UEFS

UNIVERSIDADE ESTADUAL
DE FEIRA DE SANTANA

Programa de Pós Graduação em Ciência da Computação - PGCC

ESTRATÉGIAS DE PARALELIZAÇÃO DA INVERSÃO DE MATRIZES UTILIZANDO O ALGORITMO DE GAUSS-JORDAN COM OPENMP

Herlon A. Santos e Thaiany Santana

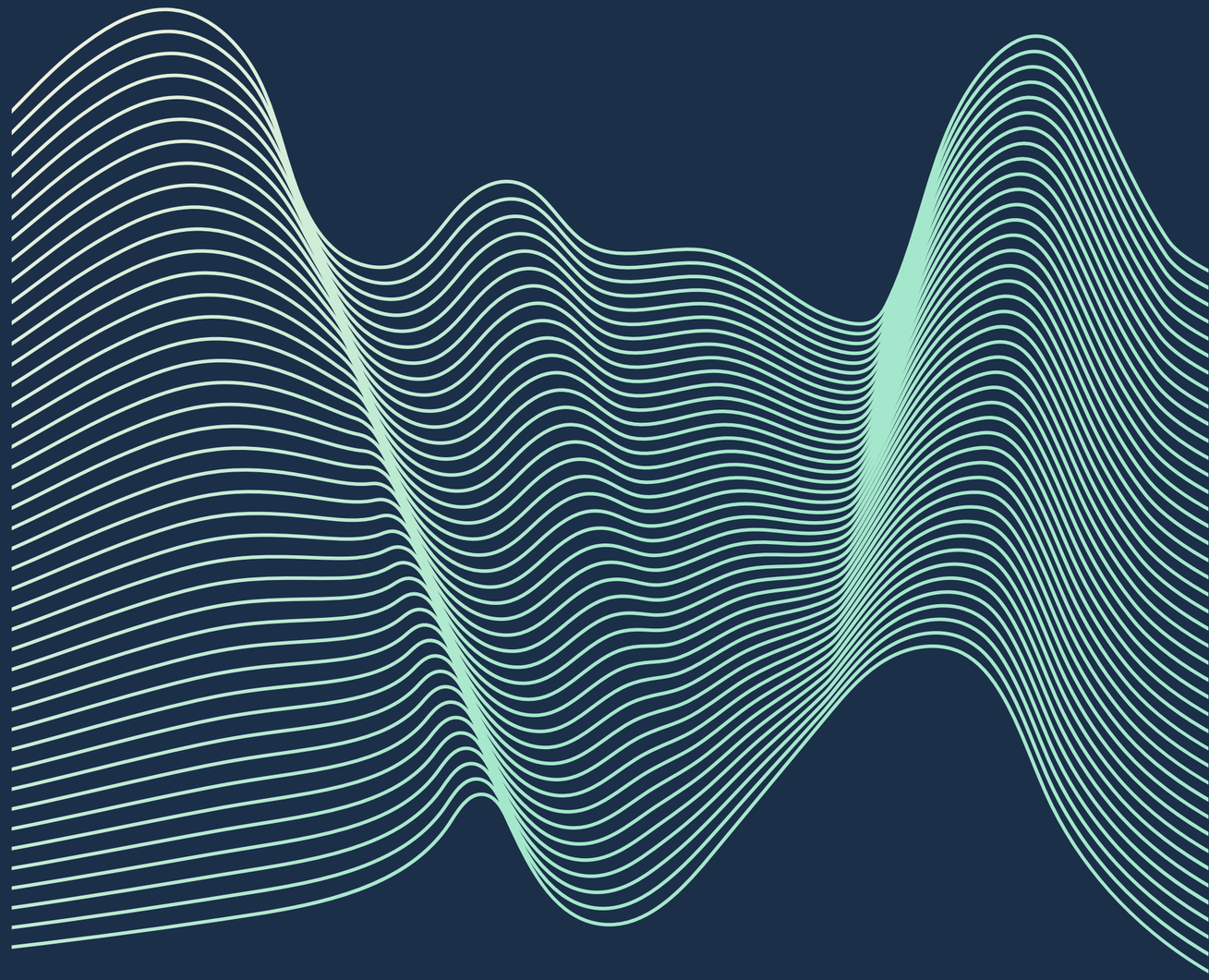
Abril 2025

Esta atividade faz parte da
avaliação parcial da

DISCIPLINA: PGCC011 - COMPUTAÇÃO DE ALTO DESEMPENHO

Sob orientação do:
Prof. Dr. Angelo Amâncio Duarte

PLANEJAMENTO

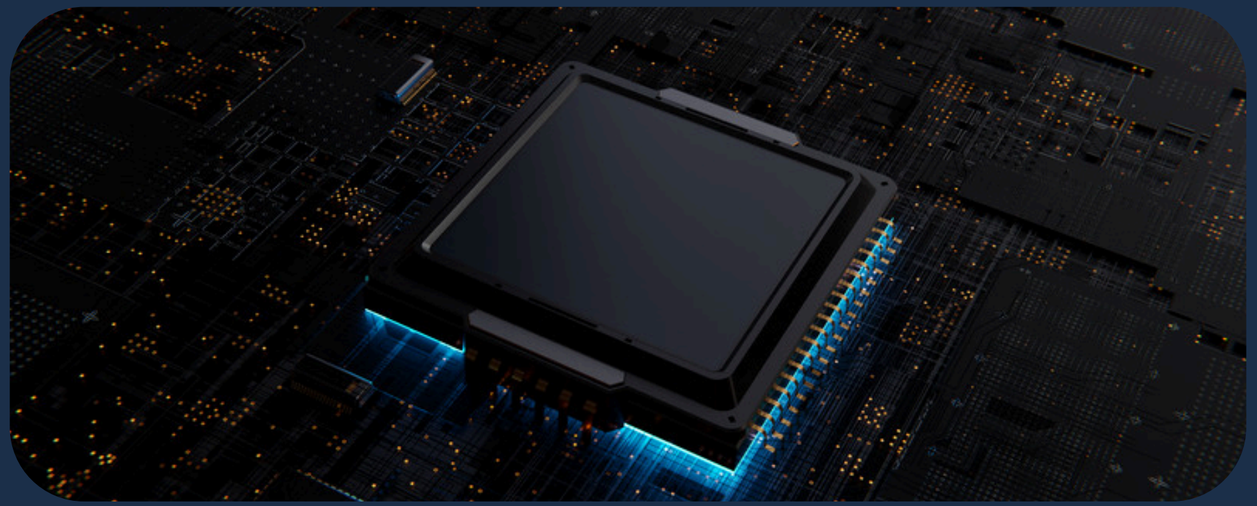


Introdução e Motivação

- Inversão de matrizes é essencial em várias áreas (aprendizado de máquina, desenvolvimento de fármacos, entre outros).

- Custo computacional elevado - O (n^3)

- Motivação: entender o comportamento da construção de matrizes inversas e buscar melhoria de performance via paralelização com OpenMP.



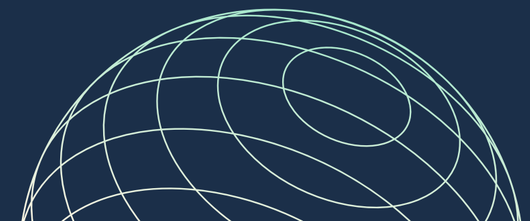
[2]

Algoritmo de Gauss-Jordan e Metodologia

- É um método direto para obtenção da matriz inversa.

- Baseia-se na transformação da matriz aumentada $[A \mid I] \rightarrow [I \mid A^{-1}]$ através de operações de linha elementares.

- Envolve:
  - Busca do pivô
  - Normalização da linha
  - Eliminação de Gauss
- Versões: Sequencial (linha e coluna) e Paralela (linha)



Algoritmo de Gauss-Jordan e Metodologia

A metodologia adotada neste trabalho consistiu na:

- implementacão,
- execucão e
- análise comparativa

Do comportamento algoritmo de inversão de matrizes baseado no método de Gauss-Jordan utilizando paralelização via OpenMP

Plataforma de Execução

- Sistema Operacional: Linux mint 22.1 Cinnamon ‘Xia’
- Kernel: 6.8-generic x86_64
- Compilador: gcc versão 9.4.0 com suporte a OpenMP
- CPU: AMD Ryzen 3 3250U with Radeon Graphics (2 núcleos / 4 threads)
- Memória RAM: 8 GB DDR4
- Armazenamento: SSD 256GB NVMe
- Ambiente de Execução: Terminal Linux via linha de comando

[2]

Serialização e Pontos Chave de Paralelização

[2]

- Foram desenvolvidos dois programas em linguagem C:
- Versão sequencial (im_serial.c): que permite selecionar a orientação da eliminação (por linhas ou por colunas) via argumento de linha de comando.
- Versão paralela (im_parallel.c): utiliza a biblioteca OpenMP para paralelizar as etapas de normalização da linha pivô e a eliminação de Gauss, adotando orientação por linhas.
- Ambas as versões utilizam uma mesma matriz de entrada, gerada aleatoriamente e inversível, a fim de garantir a comparabilidade dos experimentos.

Estratégias de paralelização com OpenMP

1. Paralelização do Loop de Inicialização da Matriz Identidade

[1]

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        Ainv[i*n + j] = (i == j) ? 1.0 : 0.0;
    }
}
```

Cada thread recebe um subconjunto das linhas para inicializar.

Estratégias de paralelização com OpenMP

2. Paralelização da Busca do Pivô com Redução

[1]

```
// Redução para encontrar o pivô em paralelo
#pragma omp parallel
{
    int local_pivot_row = pivot_row;
    double local_pivot_value = pivot_value;

    #pragma omp for nowait
    for (int i = k + 1; i < n; i++) {
        double abs_value = fabs(temp_A[i*n + k]);
        if (abs_value > local_pivot_value) {
            local_pivot_value = abs_value;
            local_pivot_row = i;
        }
    }
```

[1]

```
    // Atualizar o pivô global
    #pragma omp critical
    {
        if (local_pivot_value > pivot_value) {
            pivot_value = local_pivot_value;
            pivot_row = local_pivot_row;
        }
    }
}
```

Realiza-se uma redução para encontrar o pivô, onde cada thread procura o valor máximo em sua porção designada e depois usa uma seção crítica para atualizar o valor global caso necessário.

Estratégias de paralelização com OpenMP

3. Paralelização da Normalização da Linha do Pivô

[1]

```
#pragma omp parallel for
for (int j = 0; j < n; j++) {
    temp_A[k*n + j] /= pivot;
    Ainv[k*n + j] /= pivot;
}
```

Mais eficiente em matrizes grandes, pois divide cada elemento de uma linha pelo valor do pivô.

Estratégias de paralelização com OpenMP

4. Paralelização da Eliminação de Gauss

[1]

```
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < n; i++) {
    if (i != k) {
        double factor = temp_A[i*n + k];
        for (int j = 0; j < n; j++) {
            temp_A[i*n + j] -= factor * temp_A[k*n + j];
            Ainv[i*n + j] -= factor * Ainv[k*n + j];
        }
```

[2]

Esta é a parte mais computacionalmente intensiva do algoritmo.

Foi utilizado schedule(dynamic) para melhor balanceamento de carga, já que algumas linhas podem terminar mais rápido que outras dependendo do valor do fator.

Estratégias de paralelização com OpenMP

5. Paralelização da Validação da Matriz Inversa

[1]

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        result[i*n + j] = 0.0;
        for (int k = 0; k < n; k++) {
            result[i*n + j] += A[i*n + k] * Ainv[k*n + j];
        }
    }
}
```

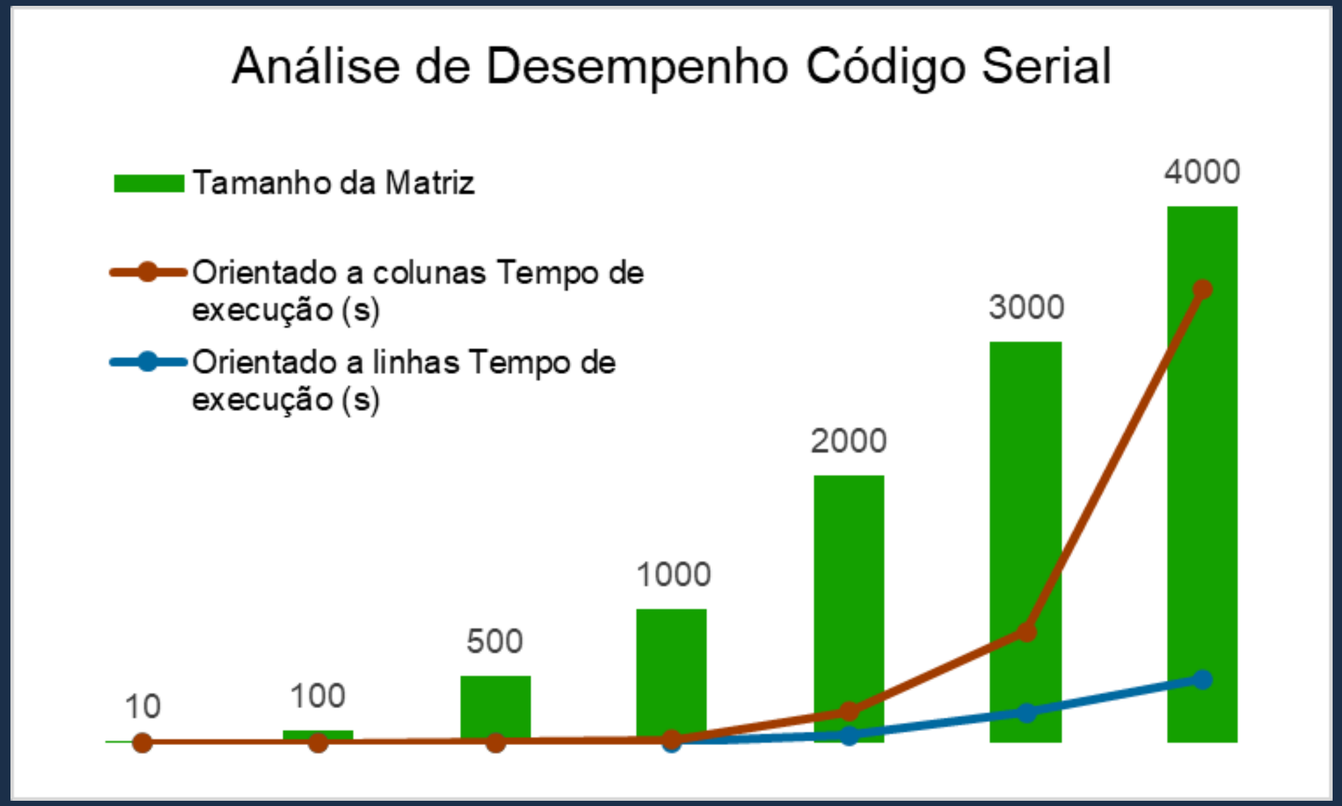
A multiplicação de matrizes para validação também foi paralelizada para melhorar o desempenho.

Resultados e Análise de Desempenho

Código Serial [1]

Tamanho da Matriz	Orientado a linhas	Orientado a colunas
	Tempo de execução (s)	Tempo de execução (s)
10	0,000029	0,000075
100	0,008665	0,020241
500	1,068074	1,642287
1000	7,319517	15,256861
2000	55,554869	179,556352
3000	228,676957	600,686618
4000	480,362635	2911,000828

[1]



Resultados e Análise de Desempenho

[1]

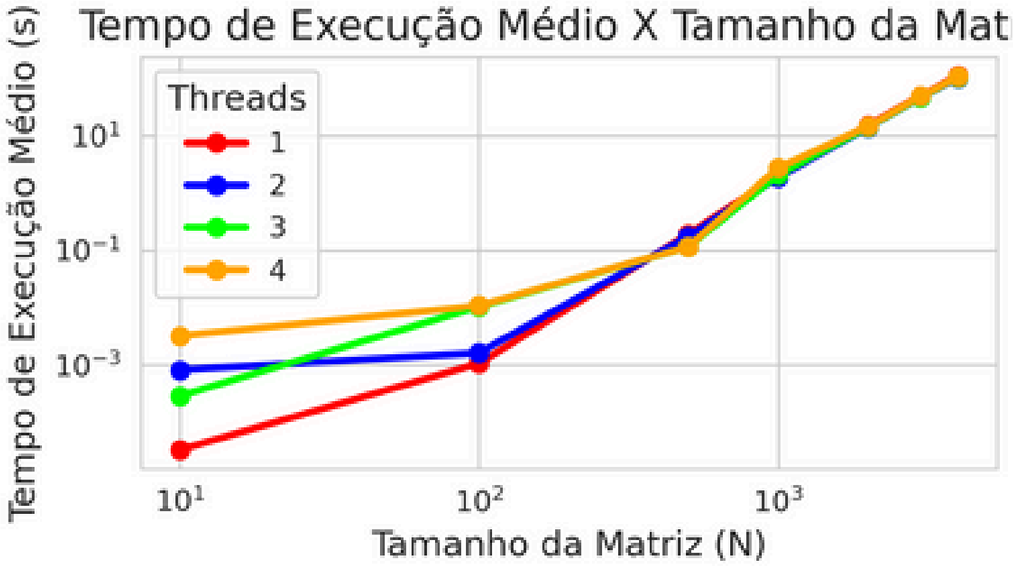
Tabela de melhorias por tamanho de matriz					
Tamanho da Matriz	Tempo Serial (s)	Tempo Paralelo (s)	Melhoria (%)	Speedup	Threads Ótimo
10	0.000029	0.000033	-12,64	0.887755	1
100	0.008665	0.001053	87,85	8.228870	1
500	1.068.074	0.115036	89,23	9.284693	3
1000	7.319.517	1.885.357	74,24	3.882298	2
2000	55.554.869	14.172.381	74,49	3.919939	2
3000	228.676.957	47.123.314	79,39	4.852735	2
4000	480.362.635	104.475.110	78,25	4.597867	2

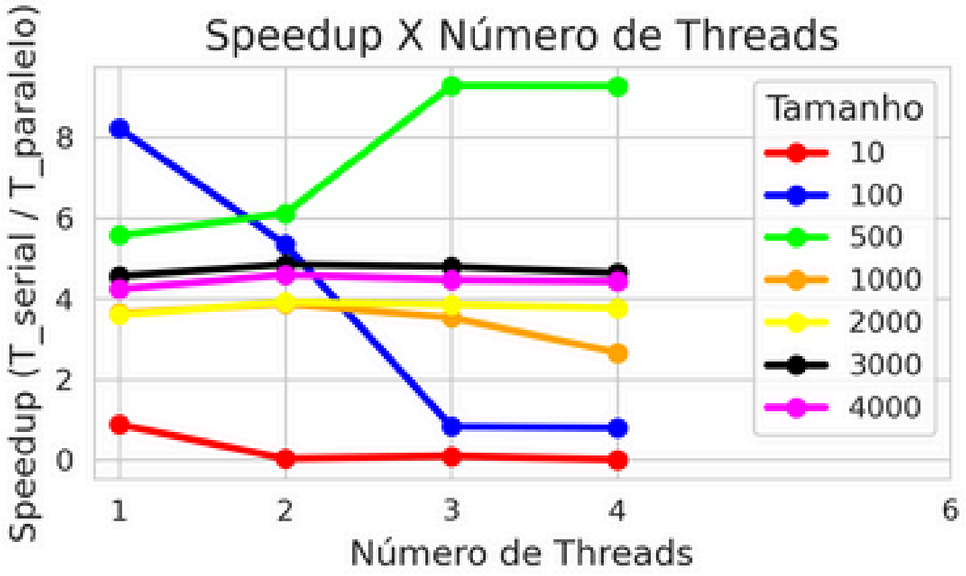
[1]

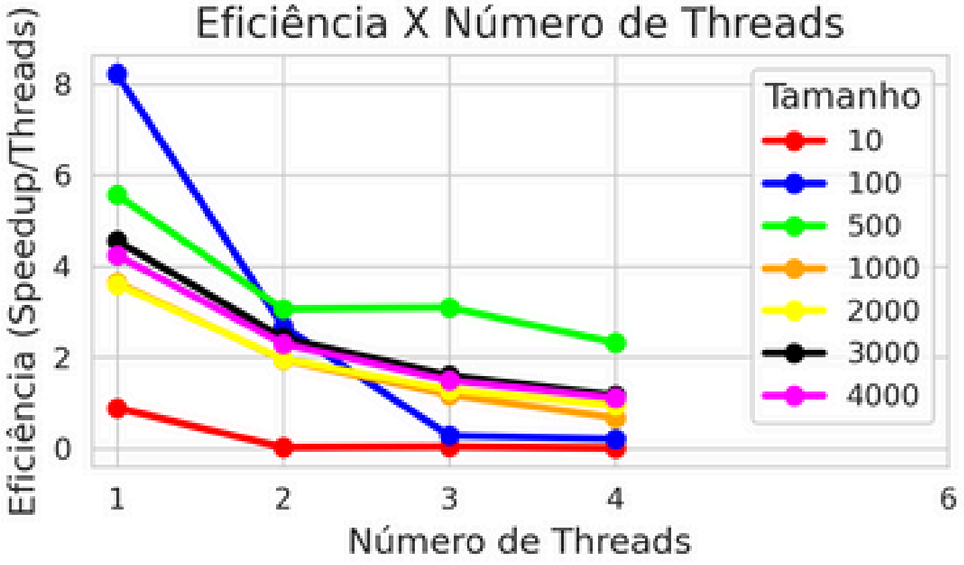
Tempo médio por tamanho de matriz e número de threads				
Tamanho da Matriz	Número de threads			
	1	2	3	4
10	0.000033	0.000813	0.000286	0.003197
100	0.001053	0.001618	0.010372	0.010792
500	0.191714	0.174658	0.115036	0.115187
1000	2.009424	1.885357	2.071090	2.748241
2000	15.425779	14.172381	14.402178	14.735922
3000	50.131242	47.123314	47.662517	49.289630
4000	113.222778	104.475110	107.448410	108.424802

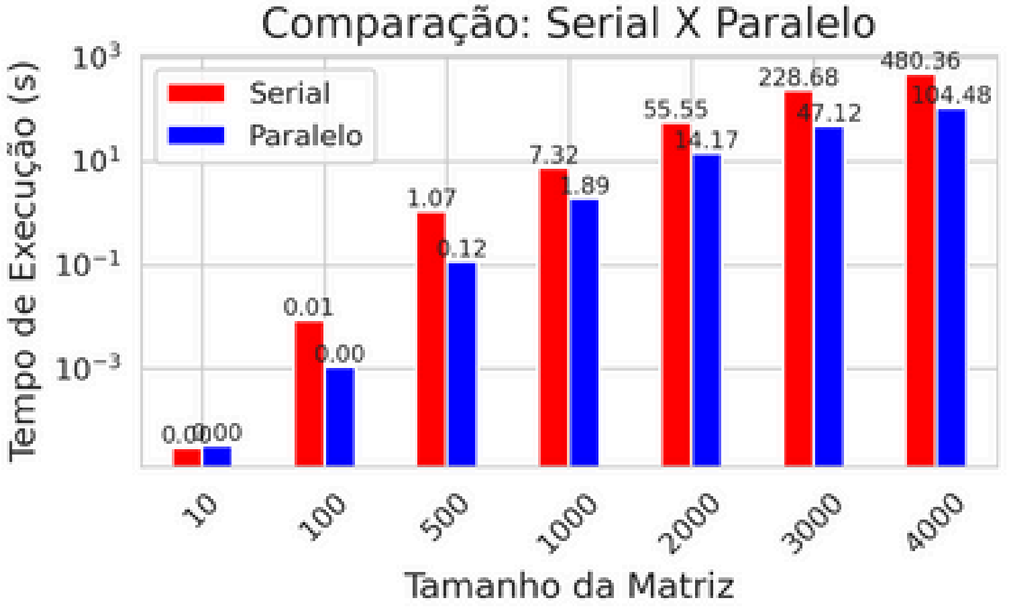
Comparativo entre os códigos serial e paralelo

Resultados e Análise de Desempenho





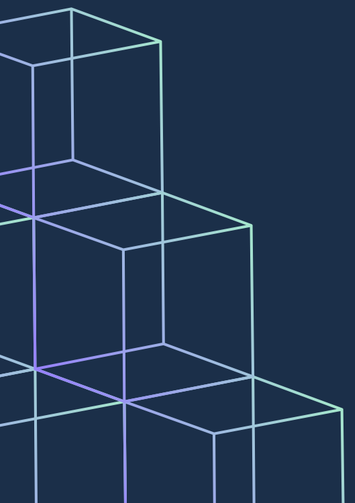




[1]

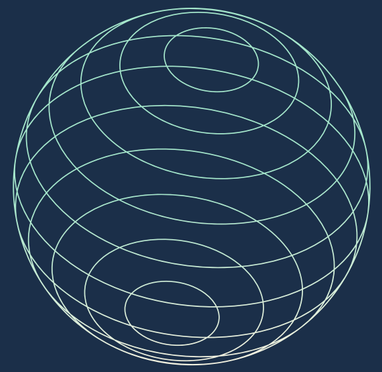
Conclusões e trabalhos Futuros

- Inversão de matrizes é essencial em várias áreas (aprendizado de máquina, sistemas de controle, entre outros).
- Custo computacional elevado.
- Motivação: melhorar o desempenho via paralelização com OpenMP.



REFERÊNCIAS

*Todas as imagens foram desenvolvidas através dos resultados encontrados pelos autores [1] ou sob a licença do canva gratuito [2] (disponível em: http://canva.com)



1. Barney, Blaise. OpenMP Tutorial. Lawrence Livermore National Laboratory, 2010. Available at: https://hpc.llnl.gov/tutorials/openMP/.

2. Golub, Gene H. and Van Loan, Charles F. Matrix Computations. 4th ed. Johns Hopkins University Press, 2013.

3. Gupta, Ravi and Misra, Rajiv. Parallel matrix inversion algorithm and its implementation using OpenMP. In: Proceedings of the ICCCCT, 2010, IEEE, pp. 345-349.

4. Mehtre, V. V.and Tiwari, Shubham. Review on Gauss Jordan Method and it's Applications. 4th ed. International Journal of Advances in Engineering and Management (IJAEM), 2022, pp: 540-54

OBRIGADO!