Relatório PL3

Gleizielly Alves (1210133) e Márcia Rocha (1210138)

1. Introdução

Este trabalho tem como objetivo descrever a resolução dos exercícios 5 e 6 da PL3 e analisar a performance dos códigos em suas versões sequenciais e paralelas utilizando a biblioteca do OpenMP. Além disso, o trabalho analisa a performance dos códigos utilizando dois compiladores: o GCC e o LLVM.

2. Implementação, resultados e discussões

Para análise de performance foram tomadas dezoito amostras para o exercício 5 e vinte e cinco amostras para o exercício 6, utilizando cada um dos compiladores. Cada amostra contém o tempo de execução e, para o exercício 6, os números de operações de ponto flutuante por segundo (FLOPS). Em seguida, foram feitos cálculos estatísticos de média, desvio padrão, máximo e mínimo para cada implementação.

2.1. Exercício 5

No exercício 5 foram realizadas duas implementações da suavização de Gauss-Seidel. A primeira é uma abordagem sequencial e a segunda uma abordagem de execução paralela utilizando tarefas e dependências. As tabelas 1 e 2 contém informação sobre o tamanho da matriz, tamanho do bloco, número de threads utilizadas e o tempo de execução para cada uma das implementações, além dos cálculos estatísticos para análise de performance.

Tabela 1 - Amostras do compilador GCC

Matrix size	Block size	Num Threads	Sequential time (ms)	Paralallel time (ms)
1024x1024	32x32	2	8,245	4,811
1024x1024	32x32	2	8,344	4,761
1024x1024	32x32	2	8,662	5,019
1024x1024	32x32	4	8,268	2,778
1024x1024	32x32	4	8,458	2,816
1024x1024	32x32	4	8,458	2,947
1024x1024	32x32	6	8,332	2,058
1024x1024	32x32	6	8,475	2,053
1024x1024	32x32	6	8,338	2,023
1024x1024	32x32	8	8,554	2,215
1024x1024	32x32	8	8,356	1,785
1024x1024	32x32	8	8,352	1,668
1024x1024	32x32	10	8,263	1,965

Porto, 18 de novembro de 2022.

1024x1024	32x32	10	8,258	2,337
1024x1024	32x32	10	8,339	1,969
1024x1024	32x32	12	8,271	1,938
1024x1024	32x32	12	8,431	2,311
1024x1024	32x32	12	8,391	2,001
Statistical Analysis				
Average			8,378	2,636
Standard Deviation			0,109	1,053
Maximum			8,662	5,019
Minimum			8,245	1,668

Tabela 2 - Amostras do compilador LLVM

Matrix size	Block size	Num Threads	Sequential time (ms)	Parallel time (ms)
1024x1024	32x32	2	8,252	5,499
1024x1024	32x32	2	8,408	5,703
1024x1024	32x32	2	8,214	5,903
1024x1024	32x32	4	8,569	3,15
1024x1024	32x32	4	8,555	3,662
1024x1024	32x32	4	8,51	3,109
1024x1024	32x32	6	8,544	2,351
1024x1024	32x32	6	8,534	2,759
1024x1024	32x32	6	8,598	2,34
1024x1024	32x32	8	16,223	1,992
1024x1024	32x32	8	12,642	5,696
1024x1024	32x32	8	12,004	5,571
1024x1024	32x32	10	10,747	2,003
1024x1024	32x32	10	11,696	2,15
1024x1024	32x32	10	10,753	2,011
1024x1024	32x32	12	15,634	2,077
1024x1024	32x32	12	13,017	2,076
1024x1024	32x32	12	13,599	2,103
Statistical Analysis				
Average			10,694	3,342
Standard Deviation			2,584	1,516
Maximum			16,223	5,903
Minimum			8,214	1,992

Os gráficos a seguir ilustram o tempo de execução para diferentes números de threads ao utilizar o compilador GCC (Figura 1) e o compilador LLVM (Figura 2), respectivamente:

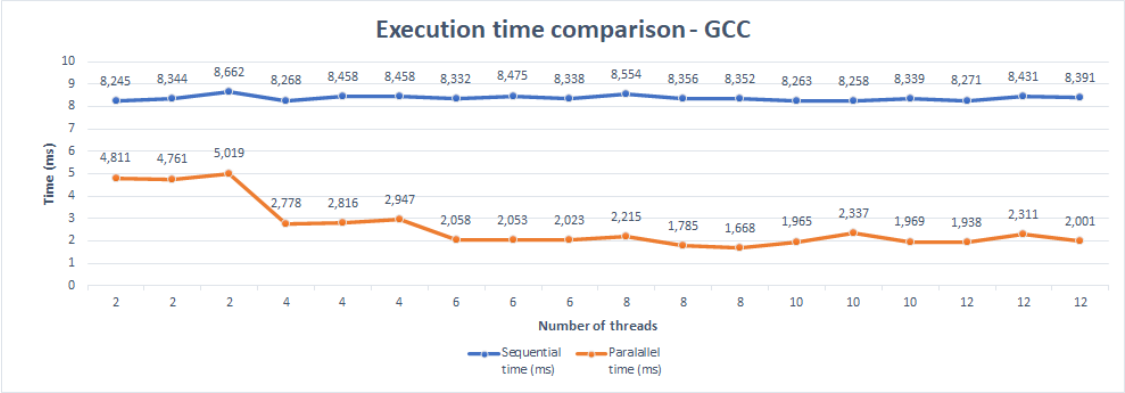

Figura 1 – Comparação dos tempos de execução do compilador GCC

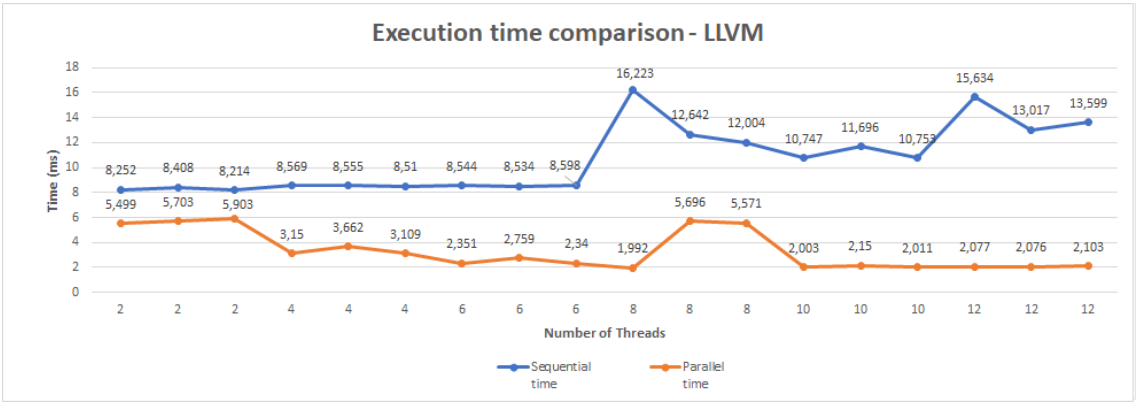

Figura 2 – Comparação dos tempos de execução do compilador LLVM

É possível perceber que o número de threads diminui o tempo de execução até um certo limite para o compilador GCC, devido às limitações do paralelismo e ao peso da criação de novas threads no tempo de execução. Além disso, o compilador GCC apresenta um desvio padrão muito menor que o LLVM, tanto para execução sequencial (Figura 3) quanto paralela (Figura 4), além de ter demonstrado maior performance, ou seja, menor tempo de execução médio.

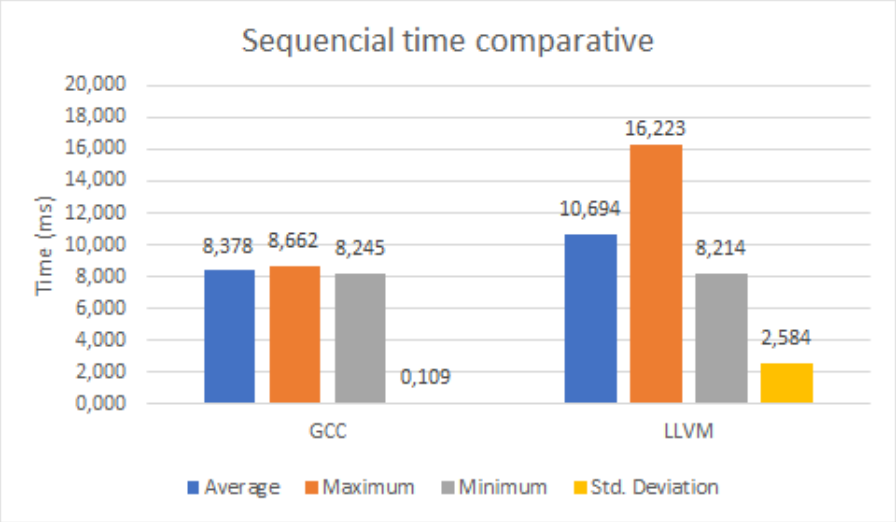

Figura 3 – Comparação do tempo sequencial entre GCC e LLVM

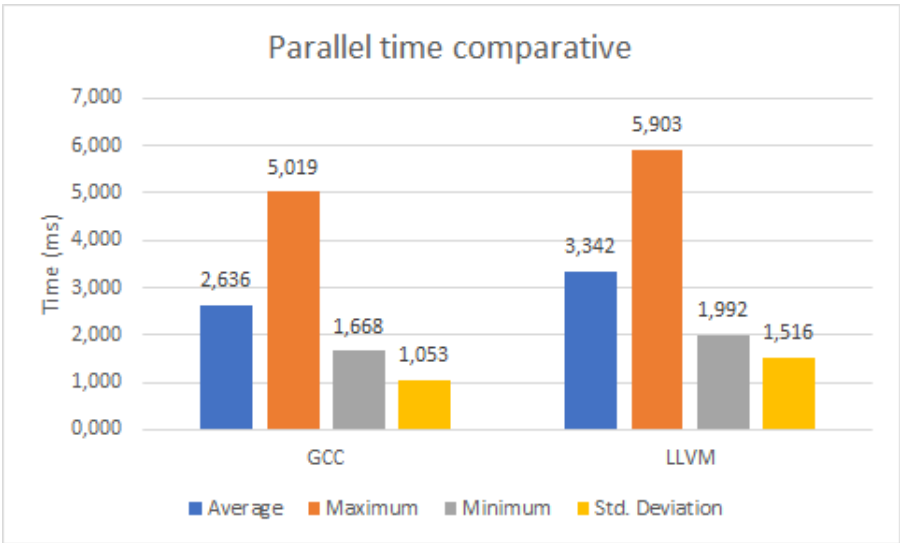

Figura 4 – Comparação do tempo paralelo entre GCC e LLVM

2.2. Exercício 6

No exercício 6, foi utilizado o algoritmo de Cholesky para fatorar uma matriz simétrica. Este algoritmo possui quatro operações *potrf*, *trsm*, *syrk e gemm*. Foram dadas duas versões deste algoritmo: a primeira trata-se de uma execução sequencial e a segunda, uma paralelização utilizando a diretiva do OpenMP *parallel for*.

O exercício 6 também propunha o desenvolvimento de outras duas versões utilizando a diretiva *task*: a primeira em conjunto com a diretiva *taskwait* e a segunda com a cláusula *depends*, na qual é possível definir a dependências de dados entre as tarefas.

As tabelas 3 e 4 mostram os dados obtidos durante a execução do programa bem como o resultado dos cálculos estatísticos. Os dados de tempo foram utilizados para a construção dos gráficos apresentados nas figuras 5 e 6, respectivamente. Os resultados da análise estatística foram utilizados para construir os gráficos da figura 7, os quais comparam as médias do tempo de execução dos compiladores, o desvio padrão, e o valores de máximo e mínimo.

Tabela 3 - Amostras do compilador GCC

Matrix size	Block size	Num threads	Sequential time (ms)	Sequential performance (flops)	Parallel-for time (ms)	Parallel-for performance (flops)	Task-wait time (ms)	Task-wait performance (flops)	Task-dependences time (ms)	Task-dependences performance (flops)
1000	10	2	40,786	8,173	24,991	13,338	42,606	7,824	42,676	7,811
1000	10	2	41,034	8,123	24,242	13,750	42,883	7,773	43,228	7,711
1000	10	2	41,899	7,956	24,510	13,600	44,483	7,493	44,408	7,506
1000	10	2	41,128	8,105	24,316	13,708	43,031	7,746	43,347	7,690
1000	10	2	40,867	8,157	24,604	13,548	43,019	7,749	43,284	7,701
1000	10	4	40,675	8,195	23,083	14,441	44,489	7,492	43,947	7,585
1000	10	4	40,791	8,172	20,539	16,229	44,543	7,483	42,710	7,805
1000	10	4	41,327	8,066	20,528	16,238	43,952	7,584	42,948	7,761
1000	10	4	41,306	8,070	20,947	15,913	44,562	7,480	43,152	7,725
1000	10	4	41,396	8,052	20,934	15,923	43,924	7,589	43,013	7,750
1000	10	6	42,289	7,882	27,546	12,101	46,217	7,212	43,217	7,713
1000	10	6	41,132	8,104	28,720	11,606	46,122	7,227	44,047	7,568
1000	10	6	41,896	7,956	28,697	11,616	46,824	7,119	43,984	7,579
1000	10	6	41,191	8,092	28,407	11,734	46,235	7,210	44,631	7,469
1000	10	6	41,237	8,083	28,244	11,802	46,156	7,222	44,498	7,491
1000	10	8	42,150	7,908	24,589	13,556	47,091	7,078	45,405	7,341
1000	10	8	41,804	7,974	25,186	13,235	46,900	7,107	45,642	7,303
1000	10	8	41,956	7,945	25,367	13,140	47,351	7,040	45,623	7,306
1000	10	8	42,527	7,838	26,305	12,672	47,310	7,046	45,610	7,308
1000	10	8	43,094	7,735	24,559	13,573	47,471	7,022	45,770	7,283
1000	10	10	41,817	7,971	185,923	1,793	45,750	7,286	43,599	7,645
1000	10	10	41,133	8,104	200,648	1,645	45,730	7,289	45,471	7,331
1000	10	10	41,159	8,099	184,382	1,808	46,292	7,201	43,958	7,583
1000	10	10	40,632	8,204	204,633	1,629	46,207	7,214	43,065	7,740
1000	10	10	42,202	7,899	185,147	1,800	44,786	7,443	42,838	7,781
Statistical Analysis										
Average			41,497	8,034	58,282	11,216	45,357	7,357	44,003	7,579
Standard Deviation			0,622	0,119	67,085	4,921	1,499	0,246	1,035	0,176
Maximum			43,094	8,204	204,633	16,238	47,471	7,824	45,770	7,811
Minimum			40,632	7,735	20,528	1,629	42,606	7,022	42,676	7,283

Tabela 4 - Amostras do compilador LLVM

Matrix size	Block size	Num threads	Sequential time (ms)	Sequential performance (flops)	Parallel-for time (ms)	Parallel-for performance (flops)	Task-wait time (ms)	Task-wait performance (flops)	Task-dependences time (ms)	Task-dependences performance (flops)
1000	10	2	34,361	9,701	24,969	13,350	46,619	7,150	47,328	7,043
1000	10	2	34,709	9,604	24,978	13,345	47,112	7,075	47,480	7,020
1000	10	2	34,233	9,737	25,044	13,310	47,122	7,074	47,166	7,067
1000	10	2	35,635	9,354	25,856	12,892	46,885	7,110	47,642	6,997
1000	10	2	34,860	9,562	27,305	12,208	46,437	7,178	47,182	7,065
1000	10	4	35,874	9,292	23,023	14,478	53,399	6,242	55,125	6,047
1000	10	4	34,484	9,666	23,257	14,333	52,790	6,314	54,794	6,083
1000	10	4	35,568	9,372	23,547	14,156	53,427	6,239	54,597	6,105
1000	10	4	35,577	9,369	22,795	14,623	53,202	6,265	54,412	6,126
1000	10	4	34,531	9,653	23,781	14,017	53,924	6,182	54,507	6,115
1000	10	6	35,954	9,271	34,373	9,698	76,841	4,338	79,543	4,191
1000	10	6	35,333	9,434	31,659	10,529	54,637	6,101	54,512	6,115
1000	10	6	35,388	9,419	34,442	9,678	77,460	4,303	80,720	4,130
1000	10	6	35,424	9,410	31,714	10,511	54,321	6,136	54,790	6,084
1000	10	6	34,695	9,608	34,189	9,750	79,003	4,219	81,654	4,082
1000	10	8	35,615	9,359	33,142	10,058	76,934	4,333	79,229	4,207
1000	10	8	36,436	9,148	31,752	10,498	78,188	4,263	82,586	4,036
1000	10	8	36,055	9,245	32,720	10,187	78,204	4,262	80,249	4,154
1000	10	8	34,640	9,623	31,514	10,577	78,582	4,242	79,226	4,207
1000	10	8	36,408	9,156	30,654	10,874	77,664	4,292	80,114	4,161
1000	10	10	35,825	9,304	72,290	4,611	103,998	3,205	97,478	3,420
1000	10	10	34,764	9,588	62,164	5,362	107,599	3,098	96,977	3,437
1000	10	10	36,033	9,251	71,099	4,688	107,967	3,087	94,525	3,526
1000	10	10	34,787	9,582	57,632	5,784	104,895	3,178	94,947	3,511
1000	10	10	36,659	9,093	73,897	4,511	104,218	3,198	97,713	3,411
Statistical Analysis										
Average			35,354	9,432	36,312	10,561	70,457	5,163	69,780	5,134
Standard Deviation			0,693	0,184	16,270	3,232	21,396	1,460	18,502	1,363
Maximum			36,659	9,737	73,897	14,623	107,967	7,178	97,713	7,067
Minimum			34,233	9,093	22,795	4,511	46,437	3,087	47,166	3,411

Analisando o gráfico da figura 5 é possível perceber que não houve variação relevante no tempo de execução em relação ao número de threads, exceto para o *parallel for*, que teve uma grande variação no tempo de execução utilizando dez threads.

O desempenho das implementações utilizando o compilador LLVM foi diferente. A partir do gráfico da figura 6, pode-se notar que o tempo de execução das versões em paralelo e com *tasks* aumentam conforme o incremento do número de threads. Enquanto, na implementação sequencial, os valores de tempo se mantiveram constantes, conforme o esperado.

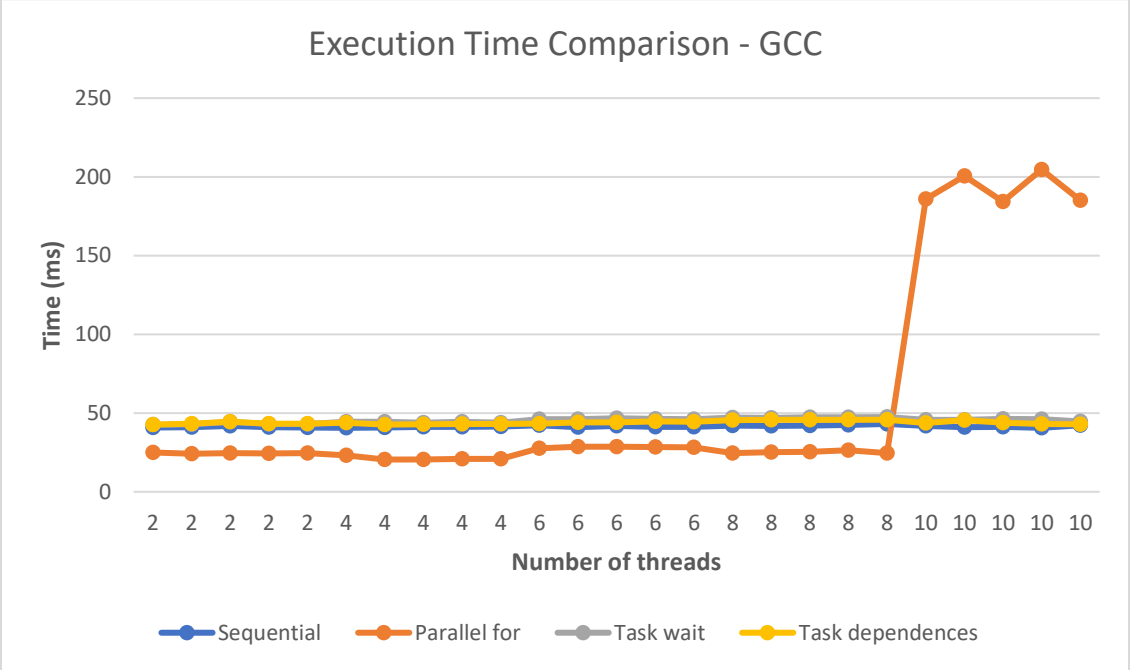

Figura 5 - Comparação do tempo de execução para o compilador GCC

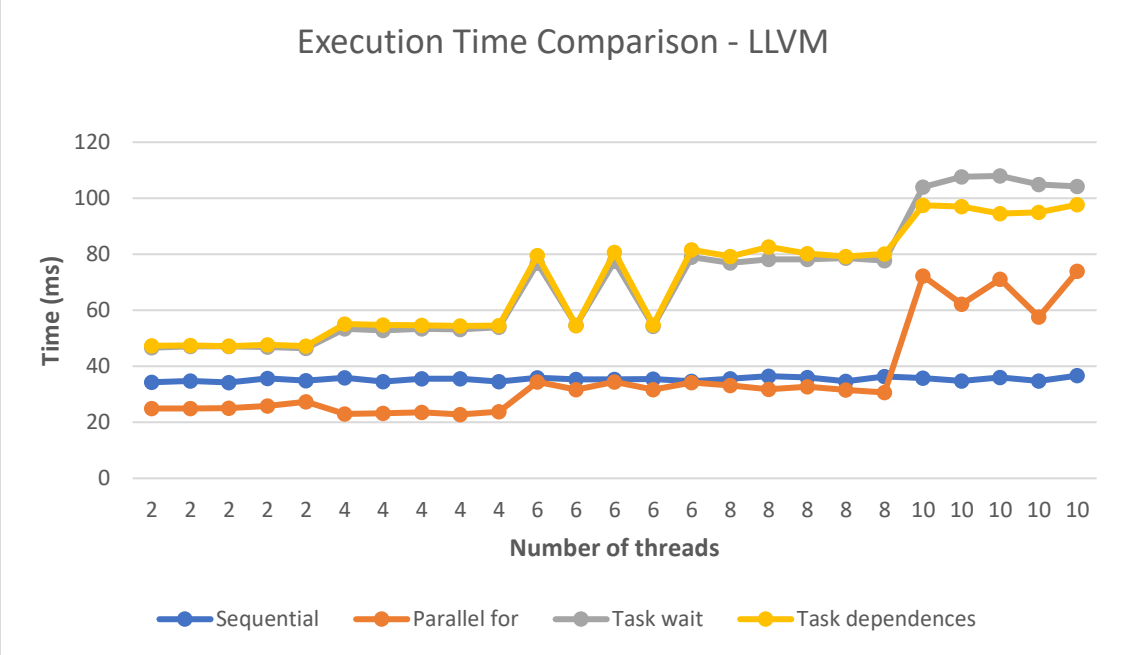

Figura 6 - Comparação do tempo de execução para o compilador LLVM

A discrepância das amostras de uma execução utilizando a mesma quantidade de threads, podem ser explicadas pelo fato de o programa não ter sido executado em ambiente isolado e nem de ter sido atribuído prioridade às tarefas do programa, podendo assim, ter

sofrido interrupção de outros processos devido ao escalonamento feito pelo sistema operacional.

A partir da análise dos gráficos da figura 7, é possível concluir que o compilador LLVM teve melhor performance apenas durante a execução das implementações sequencial e paralela sem a utilização de *tasks*.

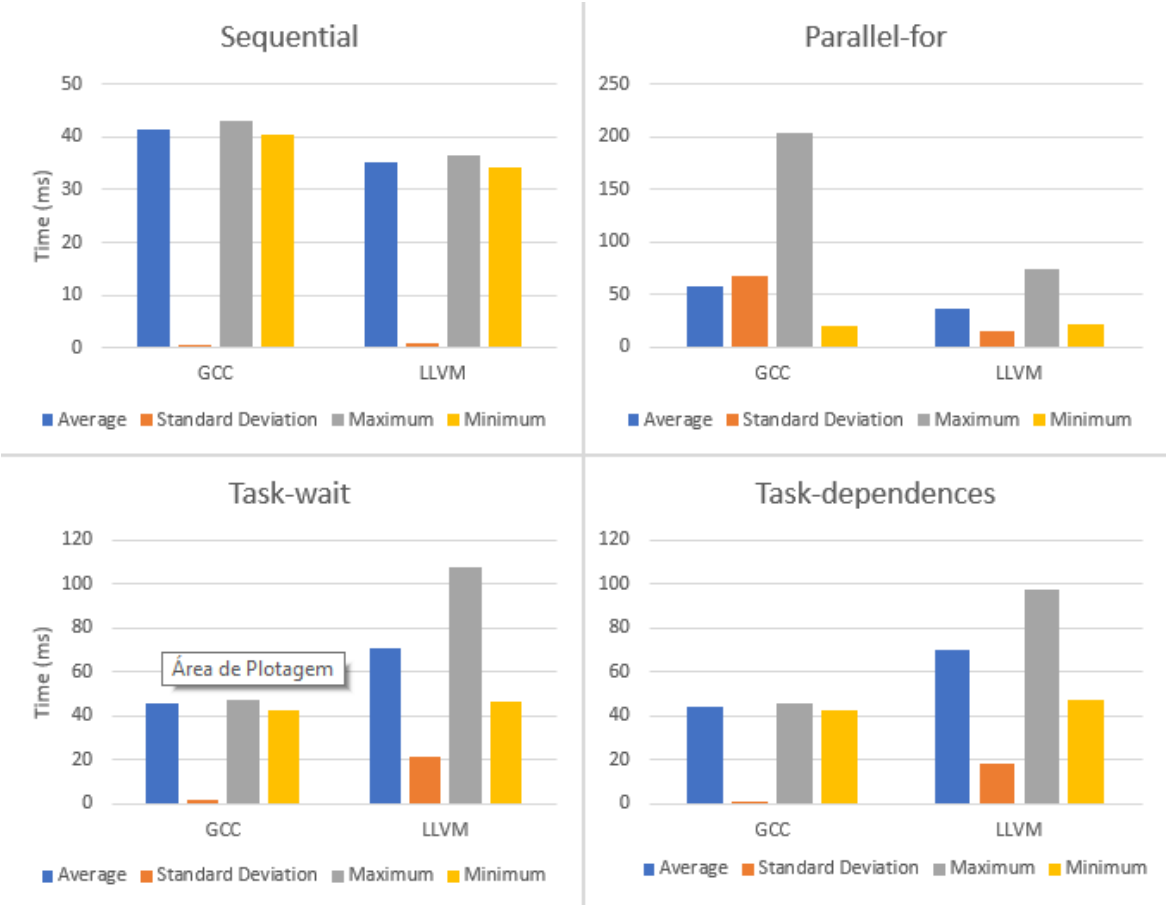

Figura 7 - Comparação do tempo execução entre os compiladores GCC e LLVM

3. Conclusões

Ao compilar ambos os exercícios 5 e 6 com os compiladores GCC e LLVM, foi possível analisar as performances em relação a variação do número de threads para cada caso. Entretanto, apesar de aplicações diferentes em cada exercício, os resultados foram os mesmos.

Ao observar os resultados das análises estatísticas das tabelas apresentadas, conclui-se que o compilador GCC apresenta melhor média de tempo de execução, além de melhores valores máximos e mínimos. Ele também apresentou um menor desvio padrão, sendo a diferença entre os compiladores ainda mais visível para a execução sequencial.

Por outro lado, o compilador LLVM apresentou uma maior variação de resultados conforme o número de threads era incrementado e menor performance. Porém, é possível observar que quanto maior o número de threads, menor a diferença entre o LLVM e o GCC, ambos os compiladores apresentaram tempos de execução paralela semelhantes.

Concluímos então, que o compilador GCC possui mais vantagens para otimização de performance, demonstrando melhores resultados, porém, o compilador LLVM apresentou excelente escalabilidade quanto ao aumento do número de threads, podendo ser mais vantajoso de ser utilizado nas construções de projetos de maiores dimensões.

4. Referências

[1] Jonatha P. Silveira1, Arthur C. Silveira1, Arthur F. Lorenzon. Comparação do Desempenho de Diferentes Compiladores em Ambientes Virtualizados através de Aplicações Paralelas. 2018.

[2] GCC vs. Clang/LLVM: An In-Depth Comparison of C/C++ Compilers. Disponível em: <https://alibabatech.medium.com/gcc-vs-clang-llvm-an-in-depth-comparison-of-c-c-compilers-899ede2be378>. Acesso em: 20 de novembro de 2022.