Facens	Sist. Operacionais – CP502	TIN3
	Prof. Marcos Jardini	11/06/2024

Multiplicação de Matrizes com OpenMP

(Open Multi-Processing)

Kevyn Feitosa Rocha	223535
Darlan Henrique de Souza Oliveira	211926

Sorocaba
2024

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

1.1 Teoria da Multiplicação de Matrizes

A multiplicação de matrizes é uma operação fundamental em álgebra linear e possui diversas aplicações em áreas como computação gráfica, processamento de imagens, criptografia e simulações científicas (Golub & Van Loan, 2013). Dadas duas matrizes A (de dimensões m x n) e B (de dimensões n x p), o resultado da multiplicação C = A * B é uma matriz de dimensões m x p, onde cada elemento $C_{[i][j]}$ é calculado pela soma dos produtos dos elementos correspondentes da i-ésima linha de A e da j-ésima coluna de B:

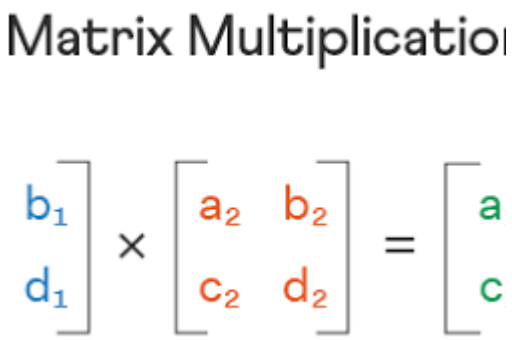

Figura 1 - Multiplicação de Matrix

A complexidade computacional da multiplicação de matrizes tradicional é $O(n^3)$, o que significa que o tempo de execução cresce cubicamente com o tamanho das matrizes. No entanto, existem algoritmos mais eficientes, como o algoritmo de Strassen, que possui complexidade $O(n^{\log 7})$, porém com maior overhead e menor estabilidade numérica (Cormen et al., 2009).

1.2 Benefícios da Paralelização com Threads e Fork

A multiplicação de matrizes é uma operação inerentemente paralelizável, pois o cálculo de cada elemento da matriz resultante é independente dos demais. Isso permite dividir a tarefa em subtarefas menores que podem ser executadas simultaneamente por múltiplas threads ou processos, explorando o paralelismo em nível de thread (multi-threading) ou em nível de processo (multi-processing). (Chandra et al. 2001)

- Threads: Threads são unidades de execução leves que compartilham o mesmo espaço de memória do processo pai. A criação e o gerenciamento de threads são mais simples e possuem menor overhead em comparação com processos.
- Fork: O mecanismo fork cria um processo filho, que é uma cópia exata do processo pai. Os processos filho e pai possuem espaços de memória independentes, o que permite maior isolamento e robustez, mas com maior overhead de criação e comunicação.

A escolha entre threads e fork depende das características da aplicação e do ambiente de execução. Em geral, threads são mais adequadas para tarefas com alta granularidade e comunicação frequente, enquanto fork é mais adequado para tarefas com baixa granularidade e comunicação esporádica.

1.3 OpenMP: Simplificando a Programação Paralela

OpenMP (Open Multi-Processing) é uma API que facilita a programação paralela em C, C++ e Fortran, fornecendo diretivas e funções para paralelizar loops e regiões de código de forma simples e eficiente. O OpenMP não leva em conta os detalhes de criação e sincronização de threads, permitindo que o programador se concentre na lógica da paralelização.

A diretiva *#pragma omp parallel for* é utilizada para paralelizar um loop *for*, dividindo as iterações entre os threads disponíveis. O OpenMP gerencia automaticamente a criação e a distribuição dos threads, bem como a sincronização no final do loop.

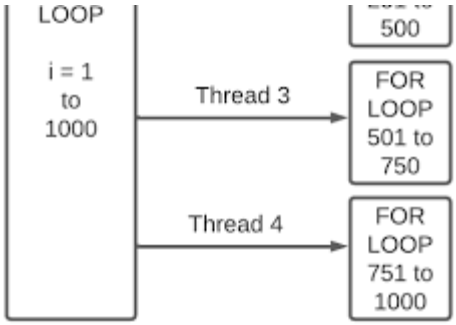

Figura 2 - Diagrama de execução paralela de um loop OpenMP

A Tabela 1 apresenta um comparativo entre threads e fork, destacando suas vantagens e desvantagens.

Tabela 1: Comparativo entre threads e fork.

Características	Threads	Fork
Criação e gerenciamento	Mais simples e leve	Mais complexo e pesado
Espaço de memória	Compartilhado	Independente
Comunicação	Mais fácil e rápida	Mais difícil e lenta
Isolamento e robustez	Menor	Maior
Granularidade de tarefas	Alta	Baixa

1.4 OpenMP Architecture Review Board (ARB)

OpenMP (Open Multi-Processing) é uma API da "*OpenMP Architecture Review Board (ARB)"* que suporta programação multi-thread/multi-processo em C, C++ e Fortran. O exemplo escolhido implementa a multiplicação de matrizes, uma operação comum em diversas áreas da computação, como processamento de imagens, simulações físicas e aprendizado de máquina:

- Processamento de Imagens: Rotação, escala e filtros.
- Simulações Físicas: Modelagem de sistemas dinâmicos.
- Aprendizado de Máquina: Treinamento de redes neurais.

A multiplicação de matrizes é uma operação paralelizável, ou seja, pode ser dividida em tarefas menores que podem ser executadas simultaneamente por múltiplas threads ou processos. Isso permite aproveitar o poder de processamento de CPUs multi-core e sistemas com múltiplos processadores, resultando em um ganho de desempenho significativo em relação à execução sequencial.

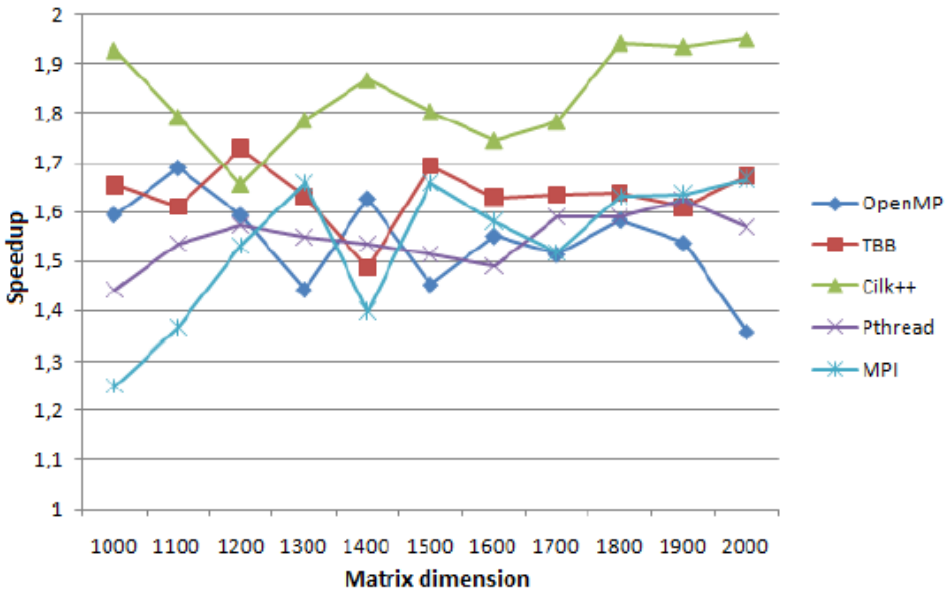

Figura 3 - Aceleração da multiplicação de matrizes com OpenMP conforme threads aumentam

O ganho de desempenho pode ser medido comparando o tempo de execução da multiplicação de matrizes com diferentes números de threads/processos. É esperado que o tempo de execução diminua à medida que o número de threads/processos aumenta, até atingir um ponto de saturação onde o overhead de gerenciamento dos threads/processos supera o ganho de paralelização.

2 Preparação para desenvolvimento

2.1 Ambiente de programação

Para a implementação e testes da multiplicação de matrizes paralela com OpenMP, foi utilizado o ambiente de desenvolvimento Dev C++ em uma máquina virtual Windows com o VirtualBox. Essa abordagem combina a facilidade de uso do Dev C++ com a flexibilidade e o isolamento de uma máquina virtual, proporcionando um ambiente controlado e reprodutível para o desenvolvimento e testes do projeto.

2.1.1 Configuração do VirtualBox e Windows

A máquina virtual Windows foi configurada no VirtualBox e o Dev C++ foi instalado, incluindo o compilador GCC (GNU Compiler Collection) com suporte a OpenMP.

2.1.2 Linguagem de Programação e Versão

A linguagem de programação escolhida para a implementação foi C++, utilizando a versão C++17. Essa versão oferece suporte completo às funcionalidades do OpenMP, incluindo a diretiva *#pragma omp parallel for* para paralelização de loops.

2.1.3 Configuração do OpenMP

O OpenMP não requer a alteração de arquivos de configuração para ajustar o número de threads utilizadas. O número de threads é definido automaticamente com base no número de núcleos de processamento disponíveis ou pode ser controlado pela variável de ambiente OMP_NUM_THREADS.

Para definir o número de threads em 4, por exemplo, basta executar o seguinte comando antes de iniciar o programa:

Bash
export OMP_NUM_THREADS=4

Alternativamente, o número de threads pode ser definido dentro do código-fonte utilizando a função *omp_set_num_threads ()*.

É importante ressaltar que a configuração ideal do número de threads depende das características do hardware e da carga de trabalho, sendo recomendado realizar testes para encontrar o valor que oferece o melhor desempenho.

2.1.4 Mensuração do Tempo de Execução

Para avaliar o desempenho da implementação da multiplicação de matrizes paralela, o tempo de execução é medido utilizando a biblioteca chrono do C++, que oferece ferramentas para trabalhar com tempo. Os principais elementos utilizados são:

high_resolution_clock: Um tipo de relógio que mede o tempo com a maior precisão possível, capturando o instante exato de início e término da execução.

duration: Uma classe que representa um período, permitindo armazenar a diferença entre os instantes de início e término.

duration_cast: Uma função para converter a duração medida em uma unidade específica, como milissegundos, facilitando a análise e comparação dos resultados.

A medição do tempo é realizada marcando o instante de início antes da multiplicação das matrizes, marcando o instante de término após a conclusão da operação e, em seguida, calculando a diferença entre esses instantes. Essa diferença, convertida para milissegundos, representa o tempo total de execução da multiplicação. Ao repetir esse processo para diferentes números de threads, é possível avaliar o impacto da paralelização no desempenho da implementação.

3 Testes e Resultados

Para avaliar o desempenho da multiplicação de matrizes utilizando OpenMP, foram realizados testes variando o número de threads. Os testes foram executados em um contêiner Docker configurado com a imagem GCC 11.2.0 e bibliotecas OpenMP, utilizando uma máquina com um processador multi-core.

3.1.1 Configuração do Ambiente de Teste

O código foi implementado em C++, utilizando a diretiva `#pragma omp parallel for` para paralelizar o loop de multiplicação das matrizes. O tamanho das matrizes utilizadas nos testes foi de 500x500 e 1000x1000.

3.1.2 Resultados

Os tempos de execução para diferentes números de threads foram registrados e são apresentados nas Tabelas 2 e 3. Na figura 4 é possível verificar o código em funcionamento.

Figura 4 - Terminal compilando o programa. (própria autoria, 2024)

Os resultados são ilustrados nos Gráficos 1 e 2.

3.1.3 Análise

Os resultados mostram que o tempo de execução diminui conforme o número de threads aumenta, evidenciando o ganho de desempenho proporcionado pela paralelização. No entanto, é importante notar que o ganho não é linear,

devido ao overhead de gerenciamento de threads e à limitação dos recursos de hardware.

Tabela 2: Computador de Kevyn Rocha.

Quantidade Threads	Matriz de 500	Matriz de 1000
1	2124 ms	26549 ms
2	1099 ms	12860 ms
4	615 ms	6894 ms
8	339 ms	4161 ms
16	241 ms	2897 ms

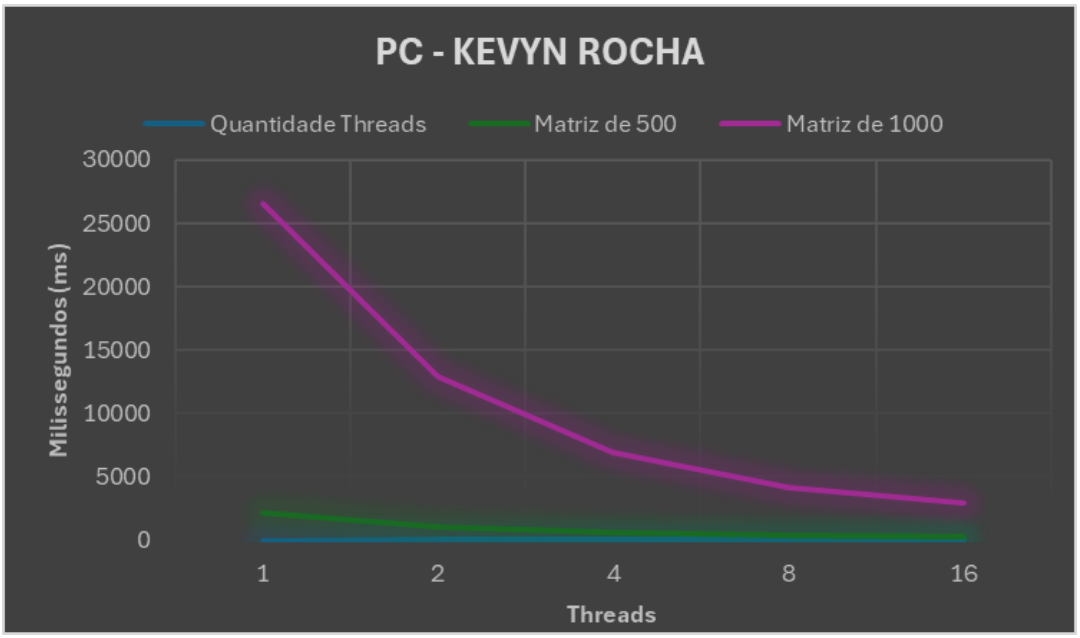

Figura 5 - Gráfico de desempenho ms / thread - Kevyn. (própria autoria, 2024)

Tabela 3: Computador de Darlan Henrique.

Quantidade Threads	Matriz de 500	Matriz de 1000
1	3041 ms	27587 ms
2	1411 ms	14907 ms
4	719 ms	9891 ms
8	560 ms	7291 ms
16	467 ms	4125 ms

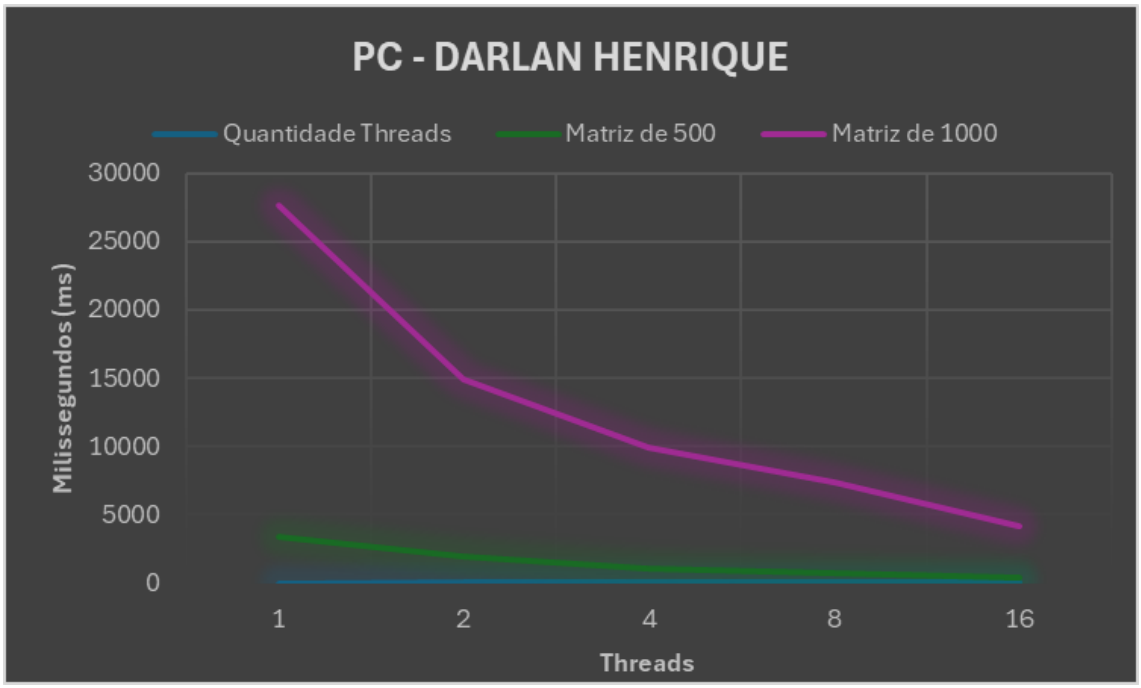

Figura 6 - Gráfico de desempenho ms / thread - Darlan. (própria autoria, 2024)

Conclui-se que a paralelização da multiplicação de matrizes utilizando OpenMP é uma técnica eficaz para melhorar o desempenho em ambientes com múltiplos núcleos de processamento. Os resultados demonstram que o uso de múltiplas threads reduz significativamente o tempo de execução, especialmente para matrizes de maior dimensão. No entanto, é importante observar que o ganho de desempenho não é linear e pode ser limitado pelo overhead de gerenciamento dos threads e pelos recursos de hardware disponíveis.

Para obter mais detalhes sobre a implementação e o código-fonte completo, acesse o repositório do projeto no GitHub:

https://github.com/DarlanHSO/matrix_vs_threads_increase

Este repositório oferece uma visão aprofundada do trabalho realizado e permite a reprodução dos experimentos, contribuindo para o aprendizado e o desenvolvimento de novas soluções para a multiplicação de matrizes paralela.

Referências

Chapman, B., Jost, G., & van der Pas, R. (2008). *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press.

Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., & McDonald, J. (2001). *Parallel Programming in OpenMP*. Morgan Kaufmann.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms (3rd ed.).* MIT Press.

Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations (4th ed.)*. Johns Hopkins University Press.

OpenMP: https://www.openmp.org/ (Acesso em: 18/05/2024)

GCC: https://gcc.gnu.org/ (Acesso em: 19/05/2024)

Docker: https://www.docker.com/ (Acesso em: 18/05/2024)