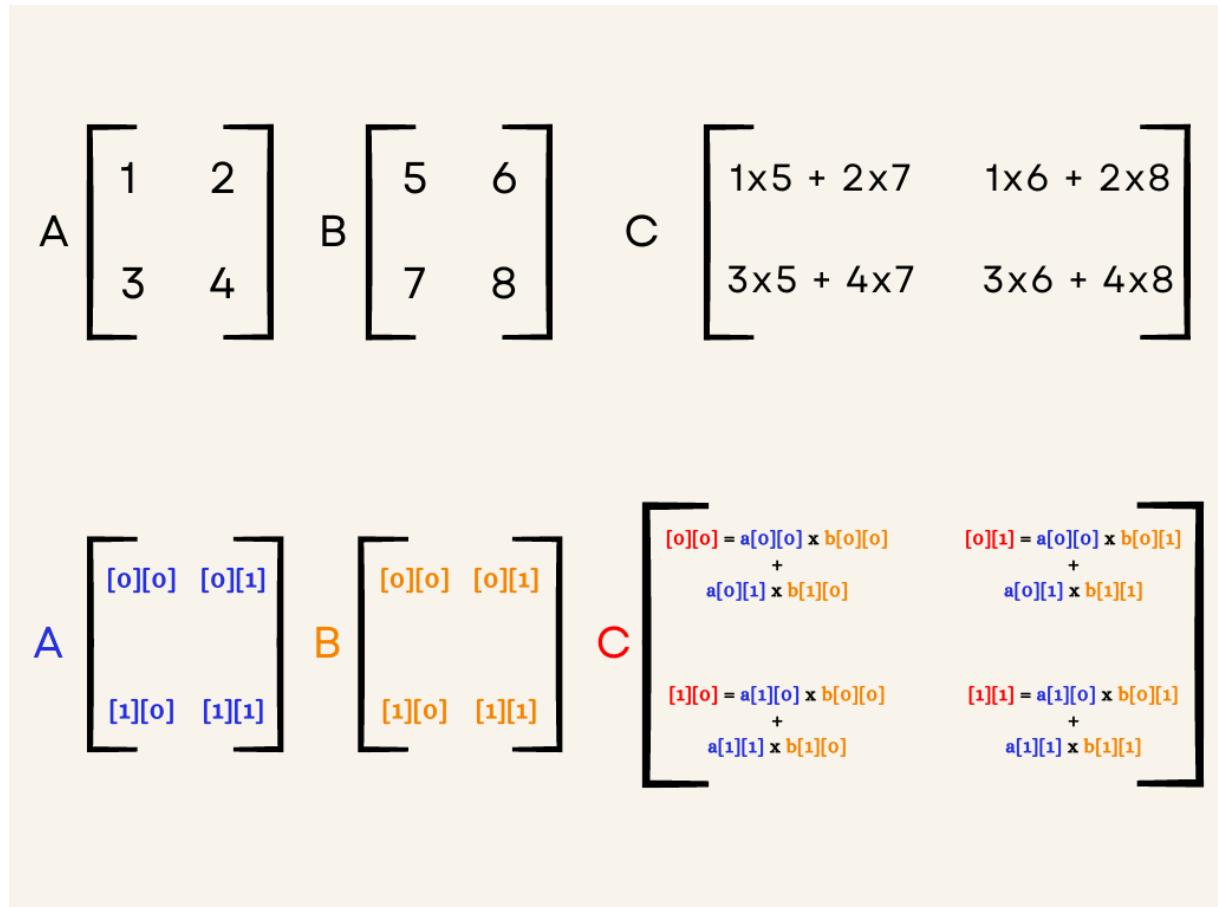
```

1  public static int multiplyRowAndColumn( int[][] a ,
2  int[][] b , int row , int col , int order ) {
3      int result = 0;
4      for ( int i = 0; i < order; i++ ) {
5          result += a[ row ][ i ] * b[ i ][ col ];
6      }
7      return result ;
8  }
```

A função apresentada no Código-fonte 2 realiza a multiplicação do elemento da matriz resultante, pegando a linha da matriz “a” atual multiplicando a coluna da matriz “b” atual.

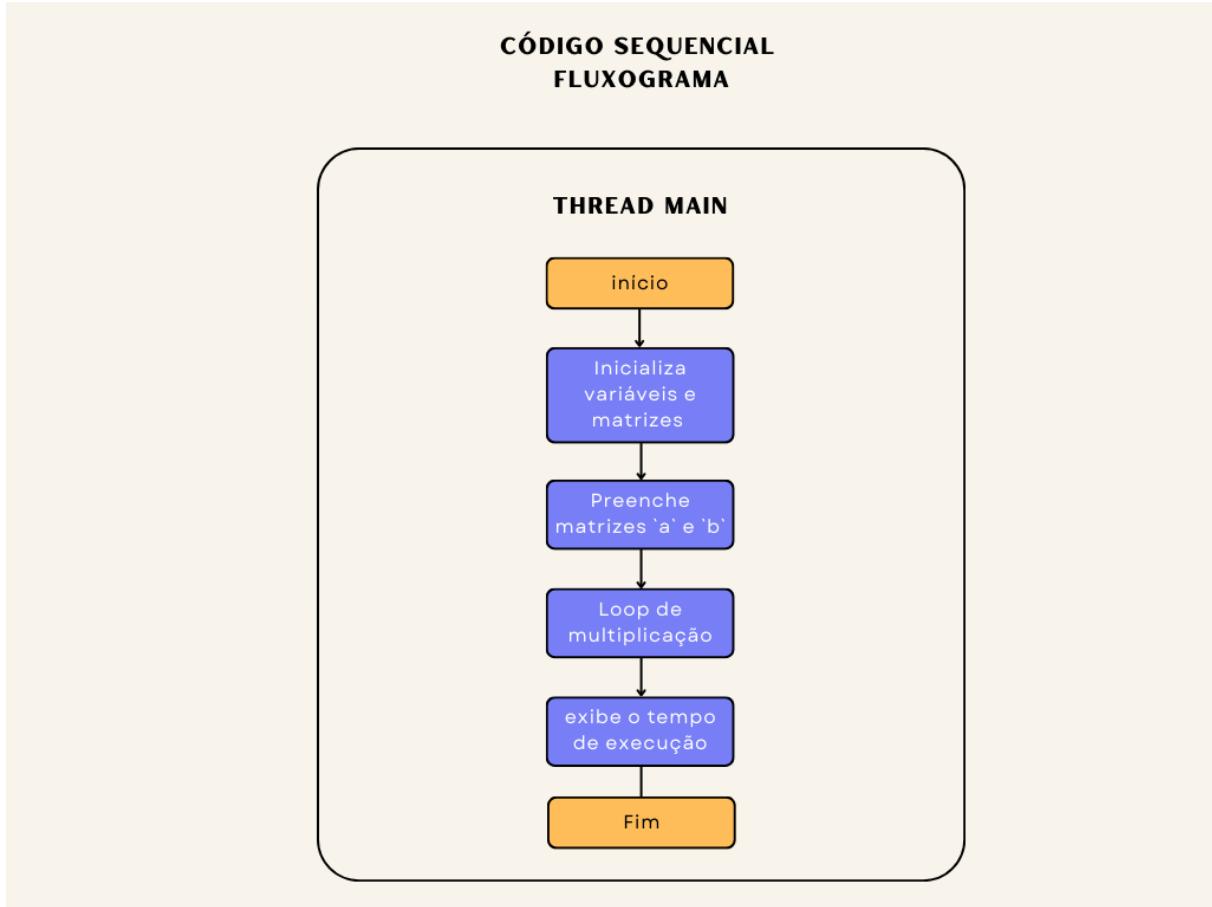
O laço *for* é repetido até que todos os elementos da linha de “a” sejam multiplicados pela coluna de “b” e somados, obtendo assim o resultado da matriz “c” na posição indicada pelos parâmetros.

Figura 6 – Exemplo do funcionamento da função para uma matriz de tamanho 2x2, onde na primeira parte é mostrado o processo padrão da multiplicação, enquanto na segunda é mostrado o processo dos elementos no algoritmo, sendo os elementos da matriz “a” representados pela cor azul, os elementos da matriz “b” pela cor amarela, e os elementos da matriz “c” pela cor vermelha



Fonte: Elaboração própria (2024)

Figura 7 – Fluxograma do código sequencial que mostra os passos realizados pelo algoritmo para concluir a tarefa da multiplicação



Fonte: Elaboração própria (2024)

6.1.2 Algoritmo Concorrente

O algoritmo concorrente procura melhorar o desempenho da multiplicação de matrizes utilizando *threads*. Cada *thread* fica responsável por calcular parte da matriz resultante, dividindo assim o trabalho entre elas, para isso temos as etapas principais:

Inicialização de matrizes: Como no algoritmo sequencial, as matrizes a, b e c são iniciadas e as matrizes a e b são preenchidas.

Divisão em blocos e *threads*: Nessa parte do código o algoritmo divide as operações em blocos de linhas e usa as *threads* para executar esses blocos de forma concorrente. A quantidade de *threads* é definida dividindo a ordem da matriz pela variável ‘*blocksize*’, que é a quantidade de linhas que cada *thread* irá calcular (para o trabalho foi definido como 8 linhas) .

Lógica da multiplicação concorrente: Cada uma das *threads* fica encarregada por calcular parte da matriz c. Essas *threads* são criadas, iniciadas e calculam sua parte por meio da função “*multiplyRowAndColumn*” de forma concorrente. Por fim, elas aguardam até que todas

as *threads* tenham terminado a execução para montar a matriz resultante *c*.

Código-fonte 3 – lógica da criação de threads

```

1   Thread[] threads = new Thread[order / blockSize];
2
3   for (int i = 0; i < order / blockSize; i++) {
4
5       final int startRow = i * blockSize;
6       final int endRow = startRow + blockSize;
7       final int finalOrder = order;
8
9       Thread thread = new Thread(() -> {
10          for (int j = startRow; j < endRow; j++) {
11              for (int k = 0; k < finalOrder; k++) {
12                  c[j][k] = multiplyRowAndColumn(a, b, j, k,
13                      finalOrder);
14              }
15          }
16      });
17
18      thread.start();
19      threads[i] = thread;
20  }

```

Inicialmente, é possível notar no Código-fonte 3 que um vetor de *threads* é criado para que seja possível gerenciá-las. O tamanho desse vetor é definido pela ordem da matriz dividida pela variável *blocksize*, que possui o valor 8, o que determina a quantidade de *threads*. Caso, por exemplo, a ordem da matriz seja igual a 32, este valor será dividido por 8, totalizando 4 *threads*. Desta forma, cada uma das *threads* ficará responsável por calcular 8 linhas da matriz resultante “*c*”.

Após esta etapa, um laço de repetição é iniciado para garantir que cada *thread* calculará a porção correta da matriz resultante. Antes da criação da *thread*, são definidas a linha inicial e a linha final que serão calculadas na iteração atual do *for*.

Então, uma nova *thread* é criada para calcular uma porção da matriz resultante. O cálculo é realizado da mesma forma que na matriz sequencial, utilizando laços aninhados e a função *multiplyRowAndColumn*.

Depois deste processo, a *thread* é iniciada e armazenada no vetor. O procedimento se

repete até que todas as *threads* sejam iniciadas.

Código-fonte 4 – Espera das threads por meio do método join

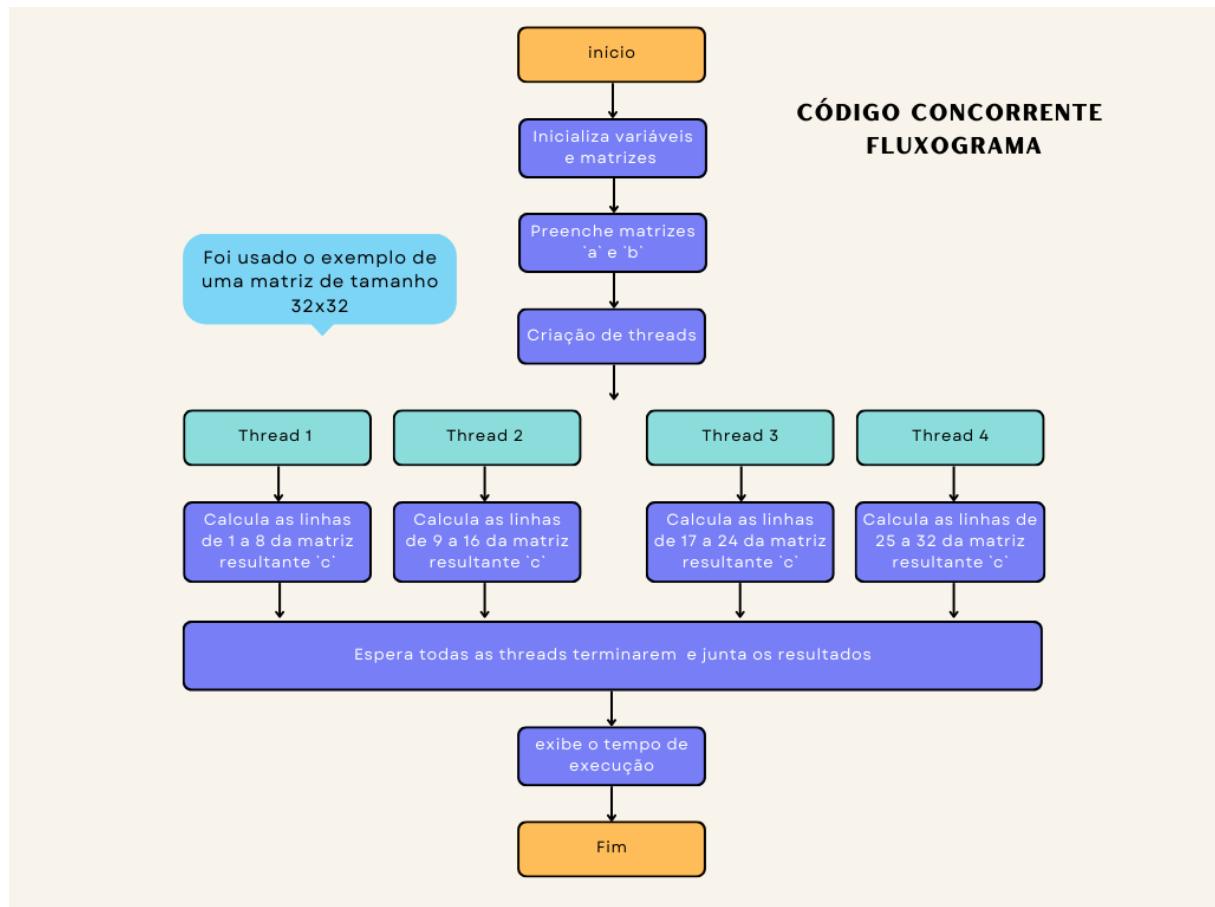
```

1   for (int i = 0; i < order / blockSize; i++) {
2       try {
3           if (threads[i] != null) {
4               threads[i].join();
5           }
6       } catch (InterruptedException e) {
7           e.printStackTrace();
8       }
9   }

```

O trecho do Código-fonte 4 mostra o processo de espera das *threads*. O método *join* é usado para garantir que todas as *threads* tenham terminado sua execução antes do programa prosseguir.

Figura 8 – Diagrama representando o fluxo de procedimentos realizados para o cálculo de uma matriz de ordem 32x32 no algoritmo concorrente



Fonte: Elaboração própria (2024)

Tabela 7 – Quantidade total de *threads* usadas no algoritmo concorrente para cada ordem de matriz. O cálculo foi dado pela quantidade de *threads* criadas no algoritmo mais a *thread main*.

Tamanho da Matriz	Quantidade de Threads Totais
16x16	3
32x32	5
64x64	9
128x128	17
256x256	33
512x512	65
1024x1024	129
2048x2048	257
4096x4096	513

Fonte: Elaboração própria (2024)

6.1.3 Algoritmo Distribuído

Nó mestre

O algoritmo distribuído possuí uma abordagem diferente dos algoritmos anteriores, por necessitar de uma conexão em rede para o seu funcionamento. O nó mestre será aquele responsável por enviar para cada nó trabalhador a parte que ele calculará. Para isso o nó mestre deve seguir as seis etapas principais mencionadas abaixo.

Inicializar as variáveis e sockets: Nesta primeira etapa, a ordem da matriz é definida através do parâmetro passado na execução do nó mestre, além da definição do número de porta, quantidade de nós trabalhadores e criação de uma lista de *sockets* que servirão para estabelecer uma conexão entre o nó mestre e os nós trabalhadores.

Esperar os nós trabalhadores conectarem: Nesta fase, o nó mestre cria um novo *socket* que aguarda conexões dos nós trabalhadores por meio da porta fornecida. A conexão permanece aberta até que todos os nós trabalhadores se conectem. A cada conexão, o *socket* do trabalhador conectado é adicionado à lista de *sockets*.

Criar conexão: Neste passo, uma classe de conexão é criada, contendo em seu construtor uma lista de *sockets*, o número de trabalhadores e a ordem da matriz. Esta classe gerencia a comunicação entre o nó mestre e os nós trabalhadores para a multiplicação das matrizes.

Enviar para os nós trabalhadores suas respectivas partes para o cálculo: Nesta etapa, o nó mestre é responsável por dividir a matriz “a” em partes iguais para enviá-las aos nós trabalhadores, juntamente com a matriz “b” por inteiro, permitindo assim o cálculo das linhas da matriz resultante.

Receber as partes da matriz resultante dos nós trabalhadores e montar a matriz

resultante: Na fase final, o nó mestre recebe de cada trabalhador uma parte da matriz resultante já calculada, ficando responsável por montar a matriz completa.

Código-fonte 5 – Inicialização de variáveis

```

1   if ( args . length != 1 ) {
2       System . out . println ( "Uso: java TCPServer <ordem>" );
3       return ;
4   }
5
6   int order = Integer . parseInt ( args [ 0 ] );
7   int serverPort = 1234;
8   int numWorkers = 4;
9   List < Socket > workerSockets = new ArrayList <>();
```

Neste trecho do Código-fonte 5 é apresentada a maneira de iniciar o código mestre, com o comando “java” seguido pelo nome da classe e, por fim, a ordem da matriz que será calculada. O argumento da ordem será armazenado em uma variável. Em seguida, é criada uma variável para armazenar a porta com um número que não esteja em uso pela máquina. Além disso, define-se o número de trabalhadores e cria-se uma lista para armazenar suas conexões.

Código-fonte 6 – Criação de conexão

```

1   try ( ServerSocket listenSocket =
2       new ServerSocket ( serverPort ) ) {
3       System . out . println ( "Servidor ouvindo na porta " +
4           + serverPort );
5
6       while ( workerSockets . size () < numWorkers ) {
7           Socket clientSocket = listenSocket . accept ();
8           System . out . println ( "Cliente conectado: " +
9               + clientSocket . getInetAddress () .
10              getHostAddress () );
11           workerSockets . add ( clientSocket );
12       }
13
14       Connection connection =
15       new Connection ( workerSockets , numWorkers , order );
16       connection . start ();
17
18   } catch ( IOException e ) {
```

```

19         System.out.println("Erro ao iniciar o servidor: "
20             + e.getMessage());
21     }

```

No Código-fonte 6, é destacado o momento em que o mestre aguarda conexões dos trabalhadores, criando um *socket* com a porta do nó mestre como parâmetro para a conexão. Quando um trabalhador se conecta, é usado o método *accept()* do *socket* criado para que a comunicação seja estabelecida. Em seguida, o *socket* do trabalhador é armazenado na lista *workerSockets*, e o processo se repete até que todos os trabalhadores estejam conectados. Subsequentemente, a classe de conexão é criada e instanciada com os parâmetros *workerSockets*, que irá conter a lista com as conexões dos nós trabalhadores, *numWorkers*, a quantidade de nós trabalhadores, e *order*, a ordem da matriz. Por fim, esta classe é iniciada pelo método *start()*.

Código-fonte 7 – Criação da classe de conexão

```

1
2   static class Connection extends Thread {
3       List<Socket> workerSockets;
4       int numWorkers;
5       int order;
6
7       public Connection(List<Socket> workerSockets,
8           int numWorkers, int order) {
9           this.workerSockets = workerSockets;
10          this.numWorkers = numWorkers;
11          this.order = order;
12      }

```

O Código-fonte 7 define os atributos, o método construtor e a herança da classe *Connection*. Esta classe herda de *Thread* para que possa ser executada utilizando o método *start()*, permitindo a conexão entre os nós trabalhadores e o nó mestre.

Código-fonte 8 – Divisão de tarefa para cada nó trabalhador

```

1   for (int i = 0; i < numWorkers; i++) {
2       Socket workerSocket = workerSockets.get(i);
3       ObjectOutputStream out =
4           new ObjectOutputStream(workerSocket.
5               getOutputStream());
6
7       int[][] aPart = new int[partSize][order];
8       System.arraycopy(a, i * partSize, aPart, 0,

```

```

10     partSize);
11
12     out.writeObject(aPart);
13     out.writeObject(b);
14
15     out.flush();
16 }
```

No Código-fonte 8, um *loop* é percorrido para fazer a divisão das matrizes que serão enviadas aos trabalhadores. Primeiramente, o nó mestre utiliza a lista de conexões para estabelecer uma conexão com o trabalhador referente à iteração do laço. Em seguida, cria uma classe *ObjectOutputStream* para que a matriz possa ser serializada e, assim, enviada para o nó trabalhador. A matriz “aPart” é criada e uma parte da matriz “a” original é copiada para ela. Esta cópia é realizada por meio do método *arraycopy* fornecido pelo Java. Por fim, a classe *ObjectOutputStream* é utilizada para serializar esta parte da matriz “a” e também a matriz “b”, enviando essas duas matrizes ao trabalhador por meio do método *flush()*. Esta operação se repete para todos os trabalhadores, cada um recebendo uma porção diferente da matriz “a” para que seja possível o cálculo da matriz resultante.

Código-fonte 9 – Recebe e monta a matriz resultante

```

1   for (int i = 0; i < numWorkers; i++) {
2       Socket workerSocket = workerSockets.get(i);
3       ObjectInputStream in =
4           new ObjectInputStream(workerSocket.
5               getInputStream());
6
7       int[][] cPart = (int[][]) in.readObject();
8       int startRow = i * partSize;
9
10      System.arraycopy(cPart, 0, c, startRow, partSize);
11      in.close();
12
13      workerSocket.close();
14 }
```

O Código-fonte 9 descreve a lógica do recebimento da matriz resultante e de sua montagem. Inicialmente, é estabelecida uma conexão com o trabalhador. Após essa conexão, a classe *ObjectInputStream* é usada para receber do trabalhador uma parte da matriz “c” por meio do método *readObject()*. Em seguida, essa parte é copiada para a matriz resultante final, e a conexão com esse trabalhador é fechada. O processo se repete até que todos os trabalhadores tenham

enviado suas partes e a matriz resultante tenha sido montada.

Nós trabalhadores :

Os nós trabalhadores são responsáveis por receber do nó mestre uma fração da matriz “c” para calcular. Esse cálculo é realizado utilizando as matrizes “a” e “b”, também fornecidas pelo nó mestre. Os nós trabalhadores seguem as seguintes etapas:

Inicialização do *socket* e obter endereço ip e porta: Nesta primeira etapa, o nó trabalhador irá inicializar um *socket* para estabelecer uma conexão com o nó mestre, além de criar as variáveis referentes ao IP e à porta do nó mestre, que são passados por meio de argumentos na linha de comando de execução do nó trabalhador.

Estabelecer conexão e receber matrizes: O nó trabalhador estabelece uma conexão com o nó mestre por meio do IP e da porta, e recebe uma parte da matriz “a” e a matriz “b” para realização do cálculo.

Criação das *threads* e cálculo da multiplicação das matrizes: Esta é uma fase fundamental, na qual é definida a quantidade de *threads* por meio da função *calculateNumThreads*, que se baseia na ordem da matriz para determinar quantas *threads* serão usadas no cálculo. Além disso, nesta fase, são definidas as linhas que cada *thread* irá calcular, e essas linhas são calculadas por meio da função *multiplyRowAndColumn*.

Envio do resultado do cálculo para o nó mestre: Por fim, o resultado do cálculo referente à parte da matriz “c” é enviado de volta ao nó mestre para que a matriz possa ser montada, finalizando o processo de multiplicação da matriz no sistema distribuído.

Código-fonte 10 – Obtenção dos parâmetros por meio de argumentos

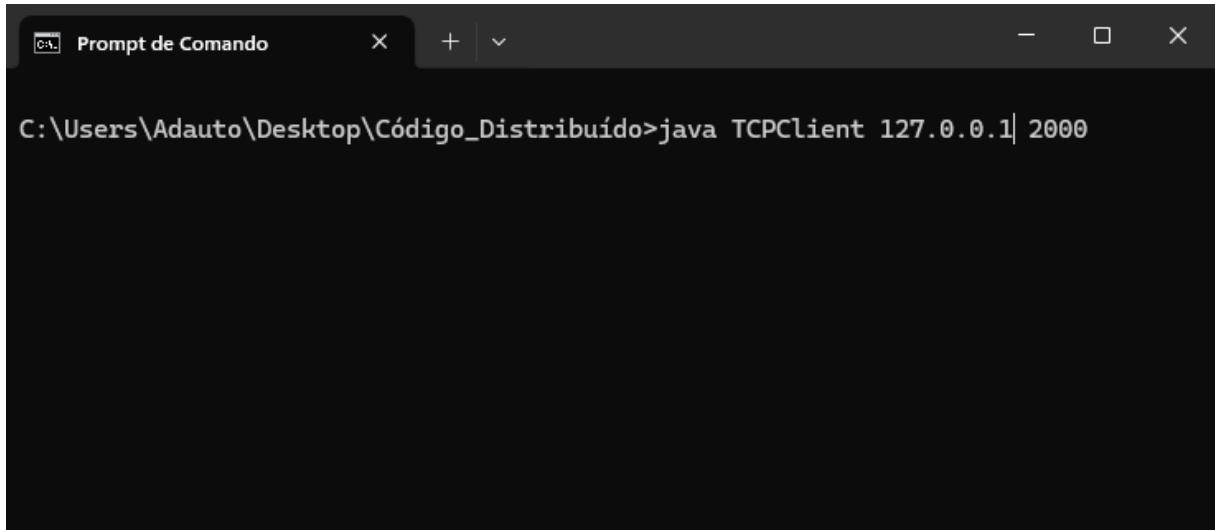
```

1   if (args.length != 2) {
2       System.out.println("Uso: java TCPClient
3           <ip_servidor> <ServerPort>");
4       return;
5   }
6   Socket s = null;
7
8   String serverIp = args[0];
9   String serverPort = args[1];

```

Os nós trabalhadores são iniciados no terminal usando o comando “java” seguido do nome da classe e do IP e porta do nó mestre, como é possível ver no Código-fonte 10, e na Figura 9. Após isso, um *socket* é iniciado para estabelecer conexão com o nó mestre, e a porta e o IP obtidos dos argumentos são guardados em variáveis.

Figura 9 – Demonstração do comando de execução do nó trabalhador, onde o comando java é utilizado seguido do nome da classe e dos argumentos: o primeiro é o número IP da máquina mestre, e o segundo é a porta em que o nó mestre está sendo executado.



Fonte: Elaboração própria (2024)

Código-fonte 11 – Recebimento das matrizes para o cálculo

```

1   s = new Socket(serverIp, Integer.parseInt(serverPort));
2   ObjectInputStream in =
3   new ObjectInputStream(s.getInputStream());
4   ObjectOutputStream out =
5   new ObjectOutputStream(s.getOutputStream());
6
7   int[][] aPart = (int[][]) in.readObject();
8   int[][] b = (int[][]) in.readObject();

```

O Código-fonte 11 esclarece como as matrizes são recebidas pelo nó trabalhador. Primeiramente, é estabelecida uma conexão com o nó mestre através do *socket*. Logo depois, é criado um *ObjectInputStream* para receber as matrizes serializadas. O método *readObject()* é utilizado para receber a matriz e armazená-la em uma variável. Esse processo ocorre com a parte da matriz “a” correspondente ao nó atual e a matriz “b”.

Código-fonte 12 – Lógica da criação das threads e cálculo da multiplicação das matrizes

```

1   int order = b.length;
2   int partSize = aPart.length;
3
4   int numThreads = calculateNumThreads(order);
5
6

```

```

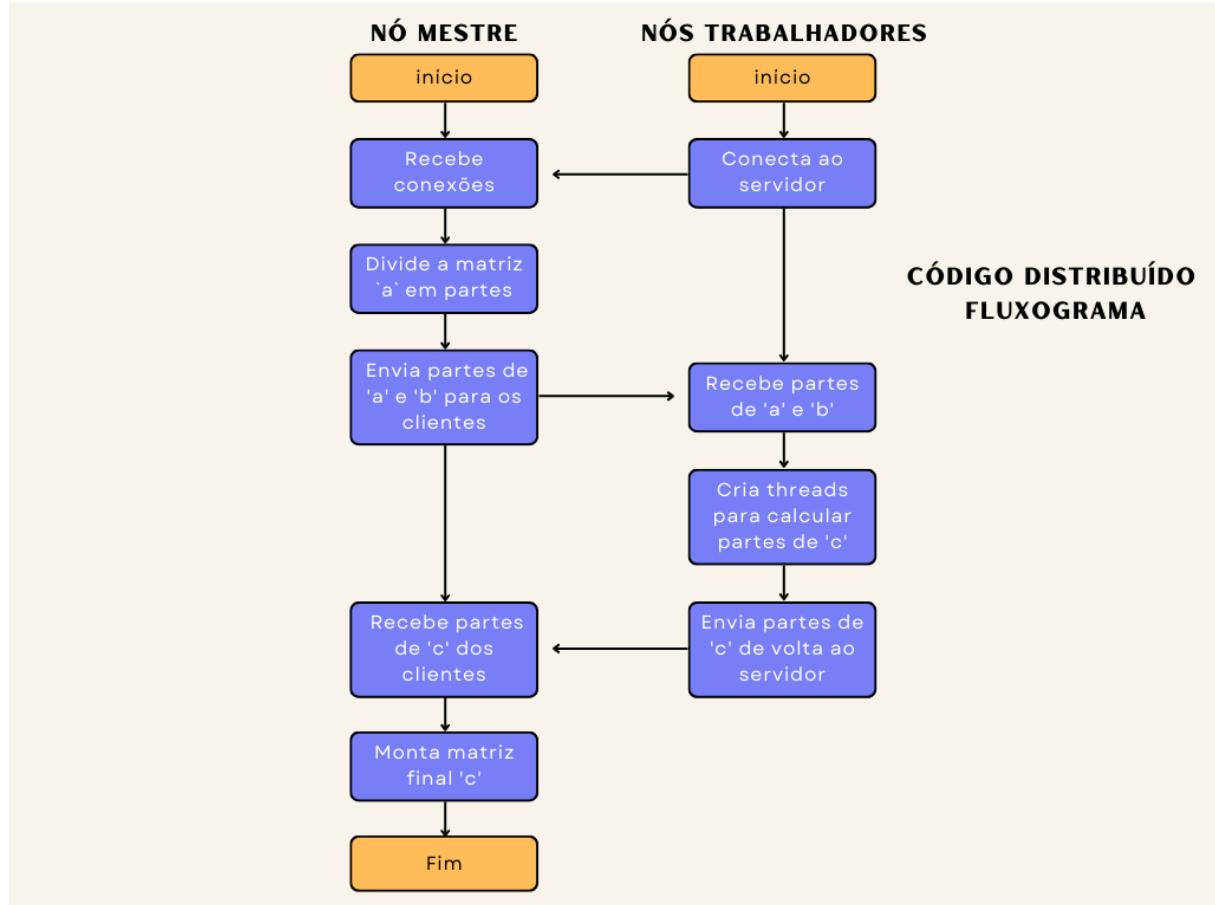
7     int partPerThread = partSize / numThreads;
8     final int[][] cPart = new int[partSize][order];
9
10
11    Thread[] threads = new Thread[numThreads];
12
13    for (int i = 0; i < threads.length; i++) {
14
15        final int threadStartRow = i * partPerThread;
16        final int threadEndRow = (i + 1) * partPerThread;
17
18        threads[i] = new Thread(() -> {
19
20            for (int row = threadStartRow; row <
21                threadEndRow; row++) {
22                for (int col = 0; col < order; col++) {
23                    cPart[row][col] = multiplyRowAndColumn
24                        (aPart, b, row, col, order);
25                }
26            }
27        });
28        threads[i].start();
29    }

```

O Código-fonte 12 apresenta uma variável chamada “`partSize`”, cujo tamanho se refere ao tamanho da parte da matriz “`a`”. O número de `threads` é definido pela função `calculateNumThreads`, que se baseia na ordem para determinar a quantidade de `threads` que será utilizada no processo. “`partPerThread`” é uma variável que define a quantidade de linhas a serem processadas por cada `thread`.

Após isso, um *loop* é criado para iniciar todas as `threads`. Dentro do *loop*, a linha inicial que a `thread` irá calcular e a última linha a ser calculada naquela iteração do laço são determinadas. Em seguida, uma `thread` é criada com essas informações, permitindo que a função de multiplicação utilizada nos outros algoritmos seja aplicada com esses parâmetros, fazendo com que cada `thread` calcule a quantidade de linhas necessárias para montar a parte da matriz resultante referente àquele nó.

Figura 10 – Fluxograma que mostra as etapas e procedimentos realizados pelo nó mestre e nós trabalhadores no algoritmo distribuído



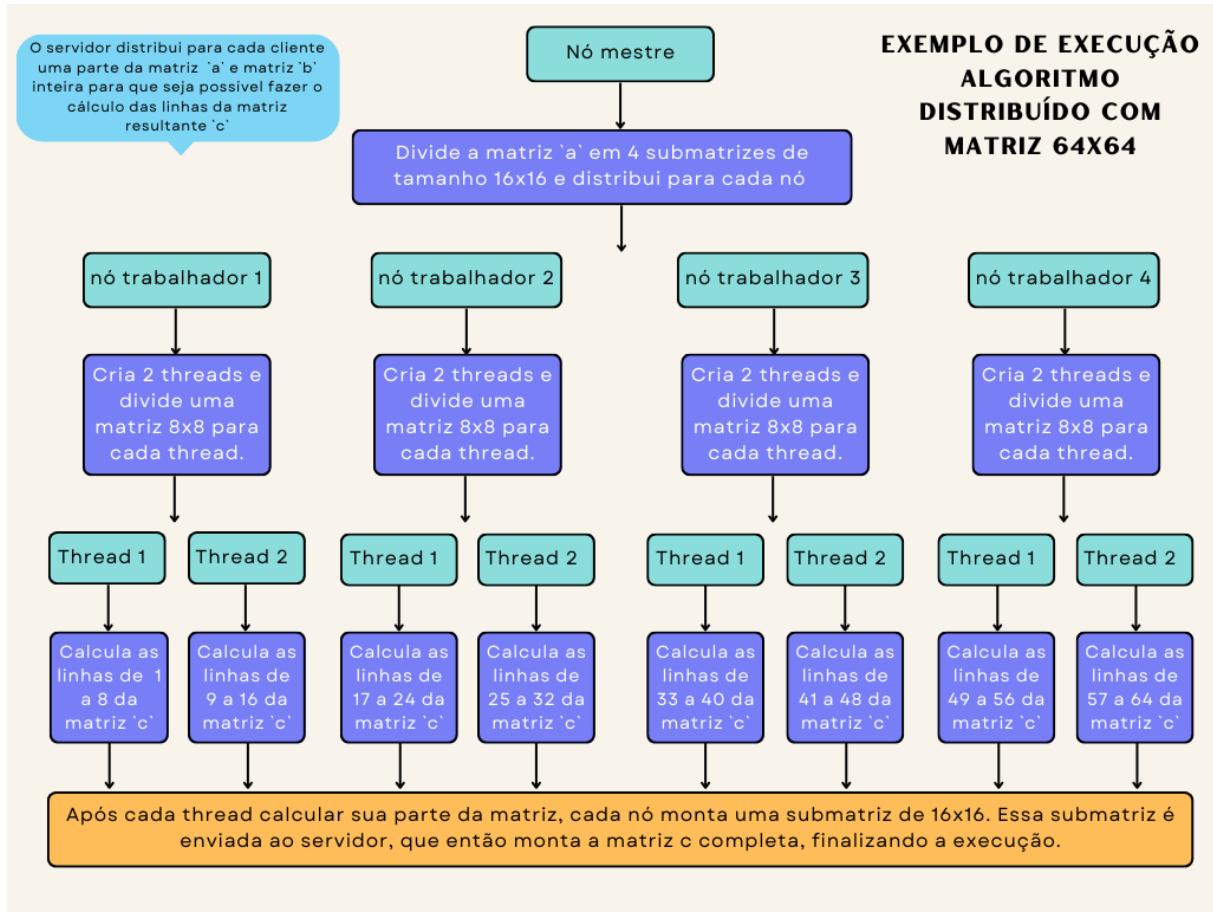
Fonte: Elaboração própria (2024)

Tabela 8 – Informações de configuração para execuções distribuídas. A tabela mostra a quantidade total de *threads*, calculadas considerando as *threads main* dos nós trabalhadores, mais a quantidade de *threads* criadas para a execução, é apresentado também a quantidade de nós trabalhadores usados para cada tamanho da matriz.

Tamanho da Matriz	Quantidade de Threads Totais	Quantidade de Nós
16x16	2	2
32x32	4	4
64x64	12	4
128x128	20	4
256x256	36	4
512x512	68	4
1024x1024	132	4
2048x2048	260	4
4096x4096	516	4

Fonte: Elaboração própria (2024)

Figura 11 – Fluxograma apresentando um exemplo de execução do algoritmo distribuído considerando uma matriz de ordem 64x64



Fonte: Elaboração própria (2024)

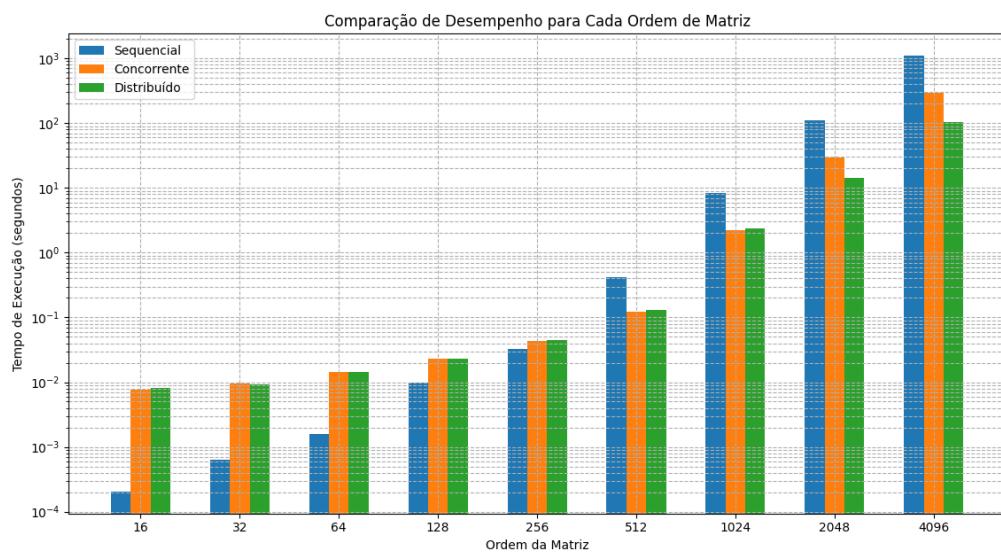
RESULTADOS E DISCUSSÃO

7.1 Análise dos resultados

Nesta seção, será apresentada uma análise dos resultados obtidos através da execução dos algoritmos sequencial, concorrente e distribuído, conforme o planejamento dos experimentos. Esta avaliação nos permite entender o desempenho de cada abordagem considerando os diferentes cenários, sempre fazendo um paralelo entre as versões sequencial, concorrente e distribuída com o objetivo de entender qual delas apresenta melhores resultados em cada cenário.

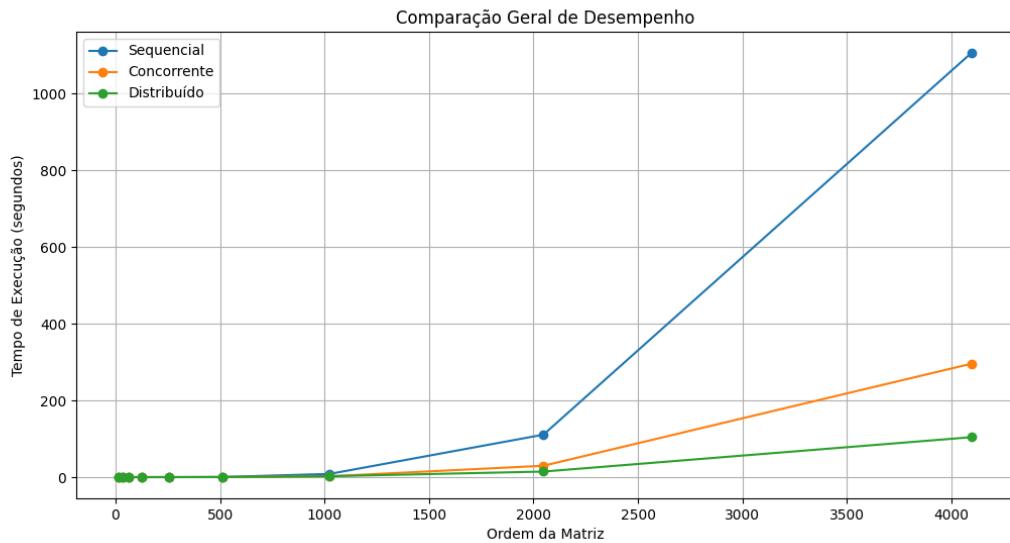
7.1.1 Comparação geral

Figura 12 – Gráfico 1: Gráfico em barras dos resultados das execuções dos algoritmos



Fonte: Elaboração própria (2024)

Figura 13 – Gráfico 2: Gráfico em linhas dos resultados das execuções dos algoritmos



Fonte: Elaboração própria (2024)

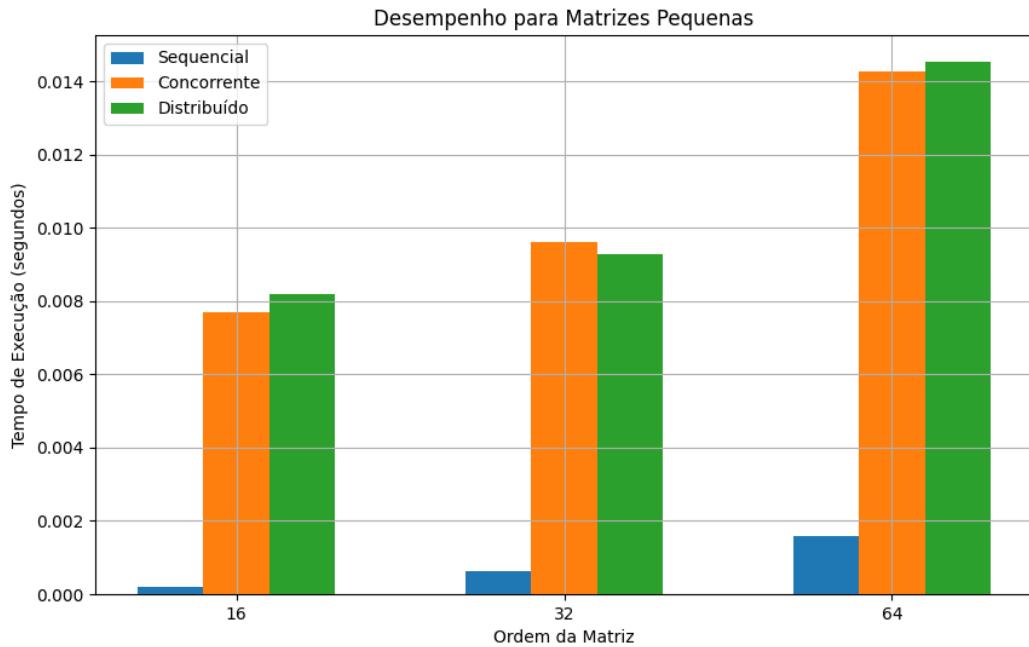
Ao analisar os gráficos, é possível extrair informações relevantes para a compreensão dos resultados dos testes realizados. Em um primeiro momento, visualizando o gráfico 2, é possível perceber que, conforme o tamanho das matrizes aumenta, existe uma tendência do algoritmo sequencial ter um tempo de execução maior do que os algoritmos concorrente e distribuído, em termos de tempo de resposta. Isso ocorre pelo fato de que, conforme o tamanho da matriz aumenta, mais recursos computacionais e processamento são necessários para que o cálculo da multiplicação seja efetuado.

Observa-se que o algoritmo concorrente em matrizes maiores obtém resultados mais rápidos em relação ao sequencial. Tal resultado acontece porque o algoritmo concorrente utiliza *threads* para compartilhamento de recursos, fazendo com que o trabalho de multiplicar as matrizes seja dividido entre as *threads*, o que resulta em um maior rendimento quando é necessário um poder computacional maior.

Nota-se também nos gráficos 1 e 2 que, a partir da matriz de ordem 2048x2048, o algoritmo distribuído passa a obter resultados superiores em relação aos outros, ou seja, tempo de respostas melhores quando comparado aos algoritmos sequencial e concorrente. Como mencionado no referencial, o algoritmo distribuído utiliza comunicação por meio de mensagens através de uma rede para que mais de uma máquina possa trocar informações. Dado isso, conforme cresce a necessidade de recursos computacionais para realizar o cálculo da multiplicação, o algoritmo distribuído passa a se destacar, já que a tarefa é dividida entre quatro computadores nesse caso, e cada um faz uma parte do cálculo, trazendo assim um resultado superior.

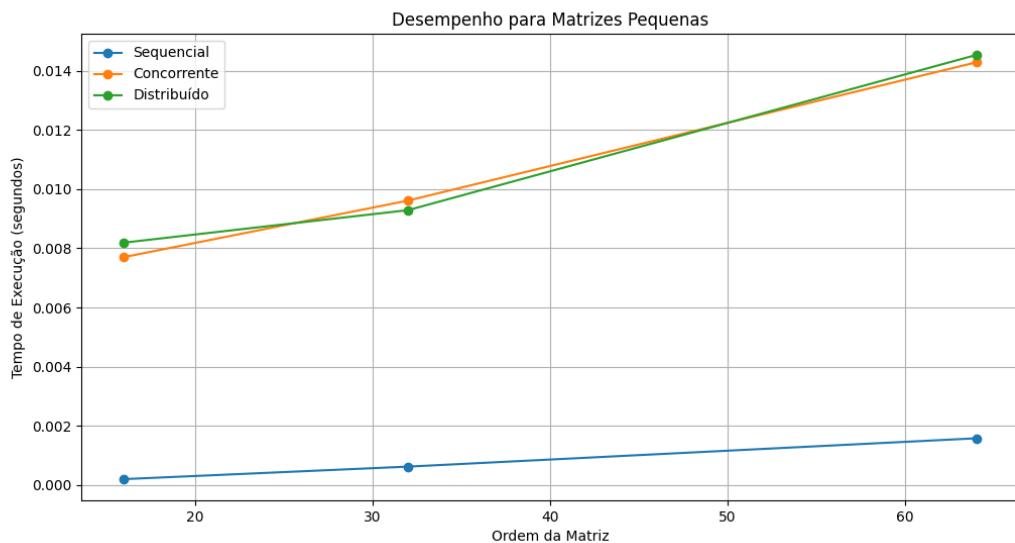
7.1.2 Comparação entre os resultados das matrizes pequenas

Figura 14 – Gráfico 3: gráfico em barras dos resultados das execuções de matrizes pequenas



Fonte: Elaboração própria (2024)

Figura 15 – Gráfico 4: Gráfico em linhas dos resultados das execuções de matrizes pequenas



Fonte: Elaboração própria (2024)

É possível observar nos gráficos uma superioridade no desempenho do algoritmo sequencial em relação aos outros algoritmos. Isso se deve ao fato de que o recurso computacional

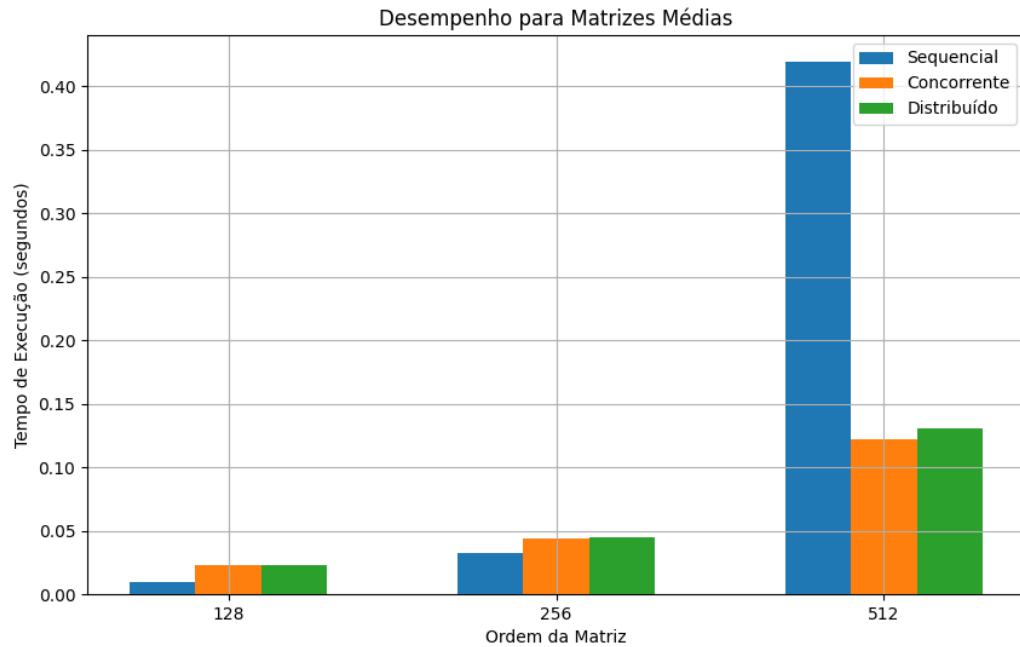
necessário para multiplicar uma matriz pequena é muito baixo, fazendo com que o algoritmo sequencial, que não depende de comunicação e apenas calcula a operação, obtenha um resultado superior aos demais. Para que o algoritmo concorrente realize esse cálculo, é necessária a criação de *threads* para sua execução. Para que as *threads* se comuniquem, existe uma abordagem de memória compartilhada, fazendo com que haja uma sobrecarga na comunicação. O tempo de compartilhamento de recursos entre as *threads*, somado ao tempo que é necessário esperar até que todas as *threads* tenham terminado sua parte do cálculo, é superior ao tempo de realização dos cálculos, resultando assim em um tempo superior ao algoritmo sequencial.

A respeito do algoritmo distribuído, é possível analisar que, em dois dos casos (matrizes de ordem 16 e 64), seu tempo de resposta foi superior aos demais, ou seja, foi o mais lento dos três. Esse resultado ocorre pela necessidade de comunicação entre as máquinas. Para que cada computador calcule parte da matriz, é necessário que haja uma troca de mensagens entre o servidor e as máquinas. O tempo desse processo é estabelecido pela comunicação através de uma rede cabeada, no caso deste experimento.

Para a matriz 32x32, o algoritmo distribuído teve uma execução mais rápida em relação ao concorrente. Esse resultado ocorre porque o algoritmo concorrente considera a média das execuções em máquinas com especificações diferentes, como mencionado no planejamento, resultando em um intervalo maior nos resultados em comparação ao sistema distribuído.

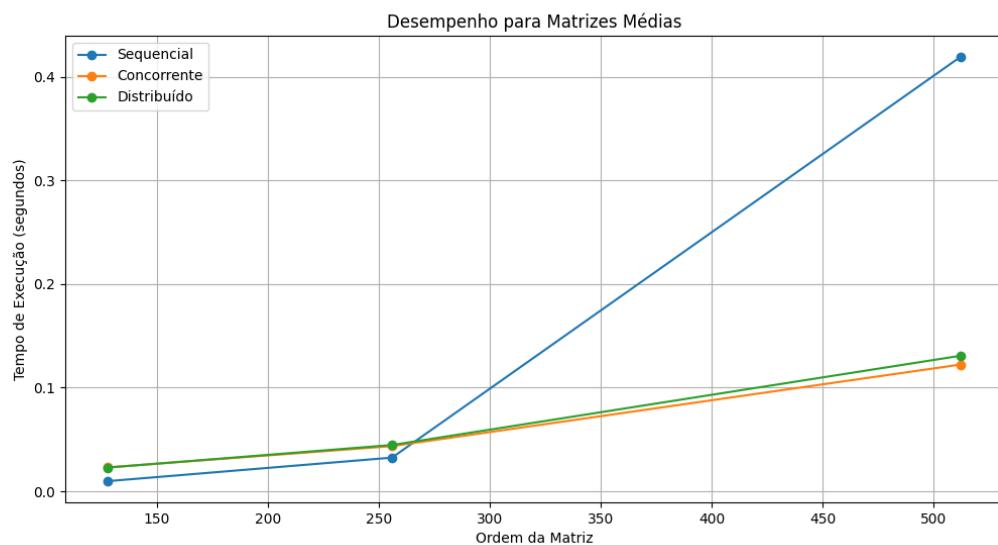
7.1.3 Comparação entre os resultados das matrizes médias

Figura 16 – Gráfico 5: Gráfico em barras dos resultados das execuções de matrizes médias



Fonte: Elaboração própria (2024)

Figura 17 – Gráfico 6: Gráfico em linhas dos resultados das execuções de matrizes média



Fonte: Elaboração própria (2024)

Semelhantemente às matrizes pequenas, para matrizes médias de ordens 128x128 e 256x256, o desempenho é maior para o algoritmo sequencial, ou seja, o algoritmo sequencial foi

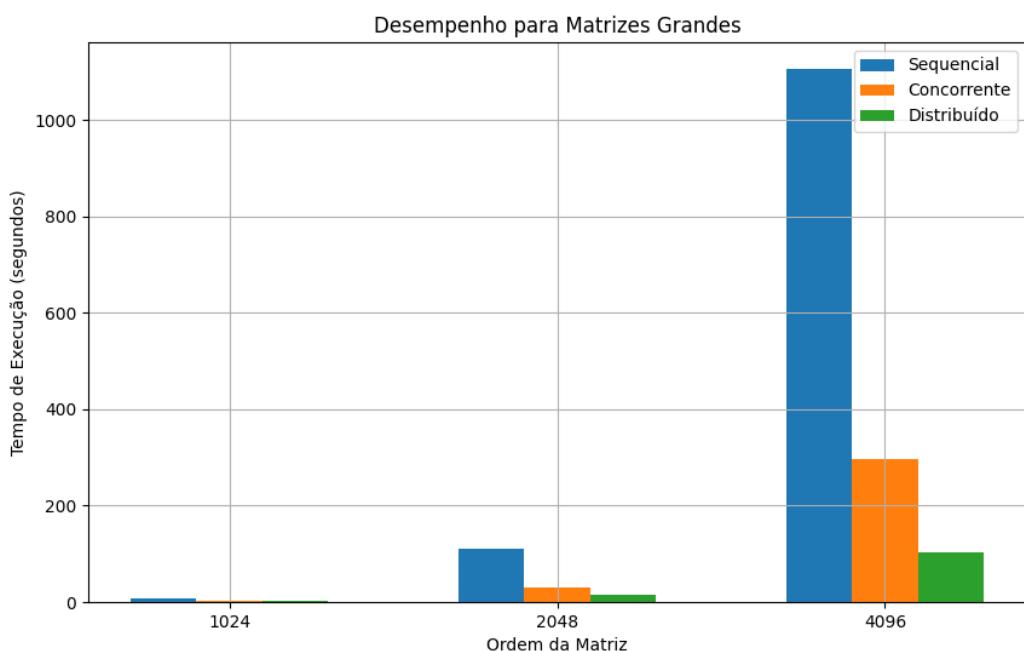
mais rápido que os demais. Entretanto, é possível perceber que a diferença entre os resultados se tornou menor, pois os recursos computacionais estão começando a se tornar um fator crítico para o desempenho. Em outras palavras, calcular essas matrizes está requerendo um desempenho maior das máquinas, fazendo com que o algoritmo sequencial comece a apresentar limitações.

Analizando o gráfico em barras na Figura 16, é possível notar que, na matriz de ordem 512x512, há uma mudança nos tempos de resposta. O algoritmo concorrente se torna aquele com o melhor desempenho. Isso ocorre devido ao paralelismo das *threads*: o tempo de comunicação entre as *threads* se torna menor do que o recurso computacional necessário para o cálculo dos elementos da matriz resultante, fazendo com que a entrega do cálculo com a abordagem de divisão do trabalho para as *threads*, somado ao tempo da comunicação entre elas, seja menor que o tempo necessário para o cálculo da matriz resultante efetuado de forma sequencial, ou seja, utilizando o algoritmo sequencial.

É possível notar também que o algoritmo distribuído apresentou um desempenho superior em relação ao sequencial, ou seja, o algoritmo distribuídos foi mais rápido em comparação ao algoritmo sequencial para matriz de ordem 512. De maneira similar a abordagem anterior, o tempo de troca de mensagens entre as máquinas, somado ao cálculo, foi inferior ao da abordagem sequencial devido ao aumento de processamento necessário para realizar a multiplicação.

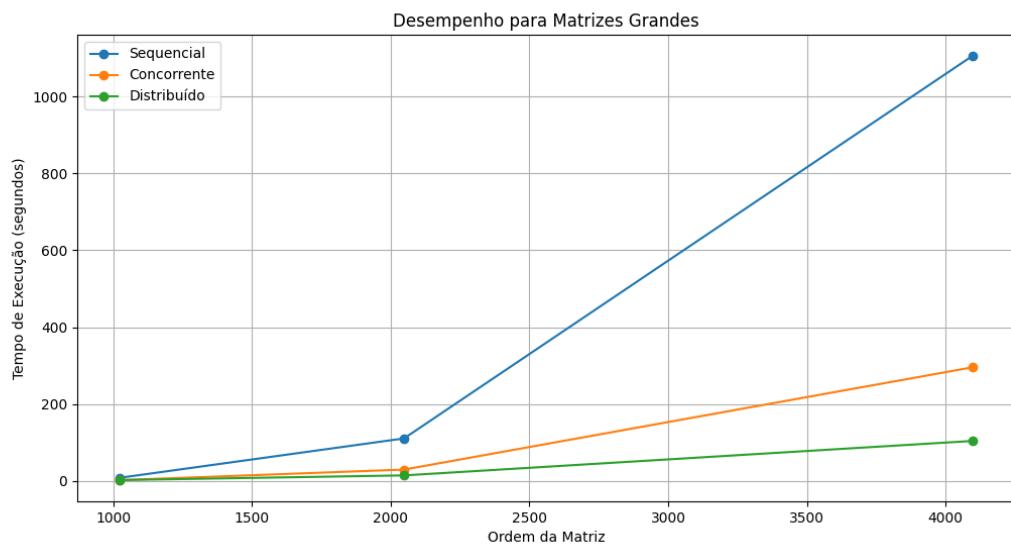
7.1.4 Comparação entre os resultados das matrizes grandes

Figura 18 – Gráfico 7: Gráfico em barras dos resultados das execuções de matrizes grandes



Fonte: Elaboração própria (2024)

Figura 19 – Gráfico 8: Gráfico em linhas dos resultados das execuções de matrizes grandes

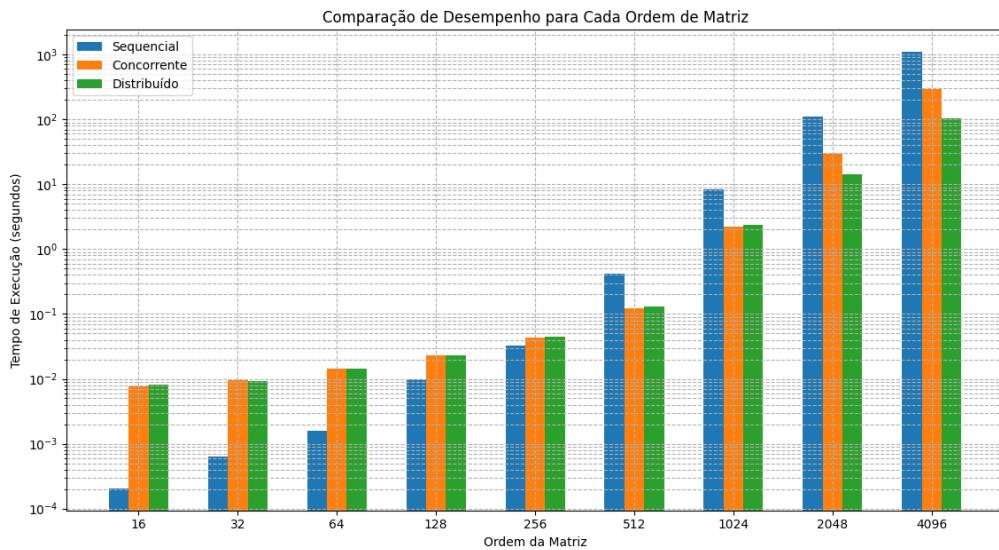


Fonte: Elaboração própria (2024)

Conforme ilustrado no gráfico em linhas presente na Figura 19, nota-se que a diferença entre a performance dos algoritmos passa a crescer de forma significativa. Esses resultados mostram que, para realizar a multiplicação entre matrizes superiores a 2048x2048, a capacidade de processamento passa a ser um fator essencial para efetuar o cálculo. Isso é evidente quando o desempenho superior do algoritmo distribuído se destaca, utilizando os recursos de cada máquina separadamente para calcular e obter o resultado final. Dessa forma, a sobrecarga de comunicação relacionada ao gerenciamento necessário para enviar e receber as matrizes entre vários computadores diferentes, e também ao uso da uma rede cabeadas, torna-se menos preponderante em relação ao tempo de cálculo para matrizes de tamanho grande, diminuindo assim o tempo total de resposta para se obter a matriz resultante, se comparado ao tempo de resposta dos algoritmos sequencial ou concorrente.

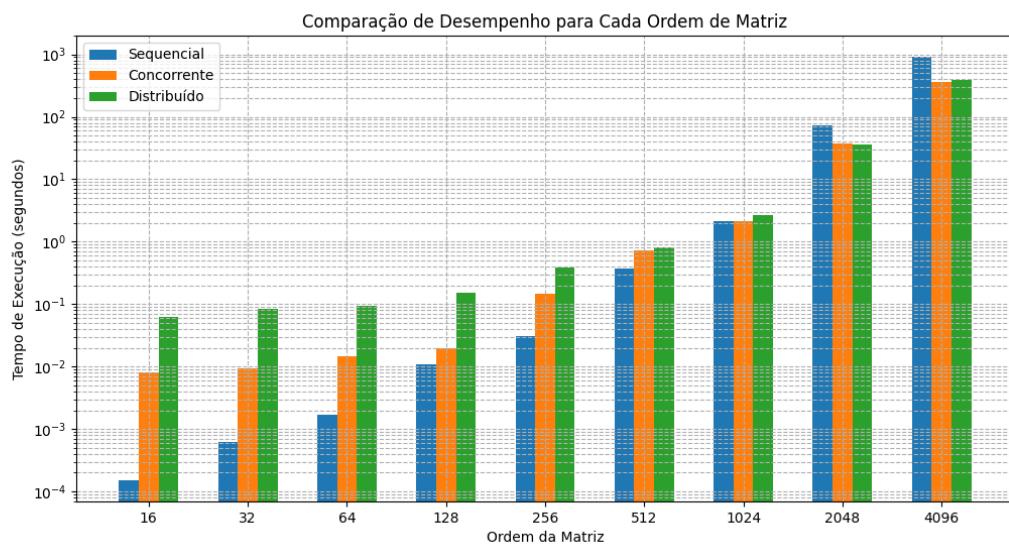
7.1.5 Comparação entre os resultados do laboratório e testes realizados no computador pessoal

Figura 20 – Gráfico 9: Gráfico em barras da comparação de desempenho para cada ordem, realizadas nas máquinas do laboratório



Fonte: Elaboração própria (2024)

Figura 21 – Gráfico 10: Gráfico em barras da comparação de desempenho para cada ordem, realizadas no computador pessoal



Fonte: Elaboração própria (2024)

Comparando os resultados dos testes realizados no laboratório e no computador pessoal é possível identificar algumas diferenças. É notório a diferença entre os resultados do algoritmo distribuído, isso ocorre pelo fato do sistema distribuído no computador pessoal estar simulando um sistema distribuído utilizando terminais, o que não possibilitou um desempenho superior da abordagem distribuída em matrizes maiores, como é no caso dos testes no laboratório que foram coletados de um sistema distribuído real.



CONCLUSÃO

Neste trabalho, foi apresentada a implementação de três algoritmos que realizam a tarefa de multiplicar matrizes, uma tarefa comum na matemática, mas também utilizada em outras áreas, como economia e processamento de linguagem natural voltado à computação. Dada a importância deste cálculo, este trabalho teve como objetivo implementar três versões diferentes de um algoritmo para realizar essa operação: sequencial, concorrente e distribuída, além de fazer uma análise detalhada dos resultados das execuções realizadas.

Foi feito um planejamento, definindo o escopo do projeto e as etapas que seriam seguidas desde o estudo até o desenvolvimento dos algoritmos. Após os estudos necessários e a conclusão do desenvolvimento dos algoritmos, foi iniciada uma fase de testes para validar os algoritmos e começar a coletar os resultados.

O processo de coleta de dados foi realizado no Instituto Federal de Educação, Ciência e Tecnologia Baiano Campus Guanambi, no laboratório de informática 1, localizado no pavilhão do curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, onde cada algoritmo foi executado para realizar o cálculo da multiplicação de matrizes de tamanhos variados. Posteriormente, com todos os dados coletados, foi feita a análise dos resultados.

Os resultados mostraram que, para matrizes de tamanhos pequenos, o algoritmo mais indicado seria o sequencial, já que os cálculos são simples e ele executa em milissegundos por não exigir nenhum tipo de comunicação, como nos outros dois algoritmos.

No entanto, em se tratando de matrizes um pouco maiores, como a matriz de tamanho 512x512, o algoritmo sequencial já deixa de ser o indicado, já que a complexidade do cálculo aumenta, exigindo mais recursos computacionais. Para esses casos de matrizes médias, o desempenho do algoritmo concorrente em relação ao distribuído foi semelhante, mas o concorrente apresentou resultados mais rápidos, fazendo com que ele seja indicado nessas situações, pois não necessita de máquinas conectadas em rede para sua execução.

Por fim, para as matrizes maiores, o desempenho do algoritmo distribuído se destacou pelo fato de conseguir calcular matrizes grandes mais rapidamente em relação aos outros, devido ao seu alto poder computacional, já que a tarefa é dividida entre diversas máquinas.

Dado isso, é possível concluir que, levando em conta os testes realizados neste trabalho, cada algoritmo possui um cenário de aplicação em que se destaca. O algoritmo sequencial é mais eficiente para matrizes pequenas, o concorrente se torna mais adequado em matrizes médias, enquanto o distribuído se destaca em matrizes grandes. Assim, para a escolha do algoritmo com melhor desempenho possível deve ser considerado o tamanho da matriz e recursos computacionais disponíveis.

8.1 Trabalhos Futuros

Existem muitos fatores que influenciam o tempo de resposta do algoritmo. Como trabalhos futuros, pode-se considerar a possibilidade de realizar testes considerando alguns desses fatores, como as especificações das máquinas. Uma alternativa é realizar os testes com máquinas de maior capacidade computacional, buscando entender as diferenças, caso existam. Também é possível realizar os testes com máquinas de mesmas especificações, já que durante o desenvolvimento deste trabalho não foi possível encontrar cinco máquinas com as mesmas configurações.

Outra atualização possível seria alterar variáveis como a quantidade de nós utilizados. Foram utilizados quatro nós para a realização dos cálculos; executar os testes considerando mais nós é uma alternativa. Além dessa variável, é possível também alterar parâmetros como a quantidade de *threads* utilizadas.

Outra alternativa interessante seria realizar testes considerando o algoritmo distribuído porém, variando o meio físico de comunicação entre as máquinas. Neste trabalho as máquinas estavam ligadas por meio de uma rede cabeada com cabos CAT 5e, mas acredita-se que resultados ainda melhores poderiam ser obtidos, se a rede fosse com cabos CAT 6. Da mesma forma, acredita-se que seria possível obter resultados piores caso as máquinas se comunicassem via rede Wi-Fi ao invés de se utilizar cabos de par trançado.

Realizar mais testes considerando essas alternativas enriqueceria o trabalho, tornando-o mais detalhado e permitindo acompanhar o desempenho dos algoritmos em cada um desses cenários.

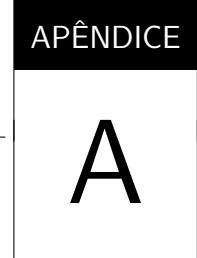
Outro ponto importante seria a revisão dos algoritmos, procurando melhorá-los e identificando gargalos de comunicação que poderiam ser resolvidos, melhorando ainda mais o seu desempenho. Todas essas alterações acrescentariam valor ao trabalho.

REFERÊNCIAS

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 3rd. ed. [S.l.]: MIT Press, 2009. Citado na página 23.
- COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Distributed Systems: Concepts and Design**. 5th. ed. [S.l.]: Pearson, 2011. Citado nas páginas 24, 25, 28, 29, 30 e 31.
- Eclipse Foundation. **Eclipse Documentation**. 2023. <<https://help.eclipse.org/2023-03/index.jsp>>. Acessado em 22 de maio de 2023. Citado na página 36.
- GOLUB, G. H.; LOAN, C. F. V. **Matrix Computations**. 3rd. ed. Baltimore, MD: Johns Hopkins University Press, 1996. Citado nas páginas 15 e 22.
- GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais**. [S.l.]: Editora Blucher, 2000. Citado nas páginas 15 e 18.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <<http://www.deeplearningbook.org>>. Citado na página 15.
- LIPSCHUTZ, S.; ABELLANAS, L.; ONTALBA, C. M. **Álgebra lineal**. [S.l.]: McGraw-Hill, 1992. v. 2. Citado na página 20.
- MACHADO, G. J. Álgebra linear e geometria analítica. 2005. Citado nas páginas 19, 20 e 21.
- Oracle Corporation. **Java SE Documentation**. 2023. Online. Acesso em 4 de maio de 2023. Disponível em: <<https://docs.oracle.com/en/java/javase/17/index.html>>. Citado nas páginas 26, 27, 28 e 30.
- PACHECO, A. G. C. Multiplicação de matrizes: uma comparação entre as abordagens sequencial (cpu) e paralela (gpu). 2019. Citado na página 23.
- PRODANOV, C. C.; FREITAS, E. C. d. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico**. 2. ed. Novo Hamburgo - Rio Grande do Sul: Editora Feevale, 2013. Citado nas páginas 32 e 33.
- RAYNAL, M. **Concurrent Programming: Algorithms, Principles, and Foundations**. 2. ed. [S.l.]: Springer Science & Business Media, 2013. Citado na página 24.
- ROSS, S. M. **Introduction to Probability Models**. [S.l.: s.n.], 2019. ISBN 0128143460. Citado na página 22.
- SANTOS, R. J. **Algebra Linear e Aplicações**. Belo Horizonte: Imprensa Universitaria da UFMG, 2006. Copyright © 2006 by Reginaldo de Jesus Santos. ISBN 85-7470-017-7. Citado na página 21.
- STRANG, G. **Introduction to Linear Algebra**. 4th. ed. Wellesley, Massachusetts: Wellesley-Cambridge Press, 2016. Citado nas páginas 19 e 20.

STÜLP, V. J.; FOCHEZATTO, A. A evolução das disparidades regionais no rio grande do sul: uma aplicação de matrizes de markov. **Nova economia (Belo Horizonte, Brazil)**, Universidade Federal de Minas Gerais, v. 14, n. 1, 2009. ISSN 0103-6351. Citado nas páginas 18 e 21.

TARANTINO, G.; MONICA, S.; BERGENTI, F. A probabilistic matrix factorization algorithm for approximation of sparse matrices in natural language processing. **ICT Express**, v. 4, n. 2, p. 87–90, 2018. ISSN 2405-9595. Artificial Intelligence and Machine Learning Approaches to Communication. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2405959518300584>>. Citado nas páginas 18 e 22.



TABELAS DE TEMPOS DE EXECUÇÃO DOS ALGORITMOS

A.1 Resultados Sequenciais

São apresentados os resultados do tempo de resposta do algoritmo sequencial, executado nas cinco máquinas do laboratório usadas no experimento, além dos resultados no computador pessoal, em forma de tabela. É exibida também uma tabela com a média das execuções das cinco máquinas.

A.1.1 Máquina : 227401-A

Tabela 9 – Tempos de execução no algoritmo sequencial realizados na máquina 227401-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.000188	0.000190	0.000185	0.000181	0.000185	0.000191	0.000186	0.000186	0.000185	0.000194	0.000187
32	0.000549	0.000581	0.002140	0.000601	0.000546	0.000553	0.000741	0.000570	0.000574	0.000563	0.000742
64	0.001678	0.001733	0.001645	0.001690	0.001677	0.001693	0.001723	0.001667	0.001649	0.001699	0.001685
128	0.009233	0.009443	0.009257	0.009229	0.009442	0.009507	0.009307	0.009207	0.009266	0.009261	0.009315
256	0.035661	0.029354	0.029999	0.030524	0.035323	0.029434	0.029547	0.030914	0.030117	0.030057	0.031093
512	0.423612	0.450361	0.269678	0.441279	0.455216	0.451776	0.452409	0.459945	0.269451	0.466051	0.413978
1024	7.956108	7.870537	7.975054	8.184645	7.867622	8.017241	7.670957	7.920296	7.966854	7.737266	7.916658
2048	105.101201	104.164204	107.089405	104.090193	102.519010	106.375019	106.244412	103.700274	102.485654	107.780923	104.955029
4096	1062.617031	1049.057218	1054.459721	1051.057874	1056.918068	1053.993395	1055.276444	1051.578361	1053.732015	1045.065059	1053.375519

Fonte: Elaboração própria (2024)

A.1.2 Máquina : 227402-A

Tabela 10 – Tempos de execução no algoritmo sequencial realizados na máquina 227402-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.000196	0.000185	0.000184	0.000187	0.000181	0.000186	0.000228	0.000186	0.000185	0.000198	0.000192
32	0.000567	0.000534	0.000595	0.000553	0.000555	0.000562	0.000559	0.000555	0.000590	0.000544	0.000561
64	0.001684	0.001848	0.001649	0.001666	0.001655	0.001679	0.001679	0.001621	0.001709	0.001719	0.001691
128	0.009217	0.009300	0.009441	0.009365	0.009670	0.010389	0.009312	0.009188	0.009366	0.009365	0.009461
256	0.030151	0.029997	0.030410	0.030000	0.034809	0.031019	0.029981	0.029713	0.029260	0.029667	0.030501
512	0.404504	0.267172	0.454602	0.444724	0.425165	0.268143	0.454866	0.268099	0.267075	0.370025	0.362438
1024	7.794020	7.781589	7.905433	8.116236	7.725614	7.927236	7.801223	7.791587	7.985800	7.981984	7.881072
2048	103.298111	107.651716	106.013305	106.753801	103.782570	105.341369	106.684761	103.145705	106.290704	104.256143	105.321819
4096	1053.255155	1042.939549	1062.089279	1043.096754	1056.369853	1046.256621	1055.492211	1043.756685	1056.150241	1051.007229	1051.041358

Fonte: Elaboração própria (2024)

A.1.3 Máquina : 227405-A

Tabela 11 – Tempos de execução no algoritmo sequencial realizados na máquina 227405-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.000186	0.000193	0.000186	0.000224	0.000188	0.000231	0.000190	0.000186	0.000187	0.000222	0.000199
32	0.000505	0.000548	0.000571	0.000599	0.000564	0.000553	0.000594	0.000545	0.000539	0.000538	0.000556
64	0.001700	0.001753	0.001677	0.001695	0.001745	0.001644	0.001951	0.003412	0.002649	0.001699	0.001993
128	0.009356	0.009314	0.009780	0.009440	0.009313	0.009100	0.009214	0.009309	0.009460	0.009273	0.009356
256	0.030389	0.032784	0.030161	0.030205	0.029444	0.029377	0.030246	0.030096	0.035409	0.030258	0.030837
512	0.428394	0.452131	0.460062	0.371139	0.267146	0.458241	0.452035	0.443921	0.266317	0.458408	0.405779
1024	7.922557	8.080675	7.892446	7.955665	7.729390	8.001485	7.966955	7.863672	7.852320	7.976756	7.924192
2048	106.713025	107.693105	104.060881	106.744172	103.712483	106.712088	107.091890	107.645475	107.552315	107.007116	106.493255
4096	1050.869740	1048.338982	1055.226010	1057.238423	1064.470351	1061.986894	1049.024215	1045.848974	1053.422337	1042.146840	1052.857276

Fonte: Elaboração própria (2024)

A.1.4 Máquina : 227407-A

Tabela 12 – Tempos de execução no algoritmo sequencial realizados na máquina 227407-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.000209	0.000219	0.000210	0.000235	0.000209	0.000215	0.000209	0.000323	0.000209	0.000226	0.000226
32	0.000633	0.000593	0.000621	0.000623	0.000699	0.000602	0.000613	0.000641	0.000614	0.000651	0.000629
64	0.001887	0.001934	0.001885	0.001895	0.001920	0.001886	0.001905	0.001880	0.001893	0.001965	0.001905
128	0.010464	0.010359	0.010457	0.010365	0.010226	0.010397	0.010408	0.010404	0.010605	0.010300	0.010398
256	0.033356	0.033414	0.039762	0.039707	0.034092	0.033333	0.033849	0.034055	0.034271	0.033752	0.034959
512	0.500294	0.499343	0.300633	0.504218	0.499919	0.498057	0.300924	0.487365	0.499831	0.500310	0.459089
1024	9.319931	8.642692	8.968473	9.047581	8.925006	9.020275	9.080440	9.018196	8.904046	9.163565	9.009021
2048	120.085042	119.252454	118.662473	114.267106	118.337811	117.883448	118.827430	117.967770	115.308055	116.388938	117.698053
4096	1180.954262	1189.445035	1181.239966	1185.760173	1170.371598	1185.020606	1180.477001	1181.753748	1191.972887	1190.123570	1183.711885

Fonte: Elaboração própria (2024)

A.1.5 Máquina : 227410-A

Tabela 13 – Tempos de execução no algoritmo sequencial realizados na máquina 227410-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.000205	0.000209	0.000209	0.000220	0.000254	0.000208	0.000216	0.000209	0.000218	0.000236	0.000218
32	0.000592	0.000627	0.000643	0.000631	0.000645	0.000614	0.000642	0.000801	0.000630	0.000613	0.000644
64	0.001901	0.001964	0.001910	0.001902	0.001904	0.001875	0.001880	0.001857	0.001862	0.001901	0.001896
128	0.010618	0.010419	0.010464	0.010373	0.010342	0.010413	0.010317	0.010492	0.010423	0.010329	0.010419
256	0.034066	0.039444	0.039906	0.033303	0.034012	0.034219	0.033792	0.033386	0.034169	0.033945	0.035024
512	0.487702	0.499303	0.472669	0.500220	0.300540	0.494106	0.496227	0.499696	0.300326	0.499383	0.455017
1024	8.987731	8.939986	8.973739	8.584972	8.986103	9.018872	9.042265	8.970129	8.985293	8.769657	8.925875
2048	117.067277	119.023395	119.794004	120.302666	119.297629	119.030057	120.657157	119.647237	116.337970	119.173346	119.033074
4096	1185.209390	1186.137090	1189.955315	1186.487262	1188.864825	1184.664958	1190.454043	1186.210888	1185.319496	1185.901376	1186.920464

Fonte: Elaboração própria (2024)

A.1.6 Média dos tempos de execução do algoritmo sequencial nas cinco máquinas

Tabela 14 – Média feita entre os tempos de execução do algoritmo sequencial nas cinco máquinas

Ordem da Matriz	Tempo de Execução (segundos)
16x16	0.000203698360
32x32	0.000626311500
64x64	0.001583494800
128x128	0.009789901220
256x256	0.032482742860
512x512	0.419260253420
1024x1024	8.331363535320
2048x2048	110.700245988960
4096x4096	1105.581300322400

Fonte: Elaboração própria (2024)

A.1.7 Computador Pessoal

Tabela 15 – Tempos de execução no algoritmo sequencial realizados no computador pessoal

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.000196	0.000179	0.000152	0.000124	0.000129	0.000151	0.000157	0.000128	0.000129	0.000167	0.000151
32	0.000591	0.000658	0.000738	0.000602	0.000620	0.000569	0.000603	0.000586	0.000621	0.000582	0.000617
64	0.001548	0.001491	0.002328	0.001672	0.001673	0.001594	0.001655	0.001683	0.001577	0.001728	0.001695
128	0.013112	0.010764	0.010167	0.009960	0.010909	0.010170	0.013029	0.010498	0.010211	0.010508	0.010933
256	0.041159	0.033325	0.035670	0.027086	0.031407	0.031800	0.025368	0.023781	0.030871	0.030224	0.031069
512	0.352250	0.331809	0.358081	0.524962	0.389650	0.341297	0.332296	0.335320	0.358913	0.345319	0.366990
1024	1.886453	1.856947	2.157558	3.161233	2.158639	2.201082	2.027662	2.141200	2.009776	2.008754	2.160930
2048	74.301585	71.782362	76.487786	71.399878	76.096915	71.054415	69.308298	71.112510	72.722081	71.562120	72.582795
4096	959.639683	919.167372	898.857192	893.076797	891.828135	892.282819	877.015038	926.355604	956.104144	950.418227	916.474501

Fonte: Elaboração própria (2024)

A.2 Resultados Concorrentes

Apresenta os resultados do tempo de resposta do algoritmo concorrente, sendo executado nas cinco máquinas do laboratório usadas no experimento, além dos resultados no computador pessoal, em forma de tabela. É exibida também uma tabela com a média das execuções das cinco máquinas.

A.2.1 Máquina : 227401-A

Tabela 16 – Tempos de execução no algoritmo concorrente realizados na máquina 227401-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.007394	0.007199	0.007497	0.007322	0.007573	0.007187	0.007210	0.007424	0.007492	0.007320	0.00736178
32	0.011205	0.008873	0.013528	0.011658	0.010629	0.008070	0.008084	0.007851	0.007717	0.010970	0.00985847
64	0.014522	0.014235	0.014350	0.013816	0.013140	0.013529	0.015418	0.013021	0.012221	0.015619	0.01398715
128	0.021538	0.019395	0.016618	0.020966	0.017558	0.018035	0.017512	0.018137	0.020754	0.019460	0.01899740
256	0.050317	0.040392	0.054059	0.037743	0.045447	0.044539	0.040985	0.042506	0.040490	0.038840	0.04353168
512	0.117389	0.119245	0.116845	0.117346	0.116563	0.122528	0.121618	0.114372	0.112692	0.109802	0.11684004
1024	2.088700	2.096595	2.080683	2.097056	2.115985	2.106242	2.112651	2.088397	2.089848	2.080701	2.09568595
2048	27.809581	27.923552	27.822593	28.478327	27.864635	28.387900	28.561053	28.418392	28.601207	27.650977	28.15182177
4096	282.817201	285.976245	281.741275	284.275056	283.215172	279.429614	279.639283	281.475025	284.107559	283.688808	282.63652383

Fonte: Elaboração própria (2024)

A.2.2 Máquina : 227402-A

Tabela 17 – Tempos de execução no algoritmo concorrente realizados na máquina 227402-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.007200	0.007290	0.007178	0.007418	0.007245	0.007345	0.007255	0.007932	0.007345	0.007259	0.0073467554
32	0.007870	0.010284	0.010710	0.008699	0.012169	0.010467	0.011230	0.008297	0.011348	0.008095	0.0099168517
64	0.016754	0.014027	0.013640	0.012780	0.013094	0.012261	0.016613	0.012572	0.012883	0.013505	0.0138129408
128	0.024713	0.024090	0.018982	0.018967	0.026108	0.022724	0.019149	0.061113	0.022356	0.019675	0.0257875087
256	0.043032	0.035365	0.057083	0.040261	0.040703	0.050707	0.040024	0.041980	0.042008	0.056354	0.0447516172
512	0.115926	0.124874	0.119830	0.113880	0.113115	0.112500	0.121732	0.111771	0.132499	0.127805	0.1193930673
1024	2.081862	2.084690	2.074680	2.075016	2.077618	2.092471	2.084976	2.069447	2.093649	2.073343	2.0807752075
2048	27.511555	28.499010	27.672247	27.472103	28.445847	27.474424	27.570090	27.580895	27.415132	28.466752	27.8108054319
4096	282.687054	281.600244	280.189281	278.672668	280.942880	280.470305	279.400055	278.866776	281.048516	280.754270	280.4632048726

Fonte: Elaboração própria (2024)

A.2.3 Máquina : 227405-A

Tabela 18 – Tempos de execução no algoritmo concorrente realizados na máquina 227405-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.007259	0.007203	0.007347	0.007228	0.007408	0.007590	0.007295	0.007441	0.007449	0.007119	0.0073338668
32	0.008263	0.007919	0.010000	0.008612	0.008409	0.008220	0.008344	0.008164	0.009221	0.010673	0.008782677
64	0.015928	0.017034	0.011690	0.014127	0.014050	0.012583	0.012583	0.011722	0.012790	0.021399	0.0143905445
128	0.032897	0.019665	0.019909	0.017743	0.027754	0.021587	0.024745	0.020462	0.020697	0.020141	0.0225599146
256	0.039872	0.041108	0.041966	0.041791	0.036780	0.037103	0.036983	0.043139	0.045234	0.042074	0.0406050443
512	0.110927	0.106713	0.110030	0.128162	0.105938	0.117011	0.120632	0.105005	0.128172	0.121352	0.1153941118
1024	2.075561	2.090760	2.068519	2.088285	2.097832	2.103412	2.078746	2.089756	2.087638	2.104547	2.0885056862
2048	27.839228	28.354601	28.333758	28.541480	28.614456	28.655445	28.492547	28.572000	28.466214	28.460544	28.4330271016
4096	280.821936	281.457672	280.243537	286.836382	281.089672	279.318154	282.183242	280.902192	283.022135	282.539608	281.8414529816

Fonte: Elaboração própria (2024)

A.2.4 Máquina : 227407-A

Tabela 19 – Tempos de execução no algoritmo concorrente realizados na máquina 227407-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.008083	0.008240	0.008355	0.008302	0.008387	0.008158	0.008132	0.008374	0.008365	0.008433	0.0082828787
32	0.008999	0.011501	0.009585	0.009890	0.008806	0.010048	0.012538	0.009085	0.008917	0.012828	0.0102196316
64	0.014643	0.015642	0.014711	0.014906	0.013646	0.014011	0.015779	0.014406	0.014502	0.014724	0.0146969463
128	0.019923	0.027931	0.021628	0.025224	0.037793	0.028767	0.019028	0.020252	0.027910	0.021901	0.025035706
256	0.042949	0.046614	0.044553	0.043995	0.044282	0.045512	0.044356	0.048839	0.042903	0.039877	0.0443879526
512	0.131144	0.121892	0.120116	0.122929	0.140983	0.129257	0.135698	0.125941	0.134435	0.128067	0.1290462045
1024	2.377011	2.366576	2.372225	2.357518	2.359410	2.357976	2.354912	2.359879	2.379691	2.367402	2.365260039
2048	31.945834	31.448621	31.887728	31.137094	31.220417	31.982870	31.902762	31.846334	32.158984	32.041526	31.7572170395
4096	312.722788	316.029051	314.194466	317.627192	315.202272	318.803401	316.470928	316.329320	315.448796	314.717469	315.7545682678

Fonte: Elaboração própria (2024)

A.2.5 Máquina : 227410-A

Tabela 20 – Tempos de execução no algoritmo concorrente realizados na máquina 227410-A

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.008059	0.008426	0.008295	0.008404	0.008204	0.008041	0.007983	0.008209	0.008183	0.008119	0.0081925046
32	0.008699	0.009097	0.008962	0.008956	0.009136	0.009118	0.008843	0.011699	0.008992	0.009403	0.0092903459
64	0.013314	0.014467	0.014463	0.012886	0.016663	0.013123	0.013306	0.014735	0.014627	0.017752	0.0145335877
128	0.022109	0.025627	0.021406	0.019021	0.022406	0.028453	0.030168	0.018806	0.021691	0.019412	0.0229100104
256	0.039615	0.039728	0.040443	0.044668	0.046033	0.045806	0.048739	0.053803	0.043845	0.043930	0.0446611373
512	0.131132	0.122171	0.126555	0.133052	0.131523	0.127367	0.134552	0.128750	0.147136	0.125703	0.1307943171
1024	2.364082	2.356008	2.353698	2.371118	2.364269	2.344013	2.355718	2.352308	2.351914	2.361168	2.3574295037
2048	32.027945	32.061650	32.212958	32.013903	32.101124	32.269688	32.126654	32.159613	32.012827	32.113386	32.1099748529
4096	316.008150	319.439609	316.692803	316.830392	316.275219	316.074360	315.945988	319.616284	314.870999	319.357656	317.1110459949

Fonte: Elaboração própria (2024)

A.2.6 Média dos tempos de execução do algoritmo concorrente nas cinco máquinas

Tabela 21 – Média feita entre os tempos de execução do algoritmo concorrente nas cinco máquinas

Ordem da Matriz	Tempo de Execução (segundos)
16x16	0.007703557100
32x32	0.009613594440
64x64	0.014284234720
128x128	0.023058108280
256x256	0.043587485680
512x512	0.122293549120
1024x1024	2.197531276540
2048x2048	29.652569239720
4096x4096	295.561359188520

Fonte: Elaboração própria (2024)

A.2.7 Computador Pessoal

Tabela 22 – Tempos de execução no algoritmo concorrente realizados no computador pessoal

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.009366	0.007879	0.008312	0.008669	0.008122	0.007588	0.008321	0.007657	0.008369	0.007890	0.00821725
32	0.008695	0.009045	0.008628	0.009111	0.008702	0.009845	0.009395	0.011090	0.010132	0.010071	0.00947127
64	0.014307	0.014804	0.013838	0.013849	0.014543	0.018032	0.014356	0.014842	0.014160	0.014136	0.01468662
128	0.028204	0.018503	0.017922	0.018365	0.018458	0.021392	0.018267	0.017614	0.018089	0.017843	0.01946569
256	0.032006	0.030132	0.029330	0.030549	0.028022	0.489249	0.417367	0.031374	0.030085	0.343908	0.14620228
512	0.686384	0.711508	0.799696	0.662532	0.854704	0.809291	0.752327	0.675096	0.659734	0.725603	0.73368734
1024	1.791029	2.043400	2.026586	2.289286	2.164846	2.376629	1.803290	2.206428	1.994446	2.311962	2.10079017
2048	35.948898	36.606661	38.346364	37.287913	35.601834	40.899910	38.932561	38.167319	38.066338	37.498867	37.73566653
4096	359.851759	367.116959	349.563622	363.938398	365.143566	356.853280	359.519066	343.391215	360.522198	354.524708	358.04247694

Fonte: Elaboração própria (2024)

A.3 Resultados Distribuídos

São apresentados os resultados do tempo de resposta do algoritmo distribuído, sendo executado nas cinco máquinas do laboratório, com uma máquina sendo o nó mestre e as outras quatro máquinas sendo nós trabalhadores dividindo a tarefa da multiplicação. Também são apresentados os resultados no computador pessoal simulando um sistema distribuído com cinco terminais representando os nós.

A.3.1 Resultados no laboratório

Tabela 23 – Tempos de execução do algoritmo distribuído realizado nas cinco máquinas do laboratório

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.052815	0.051042	0.050373	0.051123	0.043996	0.049541	0.049951	0.048971	0.050058	0.049707	0.049652
32	0.059244	0.067372	0.063032	0.059884	0.066522	0.062351	0.067950	0.057532	0.068022	0.057371	0.063178
64	0.069224	0.062161	0.068526	0.063101	0.066047	0.063125	0.055626	0.067826	0.058297	0.065468	0.061391
128	0.091828	0.091258	0.090355	0.091717	0.084092	0.084365	0.091749	0.092048	0.092096	0.086836	0.089936
256	0.183396	0.186442	0.179948	0.191706	0.202021	0.179800	0.181007	0.191543	0.175764	0.177609	0.187679
512	0.563451	0.552598	0.548176	0.558399	0.567557	0.562227	0.559716	0.555956	0.565990	0.554179	0.558825
1024	2.505026	2.513819	2.521139	2.502505	2.518098	2.539098	2.534524	2.527458	2.510787	2.542476	2.514519
2048	14.528018	14.490125	14.498633	14.472971	14.455287	14.429084	14.546657	14.475339	14.460888	14.481310	14.483426
4096	103.666605	105.004526	103.456929	103.541645	104.677071	104.290890	104.667014	104.078932	103.660708	103.884170	104.027055

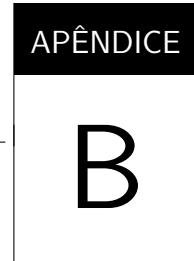
Fonte: Elaboração própria (2024)

A.3.2 Resultados no Computador Pessoal

Tabela 24 – Tempos de execução do algoritmo distribuído no computador pessoal

Ordem	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Execução 6	Execução 7	Execução 8	Execução 9	Execução 10	Média
16	0.052848	0.077904	0.059575	0.070617	0.061238	0.062445	0.069669	0.054437	0.063641	0.065431	0.063597
32	0.093083	0.085989	0.109172	0.074582	0.071834	0.089603	0.093030	0.073595	0.071411	0.072210	0.083451
64	0.096487	0.094880	0.093357	0.090679	0.110088	0.104909	0.084319	0.084796	0.101407	0.095038	0.095596
128	0.123220	0.109009	0.113988	0.106579	0.122887	0.370853	0.105406	0.142491	0.211921	0.130663	0.153702
256	0.256318	0.512311	0.411537	0.796724	0.256721	0.264000	0.238121	0.293512	0.480285	0.309197	0.381873
512	0.645467	0.707750	0.668922	0.839427	0.977499	0.945895	0.956737	0.922102	0.651161	0.838828	0.815379
1024	2.671947	2.652373	2.901273	2.551913	2.492073	2.583196	2.751650	2.285874	2.968430	2.861877	2.672061
2048	31.971551	37.453425	36.829477	37.205576	35.475721	36.210046	35.035239	34.707350	35.587480	40.337431	36.081330
4096	378.664501	383.174537	384.256252	384.060366	386.939854	383.091193	385.727613	387.424413	385.043051	375.060077	383.344186

Fonte: Elaboração própria (2024)



COMANDOS PARA EXECUÇÃO DOS ALGORITMOS

Neste apêndice serão apresentados os comandos para a execução de cada um dos algoritmos desenvolvidos.

B.1 Comando de compilação Java

Antes que os algoritmos sejam iniciados, é necessário usar um comando para que o arquivo Java seja compilado, possibilitando assim a execução do algoritmo. Para isso, deve-se navegar até a pasta onde os algoritmos estão localizados e digitar o comando presente no Código-fonte 13. O código compila todos os arquivos na pasta com a terminação “.java”; também é possível digitar o nome de um arquivo específico, caso seja apenas um.

Código-fonte 13 – Comando para execução do algoritmo sequencial

1 `javac *.java`

B.2 Execução do algoritmo sequencial

Para a execução do algoritmo sequencial, é utilizado o comando descrito no Código-fonte 14. O código é composto pelo comando de execução “java” seguido do nome da classe e o argumento referente a ordem da matriz (para esse exemplo foi usada a ordem 64):

Código-fonte 14 – Comando para execução do algoritmo sequencial

1 `java SequentialMatrixMultiplication 64`

B.3 Execução do algoritmo Concorrente

Para a execução do algoritmo concorrente, utiliza-se o comando descrito no Código-fonte 15. Semelhante ao algoritmo sequencial, o código é composto pelo comando de execução “java”, seguido do nome da classe e do argumento referente à ordem da matriz (64, no exemplo):

Código-fonte 15 – Comando para execução do algoritmo concorrente

```
1   java ConcurrentMatrixMultiplication 64
```

B.4 Execução do algoritmo Distribuído

Para a execução do algoritmo distribuído, inicie o servidor primeiramente utilizando o comando “java”, seguido do nome da classe e da ordem da matriz que deseja calcular. O Código-fonte 16 mostra como o comando deve ser executado.

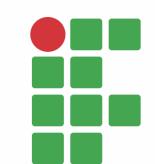
O segundo passo é inicializar os nós trabalhadores utilizando máquinas diferentes, ou terminais diferentes, caso esteja utilizando uma única máquina. A execução dos nós trabalhadores deve seguir o exemplo do Código-fonte 17, utilizando o comando ‘java’ seguido do nome da classe. Como argumentos, devem ser fornecidos o número IP do nó mestre e a porta na qual ela está sendo executado.

Código-fonte 16 – Comando para execução do algoritmo distribuído: Nô Mestre

```
1   java TCPServer 64
```

Código-fonte 17 – Comando para execução do algoritmo distribuído: Nós Trabalhadores

```
1   java TCPCliente 127.0.0.1 1234
```



**INSTITUTO
FEDERAL**
Baiano
Campus
Guanambi