Eliminação de Gauss com OpenMP e MPI

Marcelo Vieira Aguiar

Departamento de Engenharia de Sistemas e Computação, Universidade do Estado do Rio de Janeiro (UERJ)

m.aguiar@eng.uerj.br

15 de setembro de 2022

Abstract

Gaussian Elimination is one of the methods for solving systems of linear equations. These systems are present in several problems in Engineering, Computer Science and Mathematics. An example is the resolution of electrical circuits using Gaussian Elimination. However, this method has a large computational cost, since the number of operations is $\frac{2n^2}{3}$, where n is the dimension of the system. To reduce computational time, it is necessary to use parallel programming, such as OpenMP and MPI. Therefore, the objective of this work is to implement algorithms in OpenMP and MPI to reduce the Gaussian Elimination time in the resolution of linear systems. It will be seen that these algorithms can be twice as fast as the sequential algorithm.

I. Introdução

Sistemas de Equações Lineares podem ser definidos como um conjunto de equações de primeiro grau que possuem as mesmas incógnitas [1]. A Equação 1 mostra um exemplo de sistema linear, onde a_{ij} são os coeficientes, x_i são as incógnitas, b_i o termo independente e n é o número de equações ou dimensão do sistema.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \vdots \quad + \quad \vdots \quad + \ddots + \quad \vdots \quad = \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{1}$$

Essas equações são ferramentas comuns para resolução de problemas práticos presentes em diversas áreas, como engenharia (elétrica, civil, mecânica, química, etc.), matemática e computação [2]. Alguns exemplos de aplicações nessas áreas são: Resolução de circuitos elétricos, controle de tráfego de veículos, balanceamento de equações químicas, sistemas de GPS e mecanismos de busca na internet. Portanto, o objetivo dos sistemas lineares é modelar um problema, para depois encontrar a solução (valor das incógnitas).

Existem três possibilidades de solução, sendo elas:

- O sistema tem exatamente uma solução (Sistema Possível e Determinado), se o determinante for diferente de zero.
- O sistema tem infinitas soluções (Sistema Possível e Indeterminado), se o determinante for igual a zero.
- O sistema não possui soluções (Sistema Impossível), se o determinante principal for igual a zero e o determinante secundário for diferente de zero.

Existem vários métodos de resolução de sistemas lineares conforme [3]. A Eliminação de Gauss é um dos métodos, e será objeto de estudo desse trabalho. Essa forma de resolução consiste em manipular o sistema através de operações elementares, transformando a matriz estendida do sistema em uma matriz triangular chamada de matriz escalonada.

As operações usadas na Eliminação de Gauss são simples, porém, a quantidade de operações

é muito grande. Segundo [4], a quantidade de operações é $\frac{2n^2}{3}$, onde n é a dimensão do sistema. Portanto, a resolução manual de um sistema linear com uma dimensão grande é descartada. Já a solução computacional é lenta para grandes dimensões, e por esse motivo é mais vantajoso paralelizar o algoritmo da Eliminação de Gauss.

O presente trabalho tem objetivo de paralelizar o algoritmo da Eliminação de Gauss com OpenMP [1] e MPI [2]. Além disso, será comparado o algoritmo da Eliminação de Gauss executando em dois computadores com diferentes processadores.

Os resultados obtidos com OpenMP tiveram *speedup* próximo de 3 para um sistema de dimensão igual a 3000. O algoritmo em MPI teve pouco ganho (*speedup* igual a 1,43 para n igual a 1000), já que o tempo de comunicação entre os processos é grande. Foi observado que um computador com processador mais atual e com mais *threads*, tem uma performance de até 4 vezes mais rápida que um computador com processador mais antigo.

II. Trabalhos relacionados

A Eliminação de Gauss tem sido tema de diversos trabalhos. Como o trabalho [5], no qual, é feita a comparação da Eliminação de Gauss usando MPI e OpenMP com diferentes *chuncks*. O autor alcançou bons *speedups* usando OpenMP. Como, por exemplo, a matriz de dimensão 1200 obteve um ganho igual a 3,44. Além disso, foi observado pelo autor que o MPI teve pouco ganho, já que 90% do tempo era para comunicação entre os processos.

Em [6], o autor apresentou três versões do algoritmo da Eliminação de Gauss e a Decomposição LU usando OpenMP. A primeira versão foi *program omp paralell for* sem uso de *schedule*. A segunda foi com uso de *shedule static* e a última versão foi utilizado *pipeline* junto com OpenMP. Este último algoritmo teve um *speedup* igual a 3,25 para n igual a 4096. Além disso, o autor observou que o $speedup$ foi maior usando a Decomposição LU.

Já o trabalho [7], o autor utiliza um computador de alto desempenho para comparar três ferramentas de computação paralela: Cuda, OpenACC e OpenMP. Cuda e OpenACC utilizam as *threads* da placa gráfica e OpenMP utiliza as *threads* do processador. Os *speedups* para o algoritmo implementado em Cuda foram os maiores, alcançando um valor igual a 40. Já o OpenMP teve o menor $speedup$.

III. Algoritmo da Eliminação de Gauss

O objetivo da Eliminação de Gauss é encontrar as incógnitas do sistema linear. A Equação 2 mostra um exemplo de sistema linear possível e determinado.

$$\begin{cases} 2x + 1y - 3z = -1 \\ -1x + 3y + 2z = 12 \\ 3x + 1y - 3z = 0 \end{cases} \tag{2}$$

Para encontrar as incógnitas pela Eliminação de Gauss, o sistema é transformado em uma matriz aumentada, conforme 3.

$$\left[\begin{array}{ccc|c} 2 & 1 & -3 & -1 \\ -1 & 3 & 2 & 12 \\ 3 & 1 & -3 & 0 \end{array}\right] \tag{3}$$

A partir da matriz aumentada é possível aplicar as transformações lineares para encontrar a matriz escalonada. Essa matriz deve possuir as seguintes características:

- Todas as linhas não-nulas estão acima de qualquer linha composta só de zeros.
- O pivô [3] de cada linha está numa coluna à direita do pivô da linha acima.
- Todos os elementos de uma coluna abaixo de um pivô são zero.

A transformação linear aplicada a matriz aumentada segue o padrão da Equação 4. O

[1] API padrão para programação multithreading com memória compartilhada.

[2] padrão de comunicação de dados em computação paralela.

[3] são os elementos da diagonal principal.

objetivo dessa transformação é zerar todos elementos abaixo dos pivôs.

$$L_{i+1} \leftarrow L_{i+1} - R * L_i \tag{4}$$

Onde L_i são todos elementos da linha i e R é a razão do número abaixo do pivô por ele mesmo.

Considerando que $i = 0$ (primeira linha), lê-se a Equação 4 da seguinte forma: *linha 2 vai receber a linha 2, menos a razão vezes a linha 1.*

Conforme as características da matriz escalonada, deve-se iniciar a transformação linear pelo pivô da primeira coluna, para então aplicar a transformação da Equação 4 em todas as linhas abaixo do pivô da primeira coluna. Essa transformação não precisa ser sequencial. Em seguida, deve ser aplicada a mesma operação, agora considerando o pivô da segunda coluna.

Observa-se que a transformação nas linhas é possível paralelizar, porém, deve ser feito para cada pivô sequencialmente, ou seja, deve-se iniciar pelo pivô da primeira coluna aplicar a transformação linear para todas as linhas abaixo desse pivô. Em seguida, utilizando o pivô da segunda coluna, deve-se aplicar a transformação para todas as linhas abaixo dele, e assim por diante.

Aplicando a transformação linear somente na linha 2 da matriz 3, considerando o pivô da primeira coluna, tem-se que $R = -\frac{1}{2}$ e a matriz parcialmente escalonada é 5.

$$\left[\begin{array}{ccc|c} 2 & 1 & -3 & -1 \\ 0 & \frac{7}{2} & \frac{1}{2} & \frac{23}{2} \\ 3 & 1 & -3 & 0 \end{array}\right] \tag{5}$$

Aplicando a transformação para toda matriz 3 obtém-se a matriz escalonada 6.

$$\left[\begin{array}{ccc|c} 2 & 1 & -3 & -1 \\ 0 & \frac{7}{2} & \frac{1}{2} & \frac{23}{2} \\ 0 & 0 & \frac{11}{7} & \frac{22}{7} \end{array}\right] \tag{6}$$

Convertendo a matriz escalonada em sistema linear, conforme 7, é possível encontrar as incógnitas x, y e z que é a solução do sistema linear. Logo, o conjunto solução para o exemplo 2 é $(1, 3, 2)$

$$\begin{cases} 2x + 1y - 3z = -1 \\ \frac{7}{2}y + \frac{1}{2}z = \frac{23}{2} \\ \frac{11}{7}z = \frac{22}{7} \end{cases} \tag{7}$$

IV. Ambiente de execução

Para execução do algoritmo de Gauss foram utilizados dois computadores distintos a fim de testar o desempenho de um processador mais novo (CPU2) em relação a um processador mais antigo (CPU1). A Tabela 1 mostra as especificações da CPU1 e a Tabela 2 mostra da CPU2.

Especificação	**Descrição**
Processador	Intel Core i5 460-M de 2,5GHz
Geração	Intel® Core™ antigos
RAM	3,6GB
Cache	3MB
Núcleos	2
Threads	4
Hyper-Threading	Sim

Tabela 1: *Especificações da CPU1.*

Especificação	**Descrição**
Processador	Intel Core i7-8565U de 4,6 GHz
Geração	8ª geração
RAM	16GB
Cache	8MB
Núcleos	4
Threads	8
Hyper-Threading	Sim

Tabela 2: *Especificações da CPU2.*

A CPU1 tem como sistema operacional Linux Debian 11, já a CPU2 tem o sistema

operacional Windows 10. Nessa CPU, o código em C foi executado usando o módulo WSL (Windows Subsystem for Linux) [4]

V. Algoritmo sequencial da Eliminação de Gauss

O código sequencial, que será paralelizado nas próximas seções, foi obtido da Oitava Maratona de Programação Paralela de 2013 e pode ser baixado por [8]. Ele foi escrito na linguagem C e está divido em três etapas principais, sendo elas: Criação da matriz com números aleatórios (Generate Matrix), resolução do sistema linear (Solve System) e validação da solução (Validates solution).

Durante as execuções do algoritmo sequencial, foi verificado que 99,9% do tempo total era apenas para resolução do sistema linear, ou seja, a parte do código que deverá ser paralelizada é a responsável por resolver o sistema linear. A Figura 1 mostra o algoritmo sequencial.

```c
/* Gaussian Elimination */
for (i = 0; i < (n - 1); i++) {

    for (j = (i + 1); j < n; j++) {
        float ratio = Acpy[j * n + i] / Acpy[i * n + i];
        for (count = i; count < n; count++) {
            Acpy[j * n + count] -= (ratio * Acpy[i * n + count]);
        }
        bcpy[j] -= (ratio * bcpy[i]);
    }
}
```

Figura 1: *Algoritmo sequencial da Eliminação de Gauss.*

A versão completa do algoritmo sequencial e das versões paralelas que serão apresentadas nas próximas seções podem ser encontradas em [9]. Os resultados das execuções do algoritmo sequencial estão na seção Algoritmo sequencial.

VI. Paralelização com OpenMP

Como foi discutido na seção Algoritmo da Eliminação de Gauss, não é possível paralelizar a transformação linear usando pivôs diferentes, ou seja, as *threads* devem ter o mesmo pivô para fazer as operações em paralelo. Dessa forma, as operações nas linhas com mesmo pivô que serão paralelizadas. Isso implica que o segundo *for* do algoritmo sequencial será paralelizado, pois o primeiro *for* faz o controle do pivô.

Foram feitos três algoritmos paralelos usando OpenMP, sendo eles: O primeiro utiliza directiva *omp for*, o segundo utiliza a mesma directiva mas com *shedule dynamic* e o terceiro utiliza a directiva *taskloop*. A Figura 2 mostra o algoritmo com a directiva *omp for shedule dynamic* e a Figura 3 a directiva *taskloop*.

```c
for (i = 0; i < (n - 1); i++) {
    #pragma omp parallel shared(n, i, Acpy, bcpy) private(j, count)
    {
        #pragma omp for schedule(dynamic)
        for (j = (i + 1); j < n; j++) {
            float ratio = Acpy[j * n + i] / Acpy[i * n + i];
            for (count = i; count < n; count++) {
                Acpy[j * n + count] -= (ratio * Acpy[i * n + count]);
            }
            bcpy[j] -= (ratio * bcpy[i]);
        }
    }
}
```

Figura 2: *Algoritmo da Eliminação de Gauss com omp for shedule dynamic.*

```c
/* Gaussian Elimination */
#pragma omp parallel shared(n, i, Acpy, bcpy) private(j, count)
{
    #pragma omp single
    {
    for (i = 0; i < (n - 1); i++) {
        #pragma omp taskloop private(j, count)
        for (j = (i + 1); j < n; j++) {
            float ratio = Acpy[j * n + i] / Acpy[i * n + i];
            for (count = i; count < n; count++) {
                Acpy[j * n + count] -= (ratio * Acpy[i * n + count]);
            }
            bcpy[j] -= (ratio * bcpy[i]);
        }
    }
}
```

Figura 3: *Algoritmo da Eliminação de Gauss com taskloop.*

Os resultados das execuções dos algoritmos usando OpenMP estão na seção Algoritmo com OpenMP.

VII. Paralelização com MPI

A mesma lógica utilizada na paralelização do OpenMP foi utilizada para a paralelização com

[4] disponibiliza um ambiente Linux compatível no sistema da Microsoft, sem instalar emuladores e máquinas virtuais.

MPI, ou seja, cada processo deve calcular a transformação linear em uma determinada quantidade de linhas. Porém, todos os processos devem trabalhar em paralelo com o mesmo pivô.

Dessa forma, o algoritmo com MPI pode ser dividido em quatro etapas, sendo elas:

- Etapa de declaração das variáveis e envio dos dados do processo zero para os demais processos: Nessa etapa o processo zero recebe a dimensão da matriz, cria o sistema com números aleatórios e envia esses dados para os demais processos.
- Etapa de envio das linhas essenciais para cada processo: É enviado, pelo processo zero para os demais processos, apenas a linha do pivô e as linhas que o processo ficou responsável por fazer a transformação linear.
- Etapa do cálculo da Eliminação de Gauss e envio do resultado para o processo zero: Cada processo faz seu respectivo cálculo e envia o resultado para o processo zero.
- Etapa de contabilização do tempo e verificação do resultado: Essa etapa é realizada apenas pelo processo zero.

A implementação do algoritmo da Eliminação de Gauss em MPI pode ser encontrada em [9]. Os resultados da execução desse algoritmo estão na seção Algoritmo com MPI.

VIII. Resultados experimentais

Para todos os resultados experimentais, foram executadas 10 rodadas de teste e calculada a média para ter uma medida mais assertiva.

i. Algoritmo sequencial

A Tabela 3 mostra o resultado em segundos do tempo de execução do algoritmo sequencial da Eliminação de Gauss na CPU1 e na CPU2.

A partir dos dados da Tabela 3, foi calculado o speedup da CPU2 em relação a CPU1, e os dados foram dispostos na Tabela 4

Dimensão	CPU1 (s)	CPU2 (s)
100	0,0058	0,0015
1000	2,4837	0,8294
2000	19,7144	6,7400
3000	69,0616	22,6300
5000	334,2289	105,6600

Tabela 3: *Tempo de execução sequencial da Eliminação de Gauss nas CPU's 1 e 2.*

Dimensão	Speedup
100	3,95
1000	2,99
2000	2,93
3000	3,05
5000	3,16

Tabela 4: *Speedup do algoritmo sequencial comparando as duas CPU's.*

ii. Algoritmo com OpenMP

Os resultados da execução do algoritmo usando OpenMP estão divididos em duas tabelas. A Tabela 5 mostra o tempo de execução dos três algoritmos na CPU1. Já a Tabela 6 mostra o tempo de execução na CPU2. Os algoritmos foram executados com 4 *threads* nas duas CPU's.

Dimensão	For(s)	Task(s)	Schedule(s)
100	0,0027	0,0023	0,0025
1000	1,6741	1,3617	1,2926
2000	14,8628	13,4293	13,3395
3000	50,1060	47,5967	47,6605
5000	228,8058	230,6732	227,8300

Tabela 5: *Tempo de execução do Algoritmo de Gauss em OpenMP na CPU1.*

A partir dos dados da Tabela 6, foi calculado o *speedup* da CPU2 em relação ao algoritmo se-

Dimensão	For(s)	Task(s)	Schedule(s)
100	0,0007	0,0009	0,0008
1000	0,3704	0,3063	0,3004
2000	2,8090	2,4300	2,3600
3000	9,2272	8,1900	8,0200
5000	52,5619	48,4800	48,3300

Tabela 6: *Tempo de execução do Algoritmo de Gauss em OpenMP na CPU2.*

quencial da mesma, e os dados foram dispostos na Tabela 7

Dimensão	For	Task	Schedule
100	2,02	1,73	1,90
1000	2,24	2,71	2,76
2000	2,40	2,77	2,85
3000	2,45	2,76	2,82
5000	2,01	2,18	2,19

Tabela 7: *Speedup do OpenMP em relação ao algoritmo sequencial de Gauss da CPU2.*

iii. Algoritmo com MPI

Os resultados desse algoritmo estão divididos de forma semelhante a seção Algoritmo sequencial. A Tabela 8 mostra o tempo de execução do algoritmo em MPI nas duas CPU's.

Dimensão	CPU1 (s)	CPU2 (s)
100	0,0033	0,0011
1000	1,8322	0,5839
2000	16,8377	5,9400
3000	56,4940	24,2700
5000	282,2222	108,0200

Tabela 8: *Tempo de execução com MPI nas CPU's 1 e 2.*

A partir dos dados da Tabela 8, foi calculado o speedup da CPU2 em relação ao algoritmo sequencial da mesma, e os dados foram dispostos na Tabela 9

Dimensão	Speedup
100	1,39
1000	1,42
2000	1,13
3000	0,93
5000	0,98

Tabela 9: *Speedup do MPI em relação ao algoritmo sequencial da CPU2.*

Os resultados anteriores foram executados no MPI com apenas dois processos. Também foi feito um experimento com diferentes números de processos (1 a 4 processos) para uma dimensão igual a 3000. Nesse experimento foi computado o tempo de execução da Eliminação de Gauss em si (conjunto de *for's*) e o tempo de comunicação entre os processos. A Figura 4 mostra os resultados obtidos.

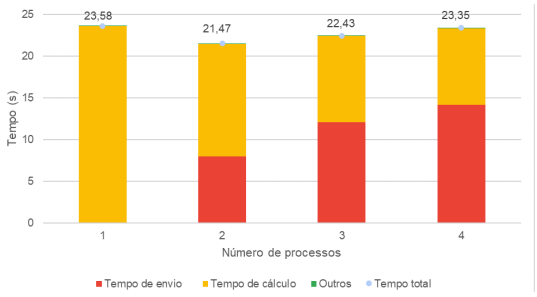

Figura 4: *Tempo de execução no MPI para diferentes números de processos na CPU2.*

IX. Avaliação de desempenho

Pode-se observar, pelos resultados da Tabela 3, que o tempo sequencial da Eliminação de Gauss é elevado. Foi feito um gráfico, conforme a Figura 5, para comparar o custo computacional do algoritmo sequencial e do algoritmo paralelizado usando tasks com diferentes dimensões do sistema linear. Os algoritmos

foram executados na CPU1, e o tempo sequencial chegou a 46 minutos para dimensão 10000.

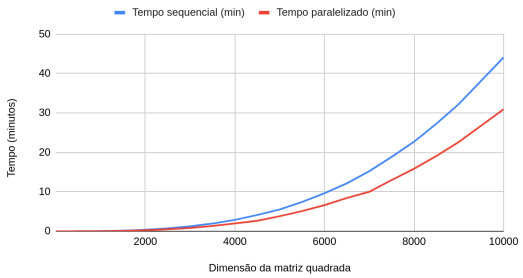

Figura 5: *Tempo de execução sequencial e paralelizado (tasks) para diferentes dimensões na CPU1.*

A CPU2 mostrou um desempenho muito superior em relação a CPU1, conforme a Tabela 4. Esse resultado era esperado pois a CPU2 apresenta especificações superiores. A Figura 6 compara o desempenho das duas CPU's executando o algoritmo sequencial da Eliminação de Gauss para diferentes dimensões. Para dimensão igual a 100, a CPU2 é 4 vezes mais rápida que a CPU1.

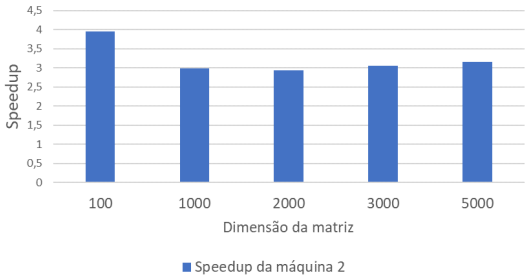

Figura 6: *Speedup da CPU2 em relação a CPU1.*

Os *speedups* da CPU2 apresentados nas Tabelas 7 e 9 foram dispostos na Figura 7. Os algoritmos usando *tasks* e *for* com *schedule* tiveram os melhores desempenhos.

Os *speedups* da CPU1 também foram calculados e plotados no gráfico da Figura 8. Nessa CPU o melhor algoritmo foi com *tasks*.

No gráfico das Figuras 7 e 8 o MPI apresentou o menor *speedup*. Isso aconteceu, pois, o tempo de comunicação é proporcional a dimensão do sistema linear e a quantidade de processos. Logo, grande parte do tempo do

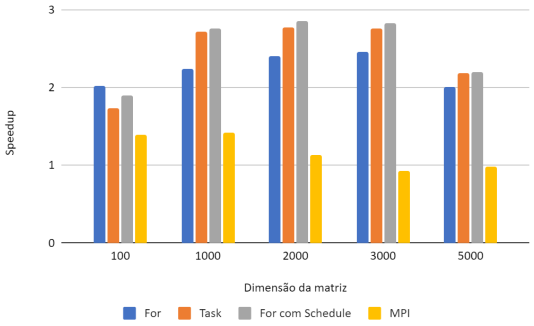

Figura 7: *Speedups da CPU2.*

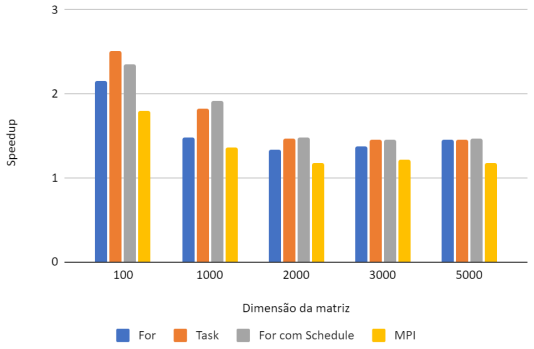

Figura 8: *Speedups da CPU1.*

algoritmo em MPI é usado para comunicação e não para o cálculo da Eliminação de Gauss, conforme a Figura 4.

Em média o algoritmo com *taskloop* apresentou o maior *speedup* pois o algoritmo não cria novas *threads* a cada iteração do primeiro *for*, como acontece com algoritmo que utiliza apenas *omp for*. No algoritmo com *taskloop* é criado apenas as *tasks*, e o tempo de criação delas é menor do que o tempo de criação de *threads*. Essa comparação pode ser feita nos algoritmos das Figuras 2 e 3, no qual, a directiva *omp parallel for* está dentro do primeiro *for*.

X. Conclusões

A partir dos resultados obtidos, percebe-se que a implementação em OpenMP usando *taskloop* na CPU2 teve ganho de desempenho significativo. A CPU1 teve um bom desempenho, mas como a CPU2 tem um processador melhor, e por isso os resultados foram melhores. O grá-

fico da CPU1, conforme a Figura 8 não ultrapassou o *speedup* igual a 2 para matrizes de dimensão 3000 e 5000. Já o gráfico da CPU2, conforme Figura 7, o *speedup* foi superior a 2.

Como foi citado na seção Avaliação de desempenho, o algoritmo com *taskloop* evita a criação e finalização das *threads* a cada iteração do primeiro *for*. Por esse motivo, o algoritmo com *taskloop*, na média, teve um melhor desempenho.

O MPI não teve muito ganho de desempenho, e foi o algoritmo mais trabalhoso para implementar. Em MPI, há muita comunicação entre os processos pois é necessário sincronizar o pivô entre eles. Esse problema da dependência do pivô é inerente a Eliminação de Gauss. A Figura 4 mostra o tempo gasto na comunicação. Como [5] cita, existe um número da dimensão do sistema linear que o desempenho do MPI pode superar o OpenMP.

Como trabalho futuro, pretende-se implementar outros algoritmos para solução de sistemas de lineares que não tenha dependência de dados quanto a Eliminação de Gauss. Como, por exemplo, a decomposição LU. Dessa forma, poderia ter um ganho maior de desempenho no algoritmo usando OpenMP e, principalmente, em MPI.

References

[1] J. A. T. Barbosa, *Noções sobre Matrizes e Sistemas de Equações Lineares*. Edições, 2a edição edition, FEUP, 2011.

[2] "Sistemas de equações: Algumas aplicações." http://www.ime.unicamp.br/ apmat/sistemas-lineares-algumas-aplicacoes/. Acesso: 10 set. 2022.

[3] M. E. Boccard, *Sistemas lineares: aplicações e propostas de aula usando a metodologia de resolução de problemas e o software GeoGebra*. Dissertação de mestrado, Respositório Institucional Unesp, 2017.

[4] M. A. G. Ruggiero, "Álgebra linear e aplicações." https://www.ime.unicamp.br/marcia/AlgebraLinear/eliminacao_gaussiana.html. Acesso: 10 set. 2022.

[5] S. McGinn and R. Shaw, "Parallel gaussian elimination using openmp and mpi," in *Proceedings 16th Annual International Symposium on High Performance Computing Systems and Applications*, pp. 169–173, 2002.

[6] P. D. Michailidis and K. G. Margaritis, "Parallel direct methods for solving the system of linear equations with pipelining on a multicore using openmp," *Journal of Computational and Applied Mathematics*, vol. 236, no. 3, pp. 326–341, 2011. Aspects of Numerical Algorithms, Parallelization and Applications.

[7] N. Gonçalves, C. Costa, J. Araújo, J. Costa, and J. Panetta, "Comparação e análise de desempenho de aceleradores gráficos no processamento de matrizes," in *Anais do XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, (Porto Alegre, RS, Brasil), pp. 43–55, SBC, 2015.

[8] O. M. de Programação Paralela, "Algoritmo sequencial da eliminação de gauss." http://lspd.mackenzie.br/marathon/13/gauss.zip, 2017. Acesso: 10 set. 2022.

[9] M. V. Aguiar, "Algoritmos sequencial e paralelo da eliminação de gauss." https://github.com/marcelov-aguiar/gauss_elimination. Acesso: 11 set. 2022.