ICMC USP SÃO CARLOS
UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

Computação de Alto Desempenho (SSC0903)

Trabalho 1: Jacobi em C/OMP

Integrantes:
Caique Honorio Cardoso - NºUSP: 8910222
Erika Hortência Pereira Cardoso - NºUSP: 10696830
Gabriel Cazzini Cardoso - NºUSP: 12547771
Lucas Machado Marinho - NºUSP: 11916645

Introdução

Este relatório apresenta a análise realizada acerca dos códigos sequencial e paralelo desenvolvidos para resolução de sistemas lineares pelo Método Iterativo de Jacobi-Richardson (Também conhecido como Gauss-Jacobi).

O relatório está estruturado da seguinte forma: inicialmente, é apresentada uma seção que descreve a metodologia de implementação, com foco especial na implementação paralela utilizando a biblioteca OpenMP. Em seguida, é detalhada uma seção dedicada aos resultados obtidos a partir da execução dos códigos para diferentes valores de N (tamanho do sistema) e T (número de threads para o código paralelo). Vale ressaltar que, para o código sequencial, a análise é conduzida considerando apenas diferentes valores de N.

Resolução do problema

Código sequencial:

Inicialização:
Inclui bibliotecas necessárias e define uma constante de parada. Gera valores pseudoaleatórios para a matriz A, vetor B e vetores X.

Iterações do Método de Jacobi-Richardson:
Executa iterações até atingir um critério de parada ou um número máximo de iterações. Calcula os novos valores de X com base nos valores anteriores e nos elementos de A. Verifica e atualiza o erro a cada iteração.
Liberação de Memória e Contagem de Tempo:

Libera a memória alocada.
Encerra a contagem do tempo e imprime o tempo de execução.

Código paralelo:

Inicialização:
O código inicia incluindo as bibliotecas necessárias e define uma constante de parada. Ele também define três funções para gerar valores pseudoaleatórios para a diagonal da matriz A, os elementos fora da diagonal e o vetor B.

A função principal começa a contagem do tempo e recebe os argumentos da linha de comando para o tamanho do sistema n, o número de threads T e a semente para geração dos valores pseudoaleatórios.

Inicialização das Matrizes e Vetores:
- A matriz A e o vetor B são inicializados com valores pseudoaleatórios em paralelo, utilizando a diretiva parallel for.
- Os vetores X atual e anterior também são inicializados em paralelo, utilizando a diretiva parallel for.

Iterações do Método de Jacobi-Richardson:
- O código executa iterações do Método de Jacobi-Richardson em paralelo, utilizando a diretiva parallel.
- Cada iteração é dividida em tarefas paralelas, uma para cada equação do sistema, utilizando a diretiva task.
- Dentro de cada tarefa, os cálculos são realizados em paralelo, utilizando a diretiva simd para otimizar a operação de soma.

Liberação de Memória e Contagem de Tempo:

Após as iterações, o código libera a memória alocada dinamicamente e encerra a contagem do tempo.
O tempo de execução é impresso na saída padrão.

Resultados

Os testes foram realizados com três cargas de trabalho: uma pequena (matriz de ordem 100), uma intermediária (matriz de ordem 1024) e uma grande (matriz de ordem 10240). Para cada uma das cargas, foram testados números diferentes de *threads* (5, 10 e 20 *threads*), com 30 execuções por quantidade de threads para cada carga. Os resultados são apresentados a seguir e estão divididos segundo a ordem crescente das matrizes.

Matriz de ordem 100:

Para uma matriz de ordem 100, com 5 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 1, abaixo:

Tabela 1: Tempo de execução sequencial e paralelo para uma carga $N = 100$ com $T = 5$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	0,0133	0,0420	0,3169	6,34%
2	0,0149	0,0434	0,3431	6,86%
3	0,0136	0,0413	0,3285	6,57%
4	0,0137	0,0414	0,3303	6,61%
5	0,0126	0,0407	0,3086	6,17%
6	0,0152	0,0459	0,3306	6,61%
7	0,0160	0,0395	0,4067	8,13%
8	0,0132	0,0401	0,3298	6,60%
9	0,0140	0,0408	0,3438	6,88%
10	0,0131	0,0402	0,3248	6,50%
11	0,0129	0,0390	0,3315	6,63%
12	0,0131	0,0392	0,3356	6,71%
13	0,0120	0,0435	0,2769	5,54%
14	0,0138	0,0454	0,3038	6,08%
15	0,0125	0,0412	0,3044	6,09%
16	0,0151	0,0416	0,3639	7,28%
17	0,0115	0,0397	0,2898	5,80%
18	0,0126	0,0412	0,3054	6,11%
19	0,0120	0,0441	0,2730	5,46%
20	0,0121	0,0418	0,2895	5,79%
21	0,0125	0,0398	0,3133	6,27%
22	0,0122	0,0448	0,2714	5,43%
23	0,0125	0,0409	0,3053	6,11%
24	0,0111	0,0410	0,2710	5,42%
25	0,0139	0,0389	0,3579	7,16%
26	0,0123	0,0391	0,3152	6,30%
27	0,0119	0,0396	0,3016	6,03%

28	0,0128	0,0420	0,3057	6,11%
29	0,0127	0,0407	0,3107	6,21%
30	0,0137	0,0425	0,3236	6,47%

Observando o comparativo dos tempos sequencial e paralelo ilustrados no Gráfico 1, nota-se que para essa carga de trabalho, o tempo paralelo é maior que o sequencial.

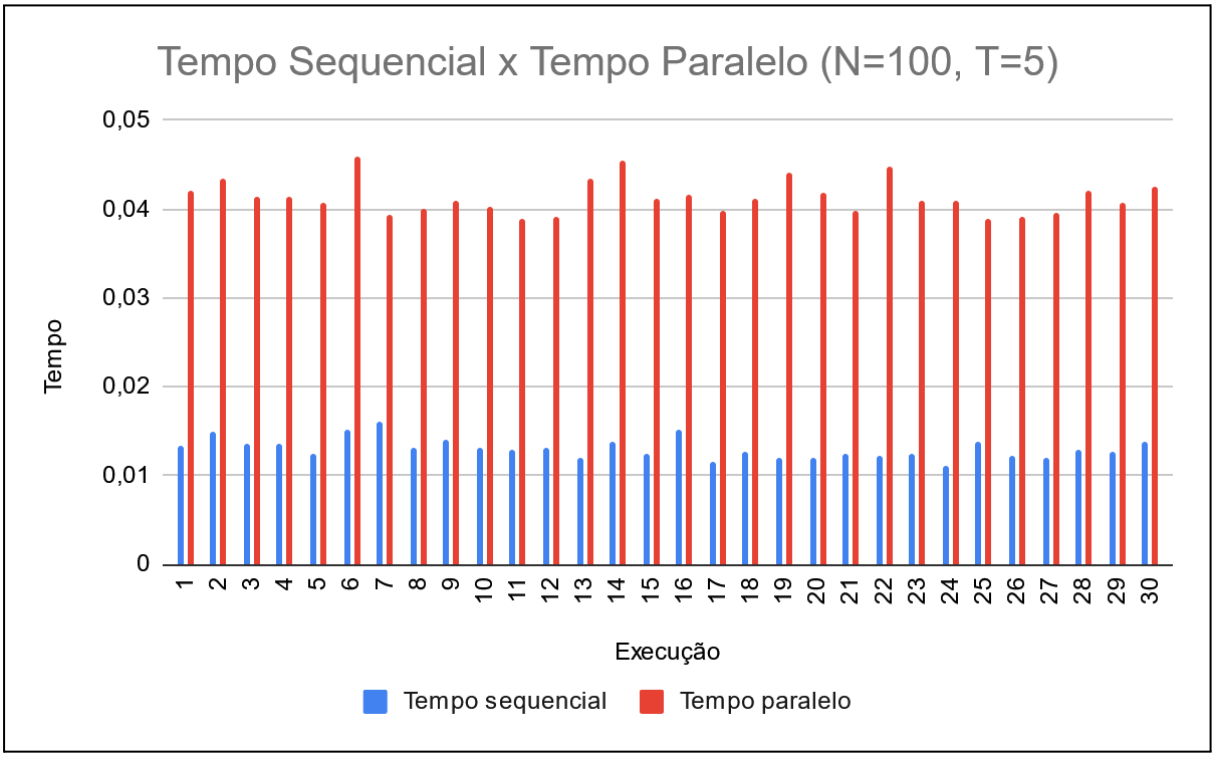

Gráfico 1: Tempo de execução sequencial e paralelo com 5 *threads* para uma matriz de ordem 100.

Para uma matriz da mesma ordem ($N = 100$), com 10 *threads*, o resultado obtido está apresentado na Tabela 2.

Tabela 2: Tempo de execução sequencial e paralelo para uma carga $N = 100$ com $T = 10$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	0,0128	0,0669	0,1909	1,91%
2	0,0132	0,0560	0,2365	2,37%
3	0,0127	0,0575	0,2213	2,21%
4	0,0151	0,0538	0,2803	2,80%
5	0,0122	0,0626	0,1947	1,95%
6	0,0126	0,0512	0,2468	2,47%
7	0,0118	0,0515	0,2285	2,29%
8	0,0123	0,0506	0,2439	2,44%

9	0,0124	0,0625	0,1981	1,98%
10	0,0129	0,0503	0,2572	2,57%
11	0,0109	0,0560	0,1940	1,94%
12	0,0134	0,0483	0,2768	2,77%
13	0,0147	0,0490	0,3000	3,00%
14	0,0126	0,0534	0,2360	2,36%
15	0,0132	0,0606	0,2176	2,18%
16	0,0110	0,0572	0,1915	1,91%
17	0,0122	0,0534	0,2295	2,29%
18	0,0196	0,0487	0,4033	4,03%
19	0,0148	0,0508	0,2917	2,92%
20	0,0124	0,0532	0,2340	2,34%
21	0,0133	0,0533	0,2501	2,50%
22	0,0139	0,0494	0,2812	2,81%
23	0,0152	0,0532	0,2847	2,85%
24	0,0123	0,0508	0,2428	2,43%
25	0,0138	0,0507	0,2724	2,72%
26	0,0149	0,0656	0,2277	2,28%
27	0,0127	0,0566	0,2239	2,24%
28	0,0126	0,0622	0,2028	2,03%
29	0,0137	0,0568	0,2412	2,41%
30	0,0122	0,0498	0,2442	2,44%

Pelo comparativo do tempo sequencial e paralelo contido no Gráfico 2, nota-se que para essa carga de trabalho, com 10 *threads*, o código paralelo continua mais lento que o sequencial, com o adicional de que o tempo paralelo apresentou um aumento em relação ao código executado com 5 *threads*.

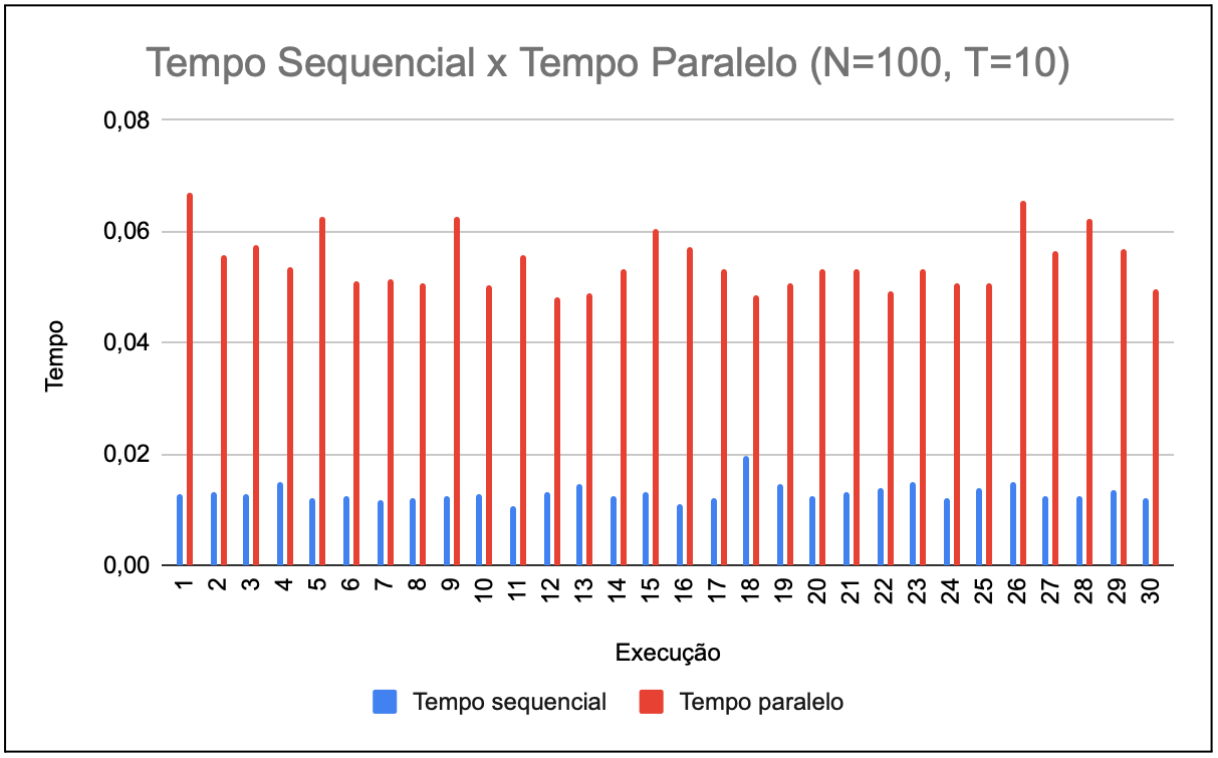

Gráfico 2: Tempo de execução sequencial e paralelo com 10 *threads* para uma matriz de ordem 100.

Por fim, para uma matriz de ordem 100 executando com 20 *threads*, foram obtidos os resultados registrados na Tabela 3.

Tabela 3: Tempo de execução sequencial e paralelo para uma carga $N = 100$ com $T = 20$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	0,0115	0,0703	0,1635	0,82%
2	0,0147	0,0709	0,2071	1,04%
3	0,0142	0,0713	0,1990	1,00%
4	0,0158	0,0721	0,2193	1,10%
5	0,0130	0,0699	0,1863	0,93%
6	0,0129	0,0713	0,1811	0,91%
7	0,0161	0,0697	0,2314	1,16%
8	0,0137	0,0695	0,1965	0,98%
9	0,0106	0,0702	0,1515	0,76%
10	0,0128	0,0682	0,1881	0,94%
11	0,0145	0,0711	0,2043	1,02%
12	0,0132	0,0681	0,1942	0,97%
13	0,0123	0,0771	0,1595	0,80%
14	0,0146	0,0703	0,2071	1,04%

15	0,0123	0,0706	0,1746	0,87%
16	0,0157	0,0701	0,2239	1,12%
17	0,0128	0,0700	0,1825	0,91%
18	0,0122	0,0748	0,1632	0,82%
19	0,0150	0,0730	0,2052	1,03%
20	0,0125	0,0707	0,1772	0,89%
21	0,0122	0,0707	0,1728	0,86%
22	0,0110	0,0683	0,1616	0,81%
23	0,0125	0,0687	0,1823	0,91%
24	0,0126	0,0688	0,1835	0,92%
25	0,0124	0,0697	0,1782	0,89%
26	0,0125	0,0728	0,1711	0,86%
27	0,0149	0,0747	0,1990	1,00%
28	0,0125	0,0771	0,1614	0,81%
29	0,0130	0,0698	0,1860	0,93%
30	0,0139	0,0713	0,1944	0,97%

Conforme o Gráfico 3, o tempo do algoritmo paralelo continua sendo maior que o sequencial, e ocorre mais um aumento no tempo com o acréscimo de novas threads.

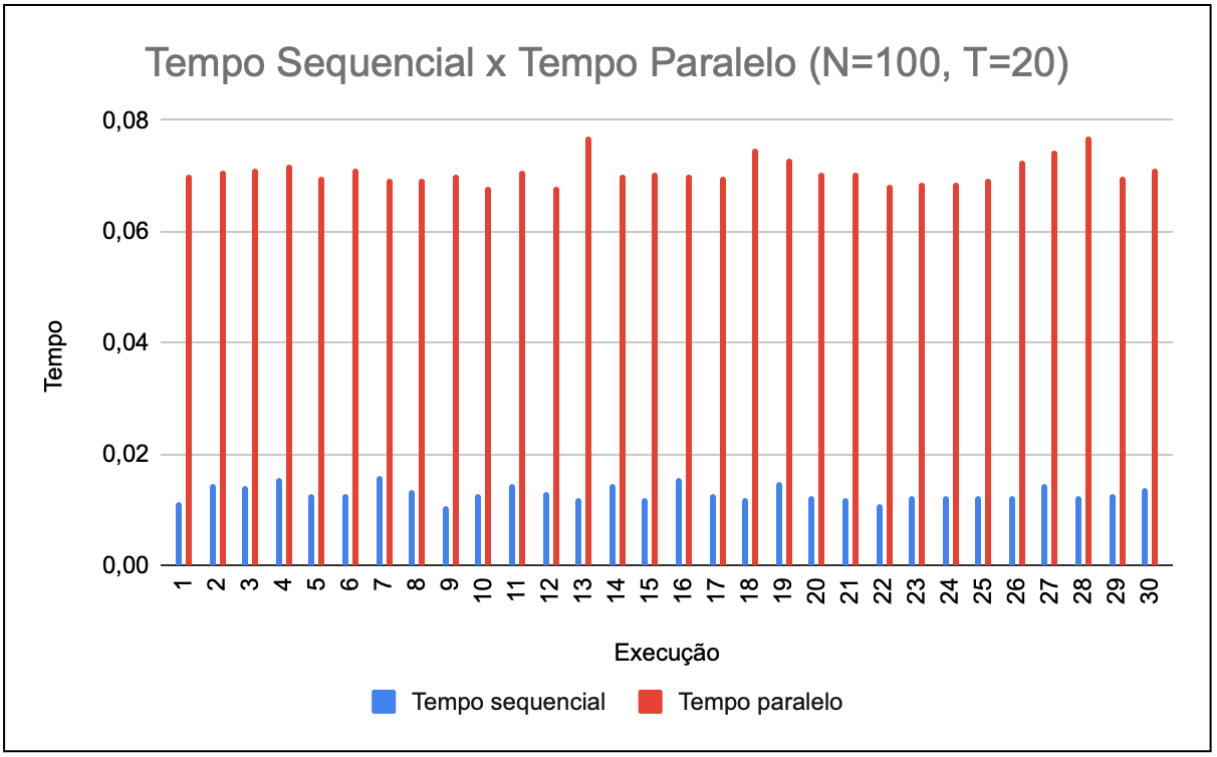

Gráfico 3: Tempo de execução sequencial e paralelo com 20 *threads* para uma matriz de ordem 100.

A Tabela 4 contém um compilado dos resultados obtidos: o tempo médio de execução dos códigos sequencial e paralelo para as três quantidades de threads, com as respectivas medianas e desvios padrão.

Tabela 4: Speedup, eficiência, tempo médio de execução, desvio padrão e mediana dos códigos sequencial e paralelo para uma carga $N = 100$ com $T = 5$, $T = 10$ e $T = 20$

t	jacobiseq	mediana	desvio padrão	jacobipar	mediana	desvio padrão	speedup	eficiência
5	0,0131	0,0129	0,0011	0,0414	0,0411	0,0019	0,3166	6,33%
10	0,0133	0,0128	0,0016	0,0547	0,0533	0,0052	0,2421	2,42%
20	0,0133	0,0129	0,0014	0,0710	0,0705	0,0023	0,1867	0,93%

O Gráfico 4 apresenta a evolução do tempo de execução para uma matriz de ordem 100. Observa-se a tendência, apontada anteriormente, de aumento do tempo de execução do código paralelo conforme o acréscimo de novas *threads*, o que levou à queda no *speedup* (registrada no Gráfico 5) e, consequentemente, a uma perda na eficiência (registrada no Gráfico 6).

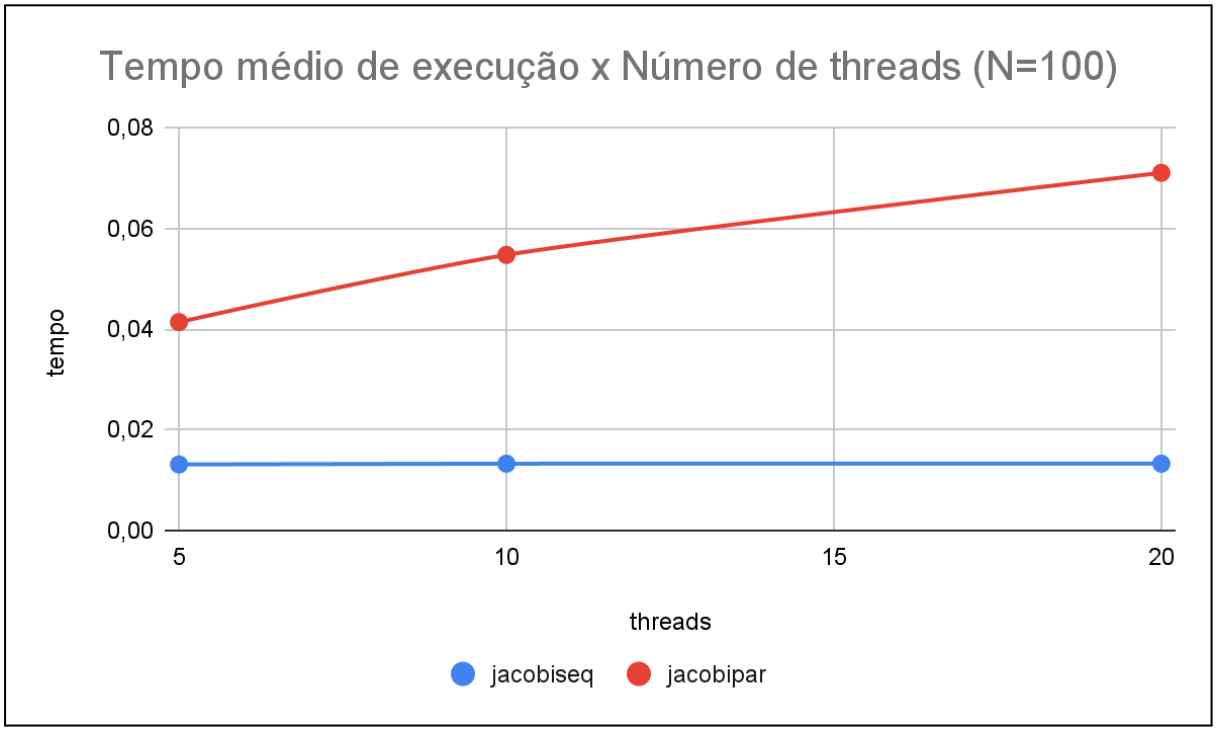

Gráfico 4: Tempo de execução sequencial e paralelo para uma matriz de ordem 100 com 5, 10 e 20 *threads*.

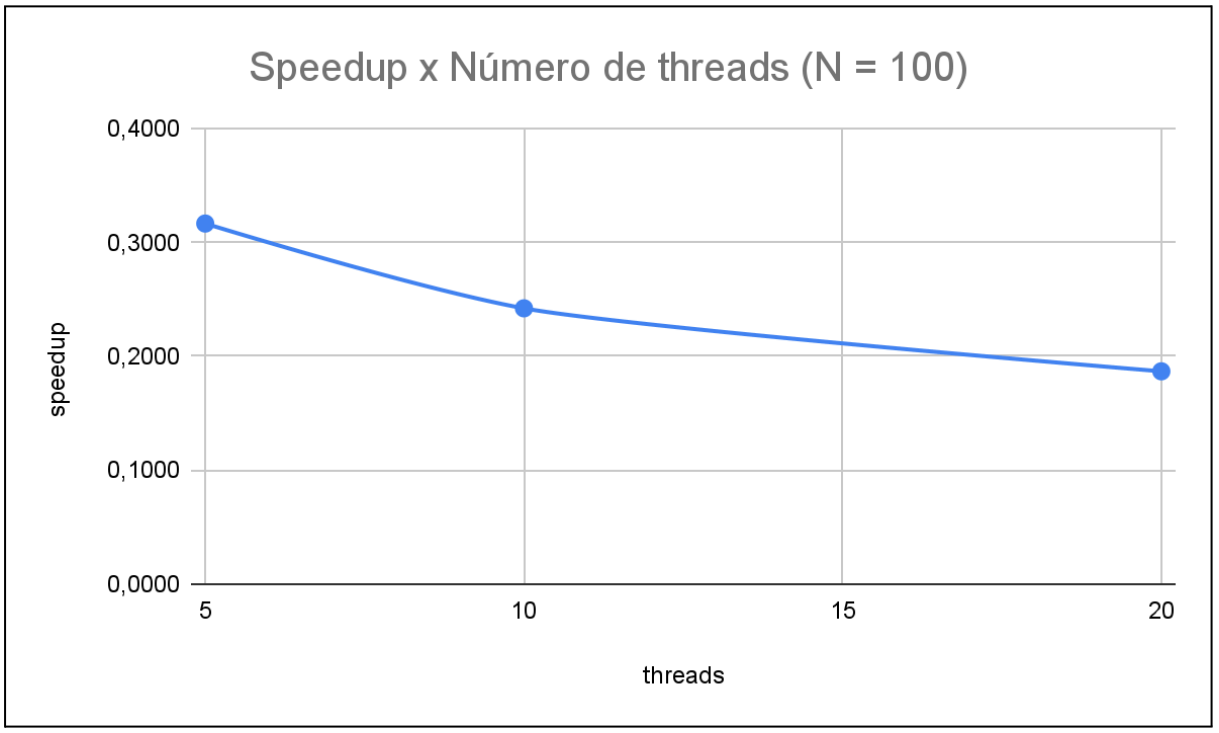

Gráfico 5: Speedup para uma matriz de ordem 100 com 5, 10, e 20 *threads*.

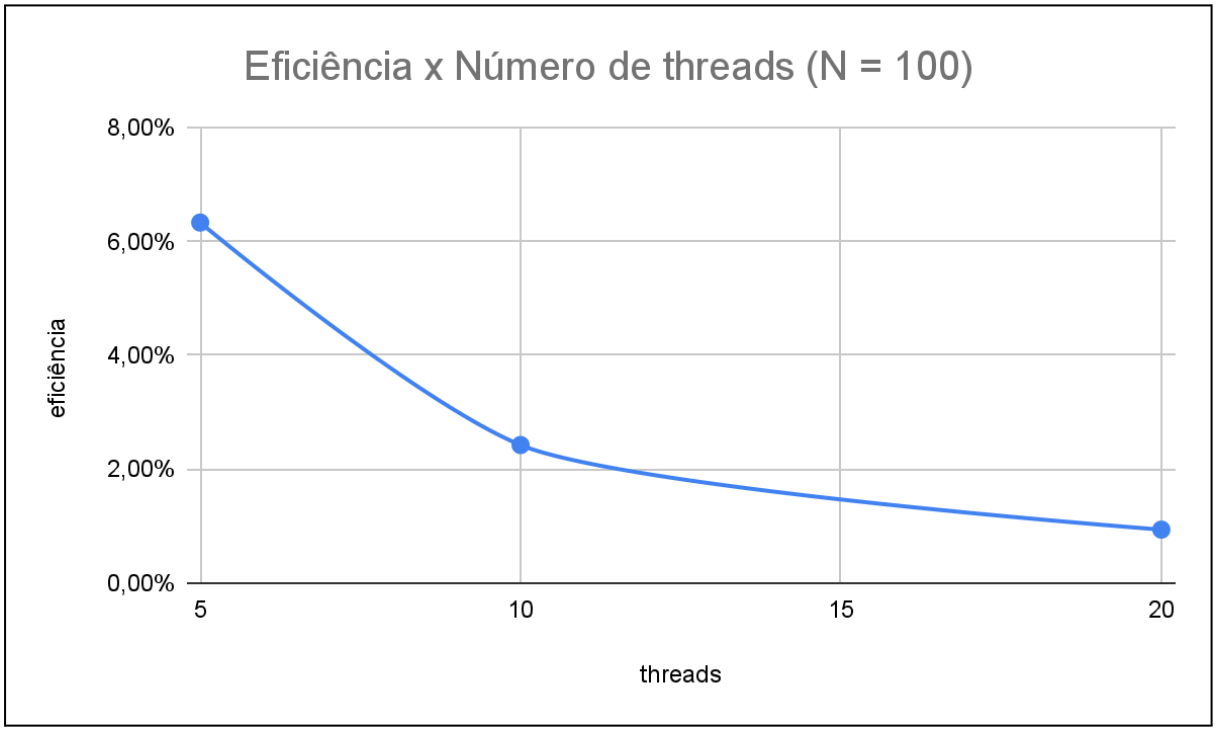

Gráfico 6: Eficiência para uma matriz de ordem 100 com 5, 10, e 20 *threads*.

Os resultados obtidos evidenciam que para uma menor carga de trabalho, os custos de paralelização são mais altos que o ganho de desempenho, possivelmente devido ao

overhead de sincronização, dada a necessidade de coordenar e sincronizar as operações entre as threads sem que haja uma carga pesada o suficiente para diluir esses custos.

Matriz de ordem 1024

Para uma matriz de ordem 1024, com 5 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 5, abaixo:

Tabela 5: Tempo de execução sequencial e paralelo para uma carga $N = 1024$ com $T = 5$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	1,4901	0,7219	2,0642	41,28%
2	1,4340	0,6895	2,0797	41,59%
3	1,4807	0,6675	2,2183	44,37%
4	1,4050	0,6634	2,1178	42,36%
5	1,4158	0,6779	2,0884	41,77%
6	1,4307	0,6900	2,0735	41,47%
7	1,4915	0,7517	1,9841	39,68%
8	1,4792	0,6913	2,1398	42,80%
9	1,4558	0,7203	2,0211	40,42%
10	1,4282	0,6978	2,0467	40,93%
11	1,4032	0,7236	1,9391	38,78%
12	1,4669	0,7686	1,9086	38,17%
13	1,4427	0,7926	1,8202	36,40%
14	1,4176	0,7062	2,0072	40,14%
15	1,4602	0,7386	1,9769	39,54%
16	1,2739	0,7367	1,7291	34,58%
17	1,2222	0,8004	1,5270	30,54%
18	1,2020	0,7420	1,6200	32,40%
19	1,5066	0,7289	2,0671	41,34%
20	1,5923	0,7453	2,1365	42,73%
21	1,4895	0,7470	1,9941	39,88%
22	1,5063	0,6953	2,1665	43,33%
23	1,4619	0,6885	2,1232	42,46%
24	1,4813	0,7036	2,1051	42,10%
25	1,4676	0,6638	2,2109	44,22%

26	1,5493	0,6645	2,3316	46,63%
27	1,4697	0,6811	2,1578	43,16%
28	1,4467	0,6931	2,0871	41,74%
29	1,5258	0,6807	2,2413	44,83%
30	1,5747	0,7576	2,0787	41,57%

Para uma matriz dessa ordem, com 5 *threads* rodando, já se nota um melhor resultado do tempo paralelo em relação ao tempo sequencial, como é possível observar no Gráfico 7.

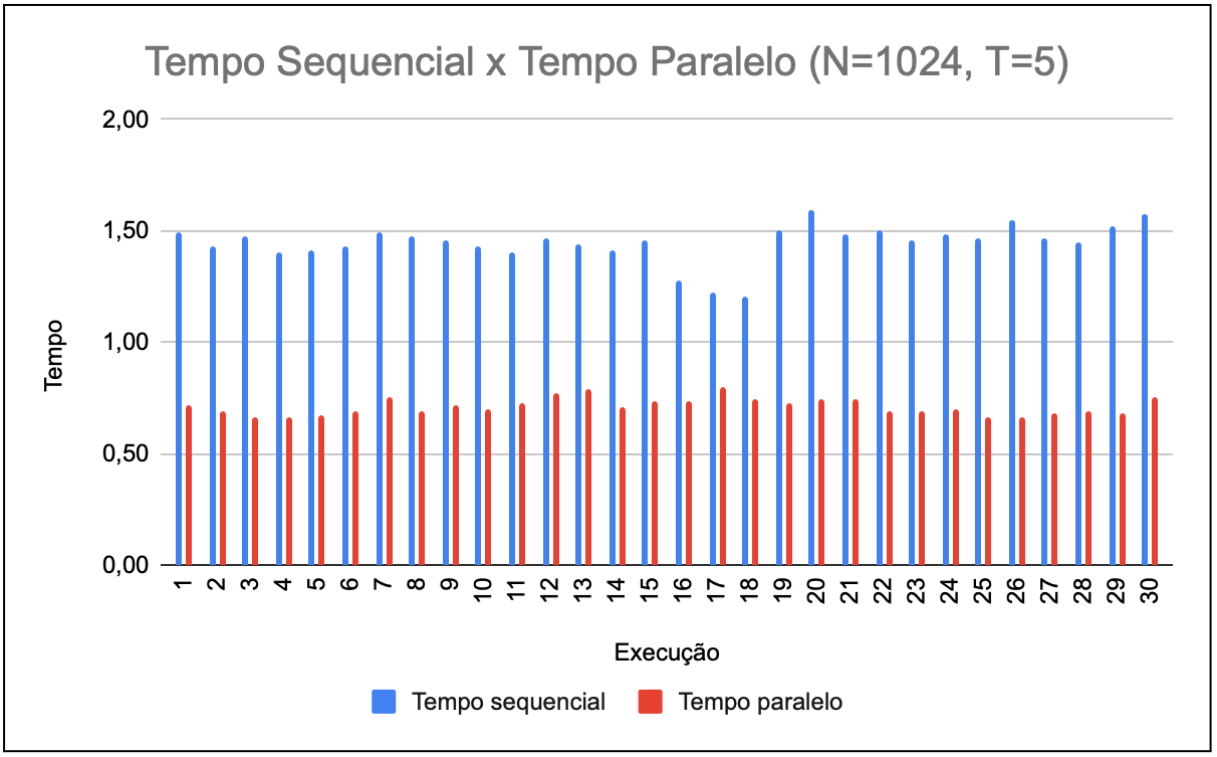

Gráfico 7: Tempo de execução sequencial e paralelo com 5 *threads* para uma matriz de ordem 1024.

Para uma matriz de ordem 1024, com 10 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 6, abaixo:

Tabela 6: Tempo de execução sequencial e paralelo para uma carga $N = 1024$ com $T = 10$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	1,4342	0,7904	1,8146	18,15%
2	1,2277	0,9244	1,3281	13,28%
3	1,4824	0,8042	1,8434	18,43%
4	1,5646	0,7422	2,1080	21,08%
5	1,6203	0,6662	2,4321	24,32%

6	1,4648	0,6599	2,2198	22,20%
7	1,5466	0,6867	2,2522	22,52%
8	1,4849	0,7554	1,9658	19,66%
9	1,5334	0,6768	2,2657	22,66%
10	1,5195	0,7281	2,0869	20,87%
11	1,5190	0,6887	2,2054	22,05%
12	1,6930	0,7929	2,1351	21,35%
13	1,6700	0,6883	2,4263	24,26%
14	1,5251	0,6867	2,2209	22,21%
15	1,5367	0,6512	2,3598	23,60%
16	1,5415	0,6022	2,5599	25,60%
17	1,4869	0,6139	2,4222	24,22%
18	1,4728	0,6665	2,2099	22,10%
19	1,4834	0,7019	2,1132	21,13%
20	1,4734	0,6881	2,1413	21,41%
21	1,4648	0,6277	2,3337	23,34%
22	1,5315	0,6400	2,3929	23,93%
23	1,5170	0,6757	2,2451	22,45%
24	1,5859	0,6548	2,4219	24,22%
25	1,5208	0,6096	2,4949	24,95%
26	1,5306	0,5942	2,5759	25,76%
27	1,4858	0,6592	2,2540	22,54%
28	1,4960	0,6153	2,4312	24,31%
29	1,6348	0,6630	2,4658	24,66%
30	1,6128	0,6231	2,5885	25,89%

Para uma matriz dessa ordem, com 10 *threads* trabalhando, o tempo paralelo segue apresentando melhores resultados em relação ao tempo sequencial, como é possível observar no Gráfico 8. A adição de *threads* também culminou na diminuição do tempo paralelo em relação ao desempenho com 5 *threads*.

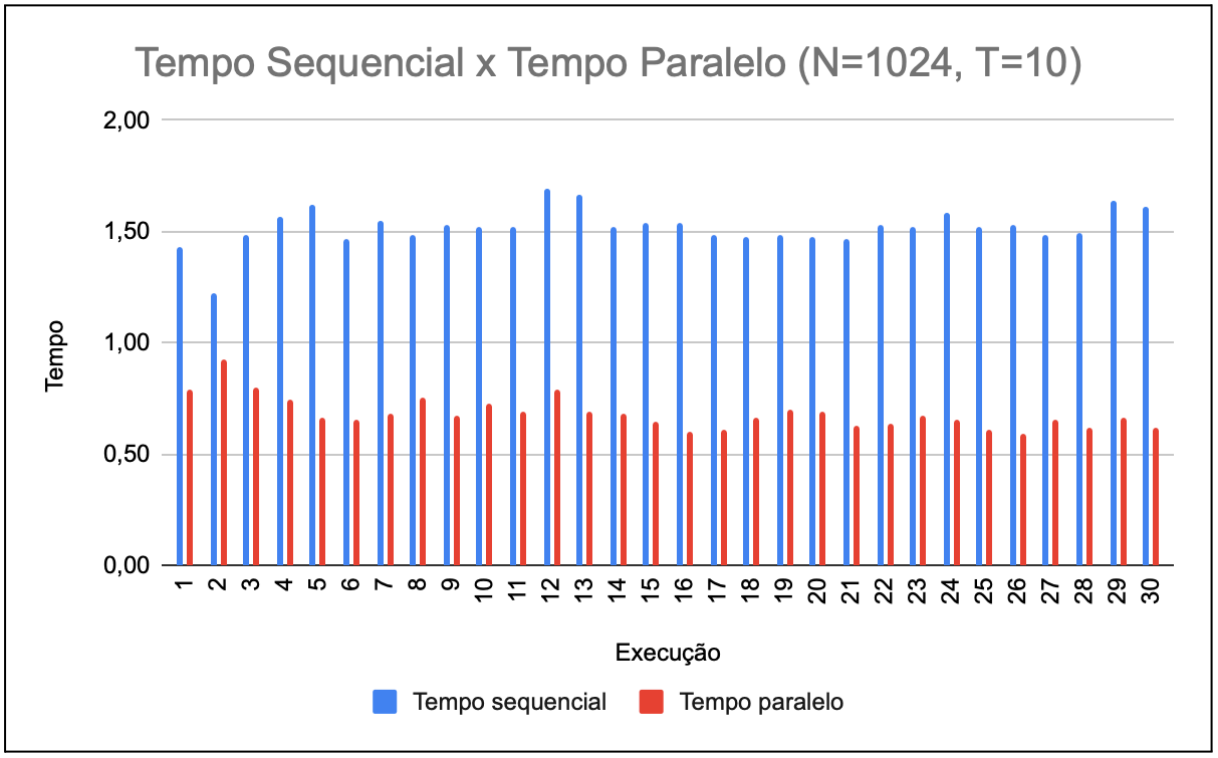

Gráfico 8: Tempo de execução sequencial e paralelo com 10 *threads* para uma matriz de ordem 1024.

Para uma matriz de ordem 1024, com 20 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 7, abaixo:

Tabela 7: Tempo de execução sequencial e paralelo para uma carga $N = 1024$ com $T = 20$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	1,4119	0,669716	2,1082	10,54%
2	1,4597	0,647327	2,2549	11,27%
3	1,4612	0,652971	2,2378	11,19%
4	1,4400	0,648595	2,2203	11,10%
5	1,4418	0,626541	2,3013	11,51%
6	1,4626	0,647169	2,2601	11,30%
7	1,4426	0,696978	2,0698	10,35%
8	1,4714	0,759863	1,9363	9,68%
9	1,4666	0,676911	2,1666	10,83%
10	1,4702	0,644479	2,2812	11,41%
11	1,3766	0,678231	2,0297	10,15%
12	1,5045	0,639987	2,3509	11,75%
13	1,4998	0,622579	2,4090	12,05%

14	1,4133	0,622105	2,2718	11,36%
15	1,5383	0,628246	2,4485	12,24%
16	1,4574	0,637987	2,2843	11,42%
17	1,5290	0,650773	2,3496	11,75%
18	1,4492	0,649844	2,2301	11,15%
19	1,4885	0,642335	2,3174	11,59%
20	1,3830	0,678522	2,0382	10,19%
21	1,4702	0,644769	2,2802	11,40%
22	1,5302	0,645718	2,3698	11,85%
23	1,6139	0,735291	2,1949	10,97%
24	1,4945	0,679012	2,2010	11,01%
25	1,5123	0,656324	2,3041	11,52%
26	1,6106	0,638424	2,5228	12,61%
27	1,6113	0,698007	2,3085	11,54%
28	1,4822	0,632298	2,3441	11,72%
29	1,5456	0,652323	2,3694	11,85%
30	1,4925	0,665093	2,2440	11,22%

Para uma matriz dessa ordem, com 20 *threads* rodando, o tempo paralelo continua apresentando melhor desempenho do que o tempo sequencial, como se observa no Gráfico 9. A adição de *threads* também resultou em uma diminuição do tempo paralelo em relação ao desempenho com 10 *threads*.

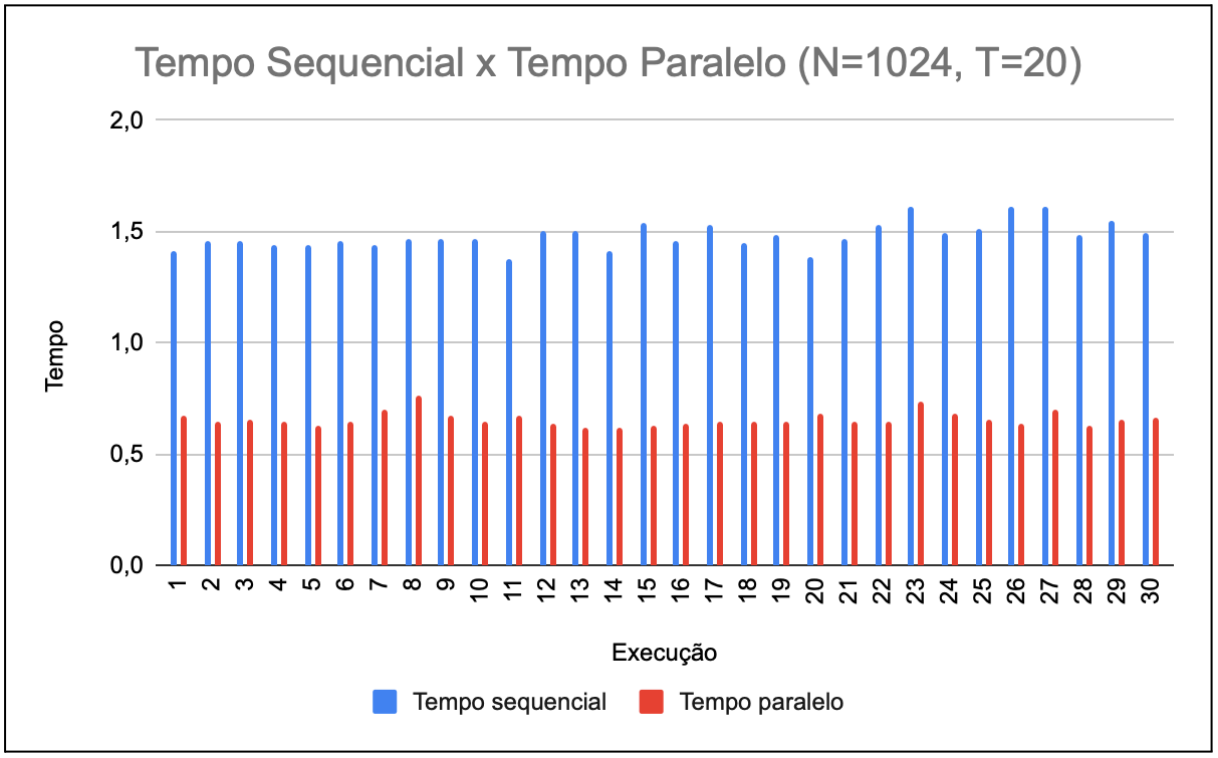

Gráfico 9: Tempo de execução sequencial e paralelo com 20 *threads* para uma matriz de ordem 1024.

A Tabela 8 contém os resultados obtidos: o tempo médio de execução dos códigos sequencial e paralelo para as três quantidades de *threads*, com a respectiva mediana e desvio padrão, para uma matriz de ordem 1024.

Tabela 8: *Speedup*, eficiência, tempo médio de execução, desvio padrão e mediana dos códigos sequencial e paralelo para uma carga $N = 1024$ com $T = 5$, $T = 10$ e $T = 20$

t	jacobiseq	mediana	desvio padrão	jacobipar	mediana	desvio padrão	speedup	eficiência
5	1,4490	1,4644	0,0868	0,7143	0,7049	0,0376	2,0286	40,57%
10	1,5220	1,5201	0,0836	0,6711	0,6711	0,0717	2,2680	22,68%
20	1,4844	1,4708	0,0597	0,6589	0,6492	0,0315	2,2526	11,26%

O Gráfico 10 apresenta a evolução do tempo de execução para uma matriz de ordem 1024. Observa-se a tendência, apontada anteriormente, de diminuição do tempo de execução do código paralelo conforme o acréscimo de novas *threads*, o que levou a um *speedup* sublinear (observável no Gráfico 11) e a uma eficiência decrescente (registrada no Gráfico 12).

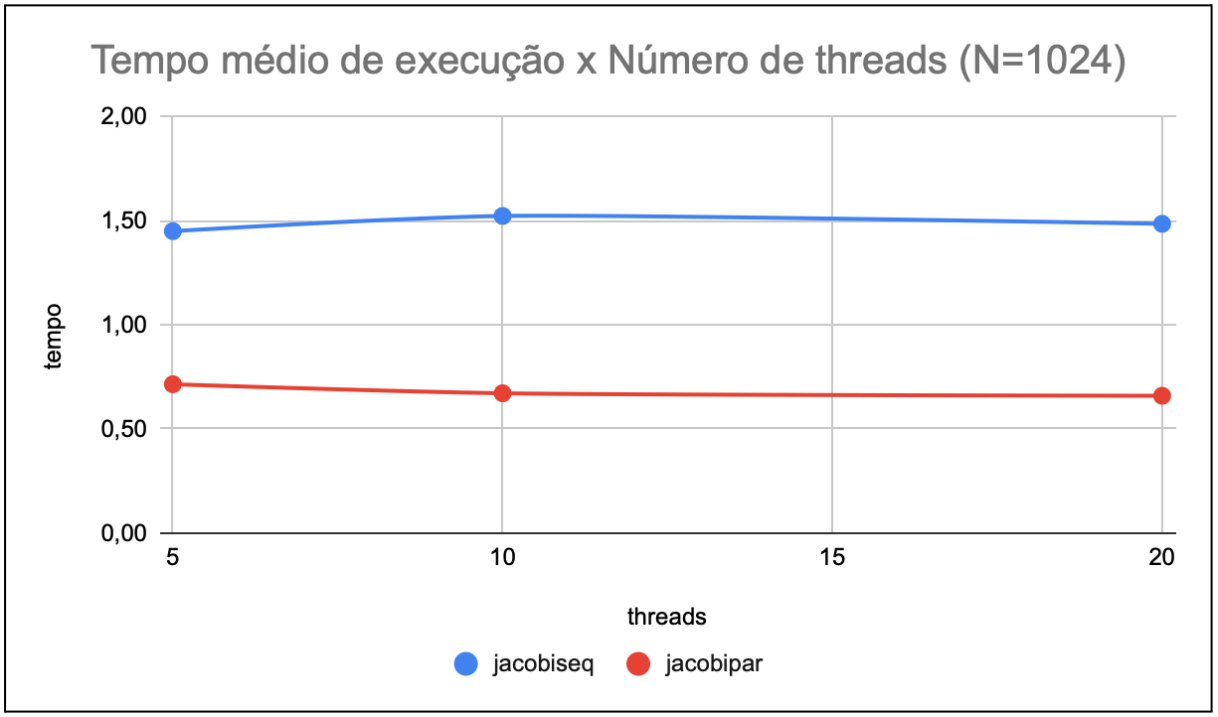

Gráfico 10: Tempo de execução sequencial e paralelo para uma matriz de ordem 1024 com 5, 10 e 20 *threads*.

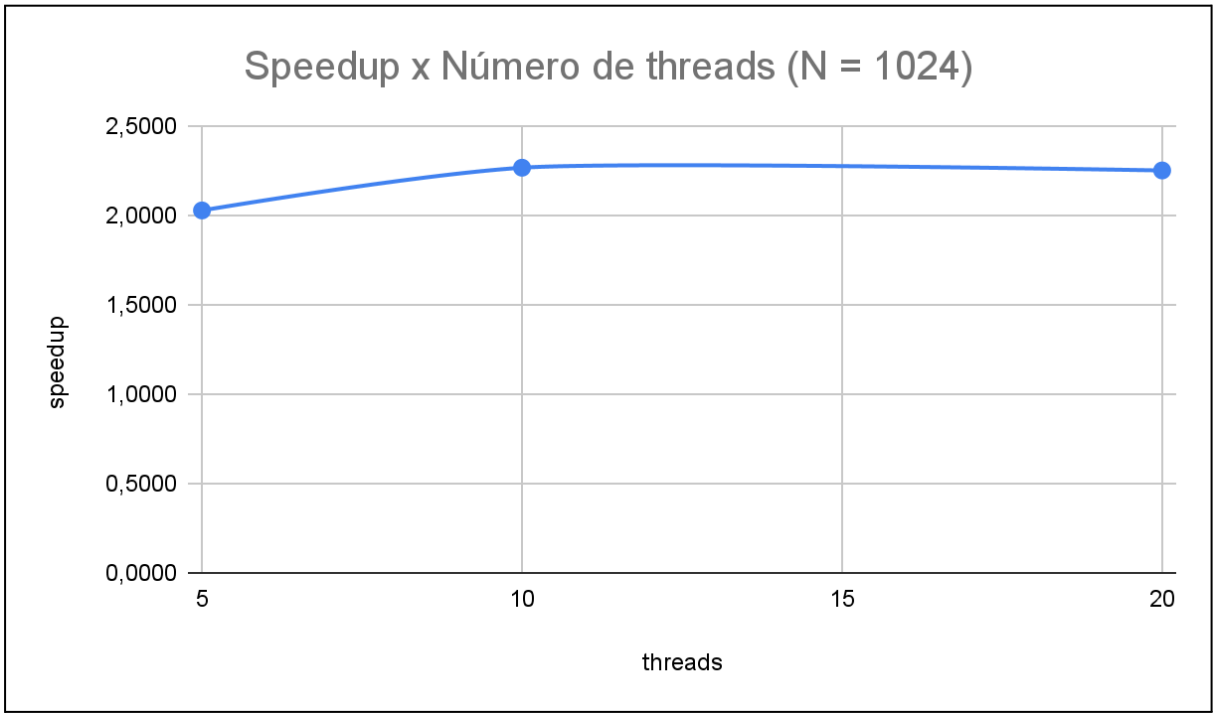

Gráfico 11: *Speedup* para uma matriz de ordem 1024 com 5, 10, e 20 *threads*.

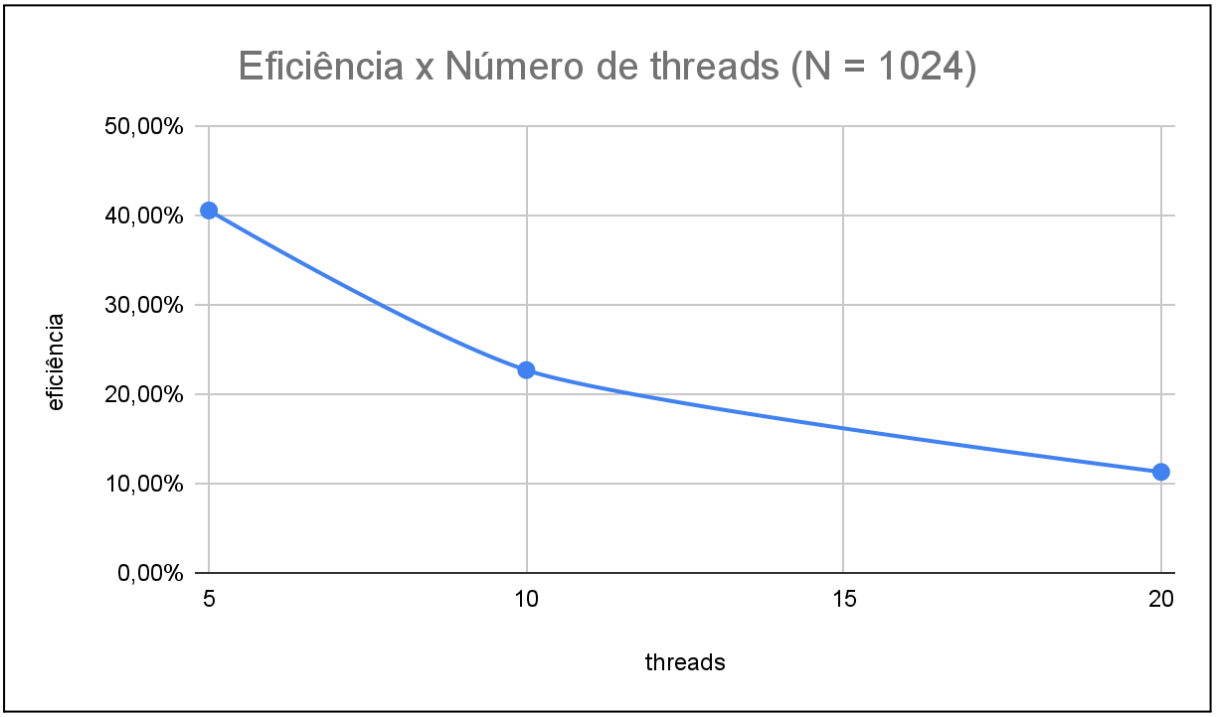

Gráfico 12: Eficiência para uma matriz de ordem 1024 com 5, 10, e 20 *threads*.

Nota-se que um aumento no *speedup* (Gráfico 11) não é acompanhado por um aumento proporcional de eficiência (Gráfico 12) ao se adicionar novas *threads*, o que pode indicar questões de escalabilidade no código paralelo.

Matriz de ordem 10240

Para uma matriz de ordem 10240, com 5 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 9, abaixo:

Tabela 9: Tempo de execução sequencial e paralelo para uma carga $N = 10240$ com $T = 5$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	159,2371	56,9356	2,7968	55,94%
2	159,7204	57,0004	2,8021	56,04%
3	155,8356	58,8230	2,6492	52,98%
4	155,2578	60,5379	2,5646	51,29%
5	155,4430	58,1796	2,6718	53,44%
6	159,0586	58,3461	2,7261	54,52%
7	156,2259	59,2787	2,6354	52,71%
8	152,1107	58,1688	2,6150	52,30%

9	154,4985	59,8680	2,5807	51,61%
10	159,0131	59,0338	2,6936	53,87%
11	154,6061	58,1568	2,6584	53,17%
12	156,9574	58,3898	2,6881	53,76%
13	155,5669	60,9121	2,5540	51,08%
14	159,2435	59,1186	2,6936	53,87%
15	158,2100	58,6785	2,6962	53,92%
16	145,3942	58,5860	2,4817	49,63%
17	158,4720	58,7605	2,6969	53,94%
18	159,3061	59,2217	2,6900	53,80%
19	158,2694	59,4901	2,6604	53,21%
20	155,4550	59,2677	2,6229	52,46%
21	156,8001	59,2008	2,6486	52,97%
22	157,2427	59,5619	2,6400	52,80%
23	152,3571	59,2534	2,5713	51,43%
24	154,4218	59,4281	2,5985	51,97%
25	156,9015	60,5800	2,5900	51,80%
26	156,8710	58,7229	2,6714	53,43%
27	154,6374	60,0059	2,5770	51,54%
28	151,9793	57,9473	2,6227	52,45%
29	152,8295	58,3639	2,6186	52,37%
30	158,9263	58,5620	2,7138	54,28%

Observando o comparativo dos tempos sequencial e paralelo ilustrados no Gráfico 13, nota-se que para essa carga de trabalho, o código paralelo apresenta melhores resultados do que o código sequencial.

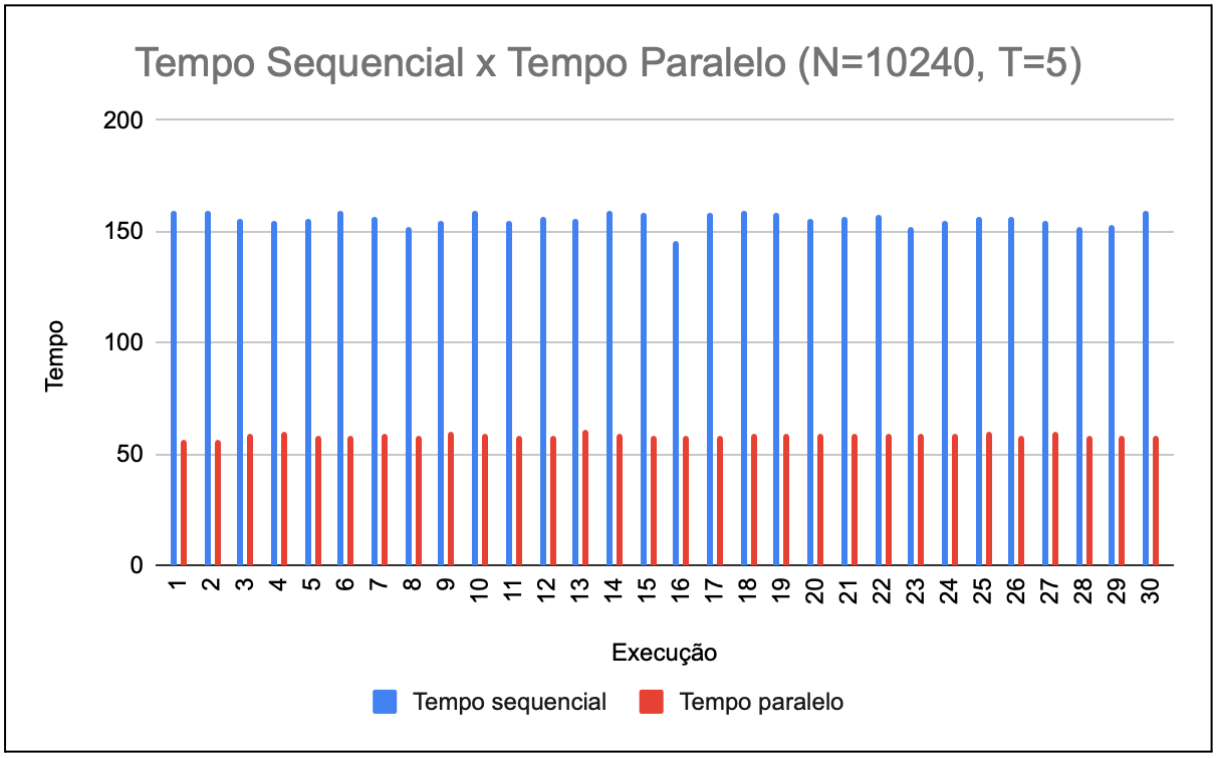

Gráfico 13: Tempo de execução sequencial e paralelo com 5 *threads* para uma matriz de ordem 10240.

Para uma matriz de ordem 10240, com 10 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 10, abaixo:

Tabela 10: Tempo de execução sequencial e paralelo para uma carga $N = 10240$ com $T = 10$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	151,6787	38,8422	3,9050	39,05%
2	149,3997	38,5233	3,8782	38,78%
3	144,6624	38,7036	3,7377	37,38%
4	151,6879	38,3031	3,9602	39,60%
5	144,1807	38,6603	3,7294	37,29%
6	144,9743	38,8201	3,7345	37,35%
7	145,5961	37,8955	3,8420	38,42%
8	148,7814	38,8060	3,8340	38,34%
9	147,4248	38,8608	3,7937	37,94%
10	147,1131	38,8057	3,7910	37,91%
11	146,1368	38,6572	3,7803	37,80%
12	147,1387	38,7122	3,8008	38,01%
13	149,4291	38,8382	3,8475	38,47%

14	148,8615	38,1364	3,9034	39,03%
15	144,5774	38,3793	3,7671	37,67%
16	146,3502	38,5536	3,7960	37,96%
17	150,2276	38,6000	3,8919	38,92%
18	145,9856	37,2528	3,9188	39,19%
19	144,7653	38,2707	3,7827	37,83%
20	147,7755	38,0676	3,8819	38,82%
21	145,9561	38,2103	3,8198	38,20%
22	145,3889	38,5071	3,7756	37,76%
23	144,6261	38,1373	3,7922	37,92%
24	141,9533	38,4297	3,6938	36,94%
25	143,3152	38,5824	3,7145	37,15%
26	144,1522	38,2416	3,7695	37,70%
27	141,4419	38,5414	3,6699	36,70%
28	149,1419	38,1983	3,9044	39,04%
29	151,9134	38,1029	3,9869	39,87%
30	152,4292	38,0333	4,0078	40,08%

Para uma matriz dessa ordem, com 10 *threads* rodando, o tempo paralelo continua apresentando melhor resultado do que o tempo sequencial, como se observa no Gráfico 14. A adição de *threads* traz uma pequena redução no tempo paralelo em relação ao desempenho com 5 *threads*.

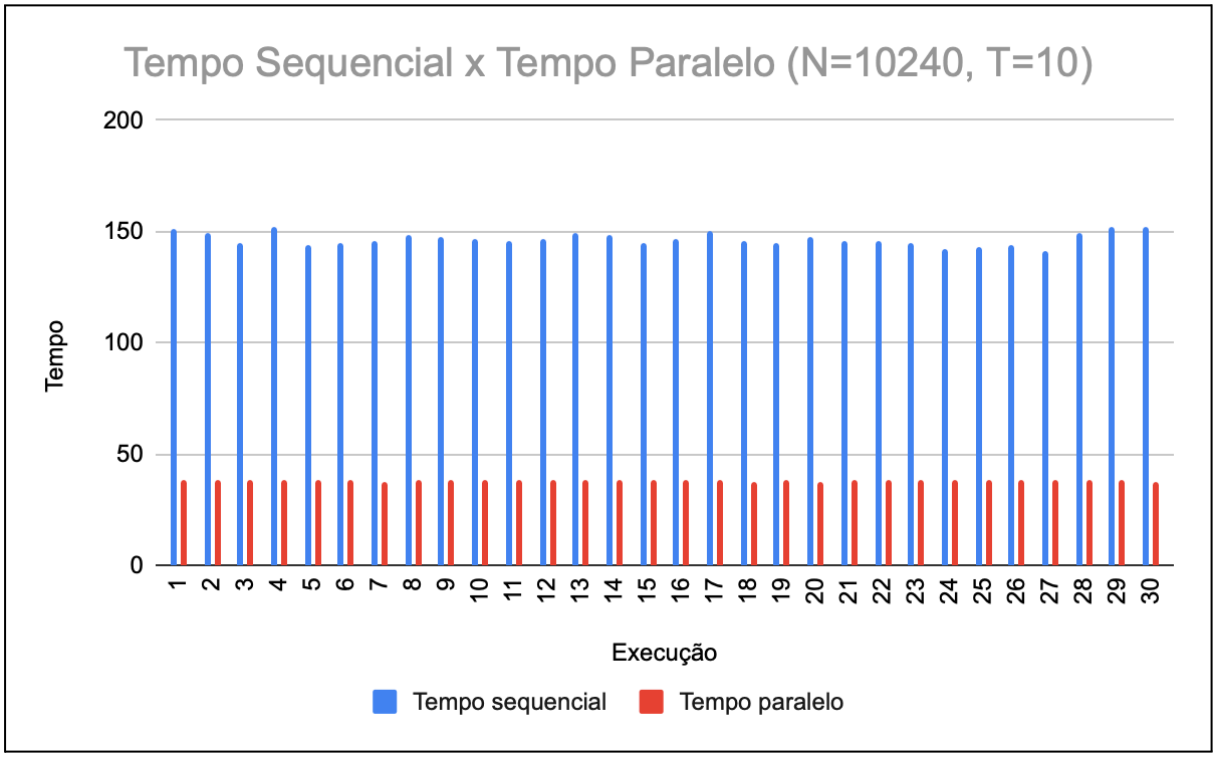

Gráfico 14: Tempo de execução sequencial e paralelo com 10 *threads* para uma matriz de ordem 10240.

Para uma matriz de ordem 10240, com 20 *threads*, observou-se, para 30 execuções, o resultado registrado na Tabela 11, a seguir:

Tabela 11: Tempo de execução sequencial e paralelo para uma carga $N = 10240$ com $T = 20$

Execução	Tempo sequencial	Tempo paralelo	Speedup	Eficiência
1	148,8773	36,4483	4,0846	20,42%
2	149,9960	35,5515	4,2191	21,10%
3	152,9508	35,4122	4,3192	21,60%
4	150,2030	35,4626	4,2355	21,18%
5	140,5752	35,4392	3,9667	19,83%
6	145,4062	36,2383	4,0125	20,06%
7	151,8198	35,4879	4,2781	21,39%
8	152,0719	36,7780	4,1349	20,67%
9	154,3242	36,3094	4,2503	21,25%
10	152,6766	35,3368	4,3206	21,60%
11	152,6078	35,8529	4,2565	21,28%
12	152,9946	35,2757	4,3371	21,69%
13	154,0840	35,6581	4,3212	21,61%

14	156,6798	35,9349	4,3601	21,80%
15	153,4757	35,4141	4,3337	21,67%
16	147,4733	35,8857	4,1095	20,55%
17	152,7105	35,6463	4,2840	21,42%
18	152,5315	35,5358	4,2923	21,46%
19	145,5302	35,5318	4,0958	20,48%
20	145,8034	35,6671	4,0879	20,44%
21	140,6229	35,6237	3,9475	19,74%
22	148,1347	35,4934	4,1736	20,87%
23	149,4649	35,8956	4,1639	20,82%
24	146,4752	35,6430	4,1095	20,55%
25	142,8904	35,6410	4,0092	20,05%
26	148,1913	35,7155	4,1492	20,75%
27	149,5815	35,5338	4,2096	21,05%
28	146,4019	35,6920	4,1018	20,51%
29	146,2979	35,5552	4,1147	20,57%
30	143,0685	35,5261	4,0271	20,14%

Com 20 *threads* rodando, o tempo paralelo continua apresentando melhor resultado do que o tempo sequencial, como se observa no Gráfico 15. A adição de *threads* traz uma tímida redução no tempo paralelo em relação ao desempenho com 10 *threads*.

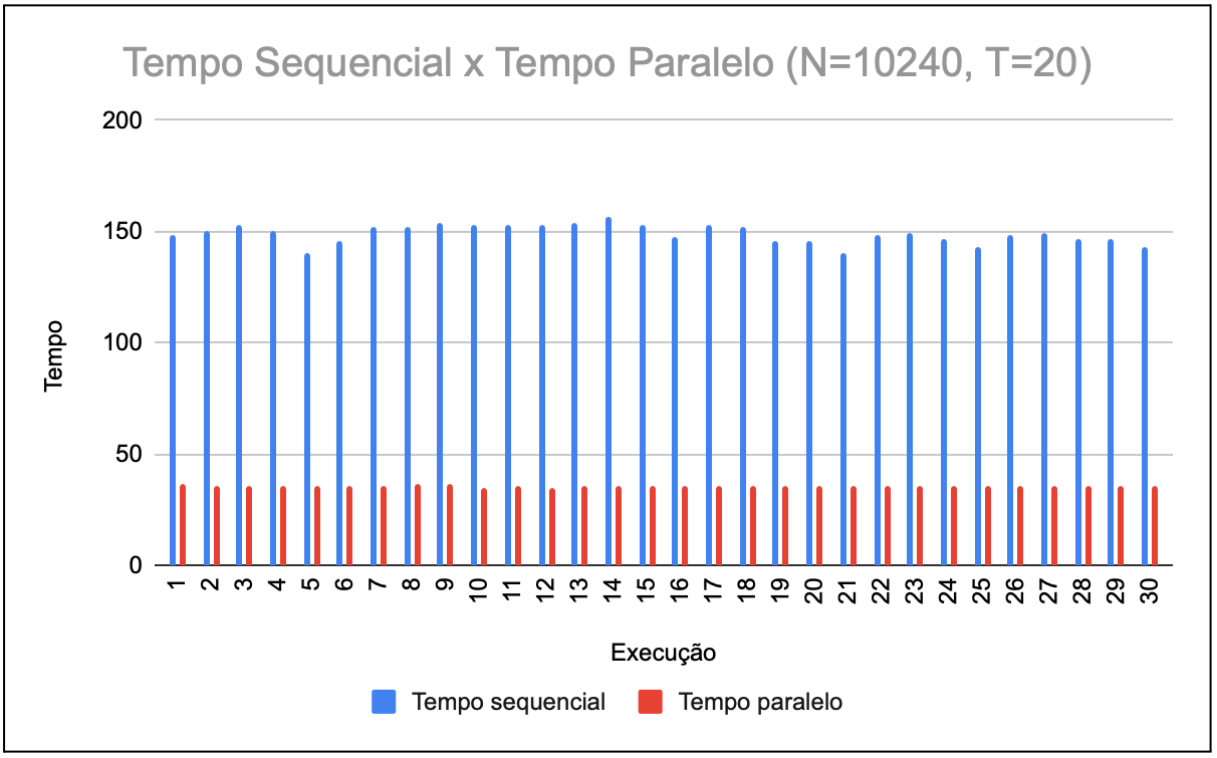

Gráfico 15: Tempo de execução sequencial e paralelo com 20 *threads* para uma matriz de ordem 10240.

A Tabela 12 resume os resultados obtidos: o tempo médio de execução dos códigos sequencial e paralelo para as três quantidades de *threads*, com a respectiva mediana e desvio padrão, para uma matriz de ordem 10240.

Tabela 12: Speedup, eficiência, tempo médio de execução, desvio padrão e mediana dos códigos sequencial e paralelo para uma carga $N = 10240$ com $T = 5$, $T = 10$ e $T = 20$

t	jacobiseq	mediana	desvio padrão	jacobipar	mediana	desvio padrão	speedup	eficiência
5	154,6949	155,4490	3,5183	58,9460	58,9284	0,9198	2,6243	52,49%
10	146,9022	146,2435	2,9433	38,4224	38,5152	0,3586	3,8233	38,23%
20	149,1307	149,5232	4,1953	35,7062	35,6323	0,3422	4,1766	20,88%

O Gráfico 16 apresenta a evolução do tempo de execução para uma matriz de ordem 10240. Observa-se novamente um *speedup* sublinear (observável no Gráfico 17) e uma eficiência decrescente (registrada no Gráfico 18). Para a maior dimensão da matriz, o código paralelo apresentou o melhor *speedup* e a melhor eficiência, na média das execuções com 5 *threads*.

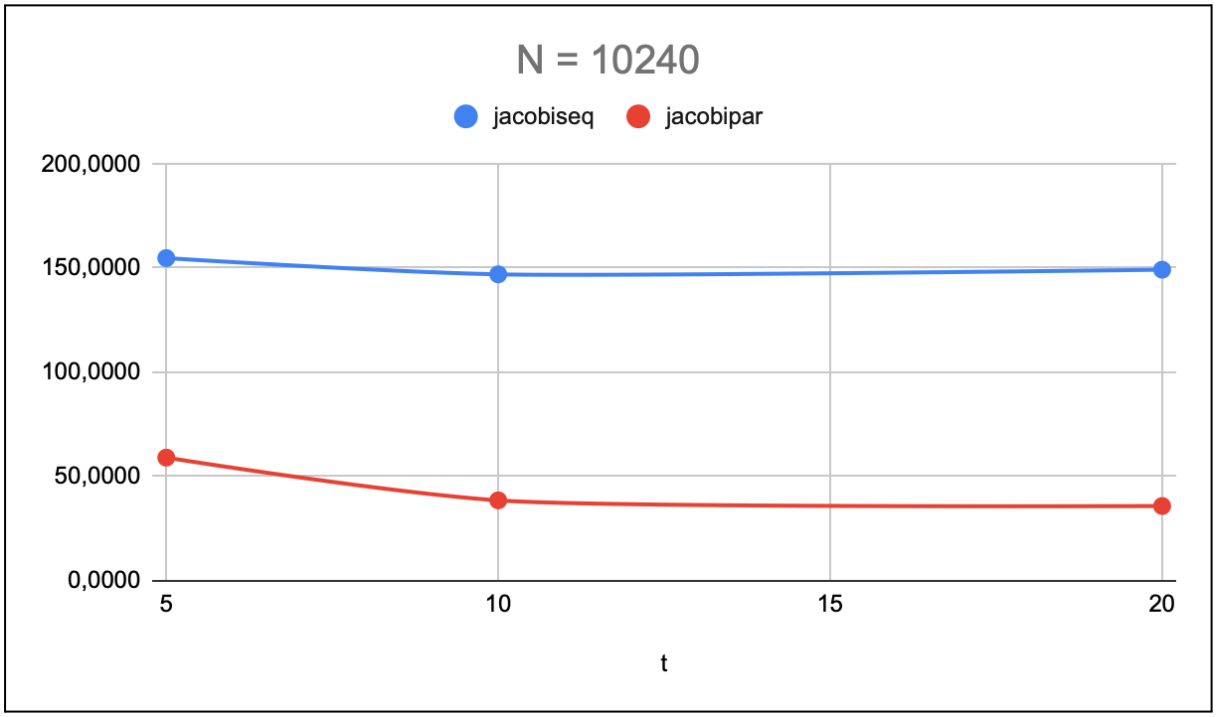

Gráfico 16: Tempo de execução sequencial e paralelo para uma matriz de ordem 10240 com 5, 10 e 20 *threads*.

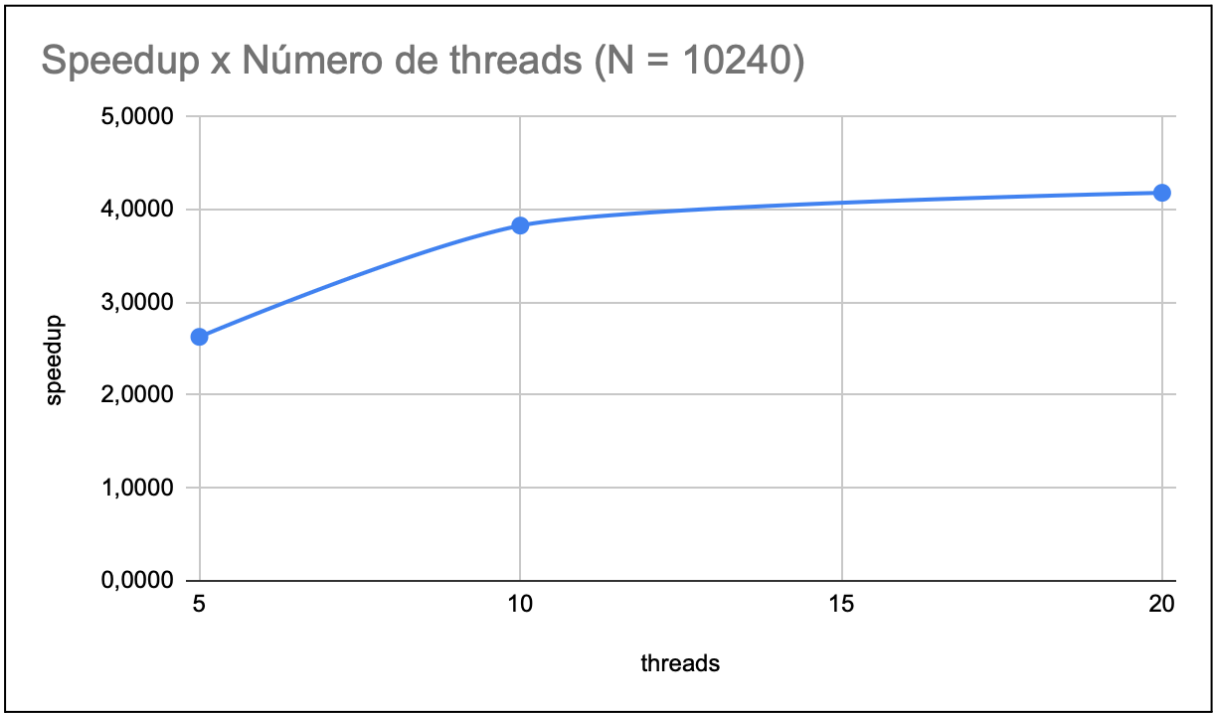

Gráfico 17: *Speedup* para uma matriz de ordem 10240 com 5, 10, e 20 *threads*.

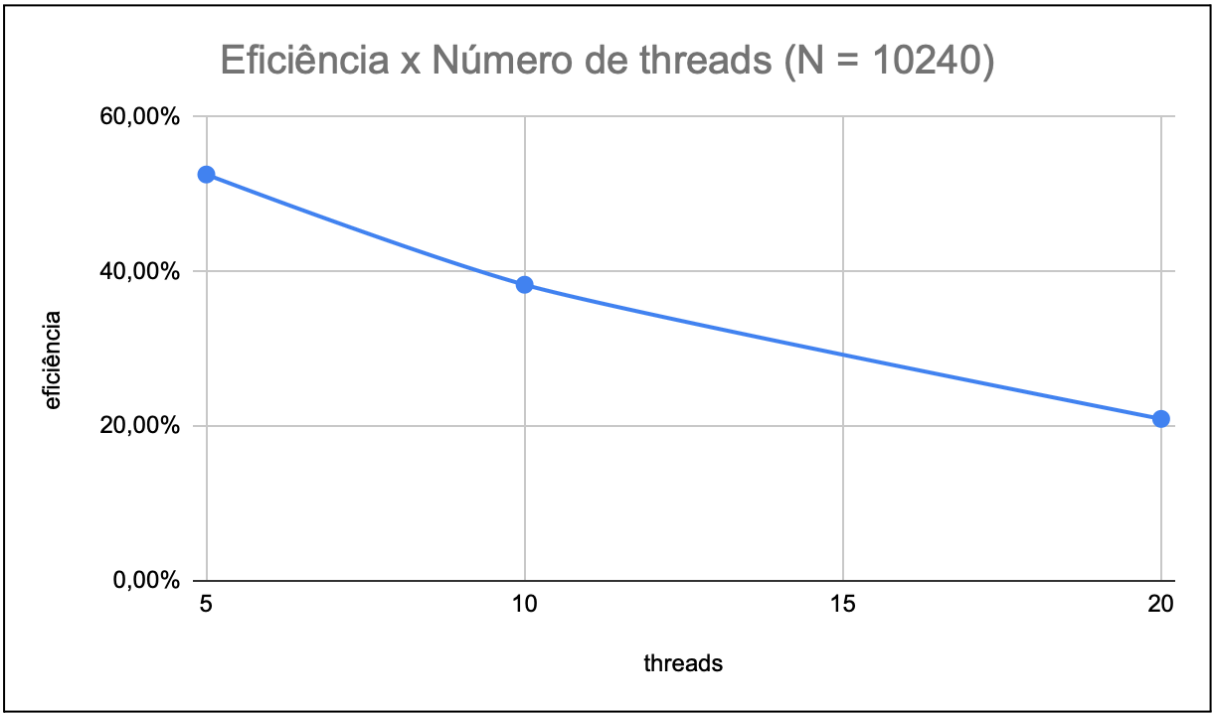

Gráfico 18: Eficiência para uma matriz de ordem 10240 com 5, 10, e 20 *threads*.

Observa-se mais uma vez que a adição de *threads* para executar tarefas paralelas não é acompanhada por um aumento proporcional no desempenho, indicando que o *overhead* associado à criação, escalonamento e gerenciamento dessas *threads* também cresce, reduzindo a eficiência geral da aplicação.

Considerações finais

A realização dos testes retornou uma amostra confiável, evidenciada pelo desvio padrão relativamente baixo em todos os cenários, o que permite realizar algumas análises a respeito do desempenho do código paralelo em relação ao código sequencial. Os gráficos 19 e 20 trazem, respectivamente, um comparativo da evolução do *speedup* e eficiência, condensando os resultados obtidos em cada um dos cenários testados. Nota-se que para a matriz de menor dimensão os custos superam os benefícios da paralelização. Para matrizes de maiores dimensões são obtidos os melhores resultados, sendo a matriz com a maior dimensão aquela que apresentou a maior eficiência. O número de *threads* utilizado também teve influência no cálculo; em todas as configurações, o menor número de *threads* testado foi o que obteve a maior eficiência, podendo-se inferir que o *overhead* de sincronização, causado pelo uso das diretivas de sincronização, foi o fator chave para o desempenho aferido.

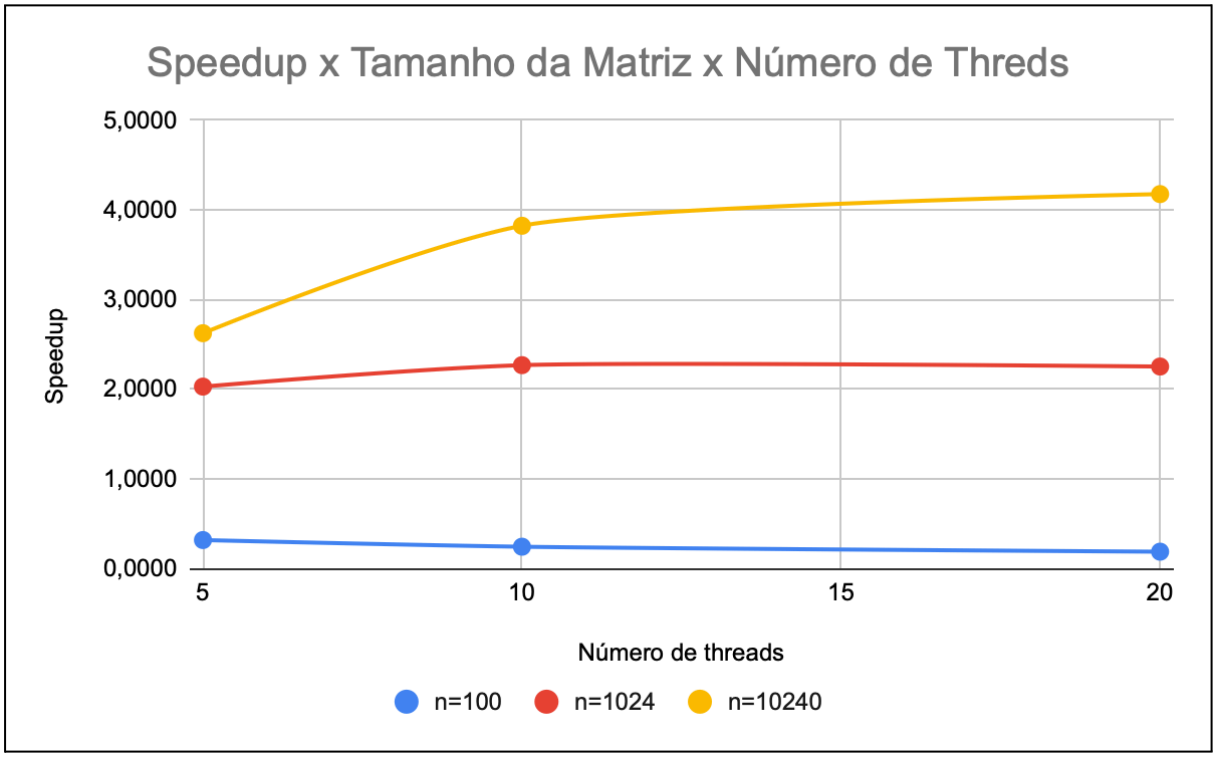

Gráfico 19: Comparativo do *speedup* para cada ordem de matriz testada com 5, 10, e 20 *threads*.

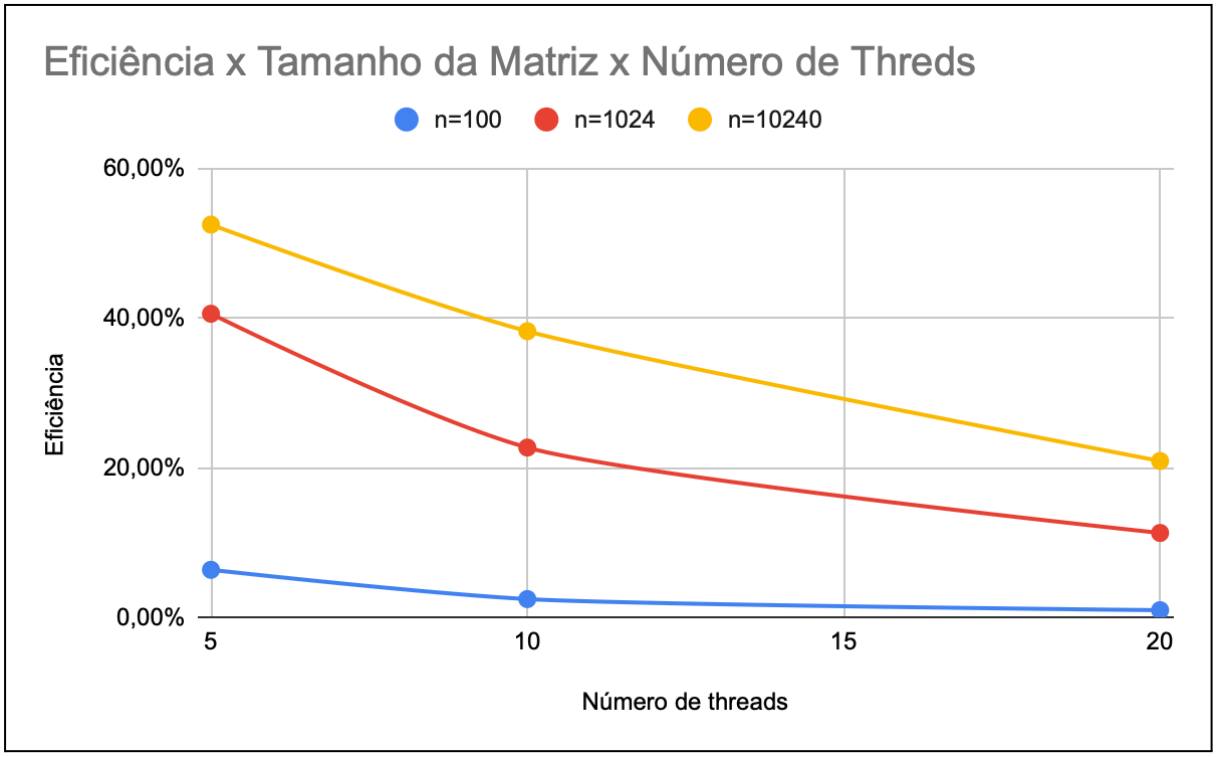

Gráfico 20: Comparativo da eficiência para cada ordem de matriz testada com 5, 10, e 20 *threads*.