UNIVERSIDADE FEDERAL DE PELOTAS
Curso de Ciência da computação

Análise de desempenho parte 2: OpenMP + MPI

Giorgio Rossa e Felipe Lopes

Pelotas, 11 de Dezembro de 2020

Em nossa análise de desempenho, utilizamos dois ambientes diferentes. Em cada ambiente foram feitos os mesmos testes, todos utilizando um número de processos igual a 4, variando o número de threads e tamanho da entrada. Cada teste específico foi executado trinta vezes com cem iterações cada, e o valor de tempo foi a média das trinta execuções. Os parâmetros de compilação utilizados em cada ambiente foram -fopenmp -lgomp -03 -std=c++11 -lpthread -fpermissive para a opção paralela e -03 -std=c++11 -fpermissive para a versão “sequencial”. A chamada versão sequencial é uma versão do programa nbody sem threads OpenMP, porém utilizando MPI, que foi implementado da mesma forma que no programa paralelo.

O ambiente de avaliação 1 é composto por um processador Fx 6300, com clock de 3,5 GHz e 6 cores físicos. Este ambiente utiliza o sistema operacional Ubuntu, versão 20.04. A versão do GCC utilizada foi a 9.1 e a do Open MPI foi a 4.0.3.

Os gráficos boxplot construídos a partir dos testes com o tamanho de entrada pequena (16384) no ambiente 1 são apresentados abaixo. Foi gerado um gráfico unindo todos os testes, e depois outros dois que detalham os testes que possuem tempos parecidos, assim como um mostrando os melhores resultados.

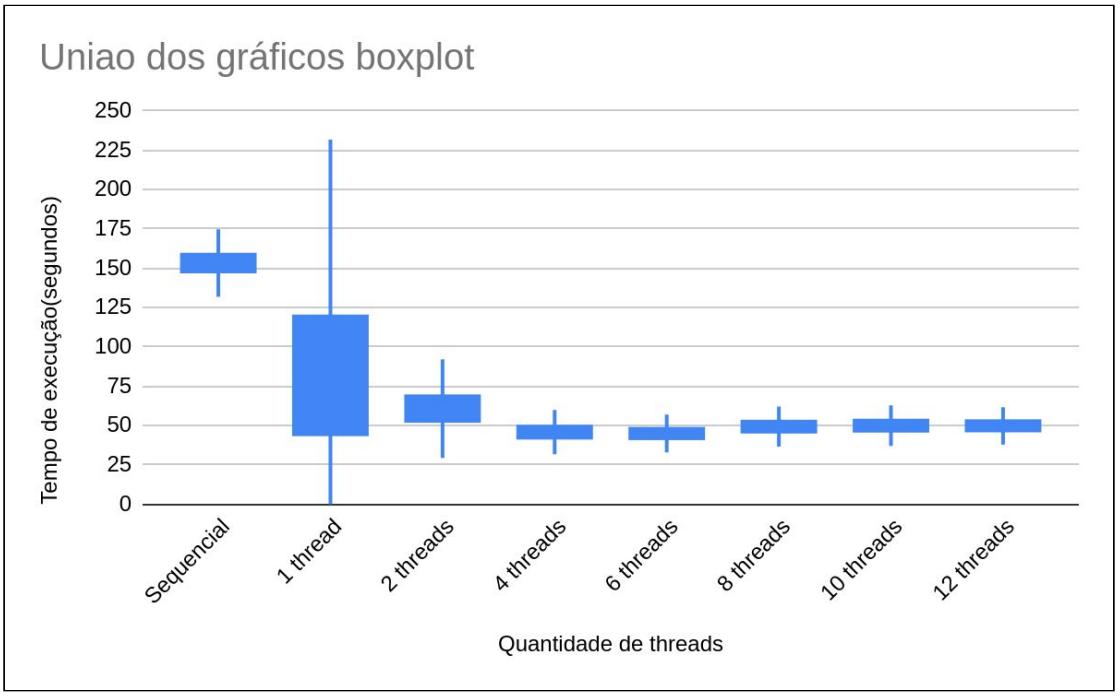


Gráfico 1. União de todos os gráficos boxplot dos testes.

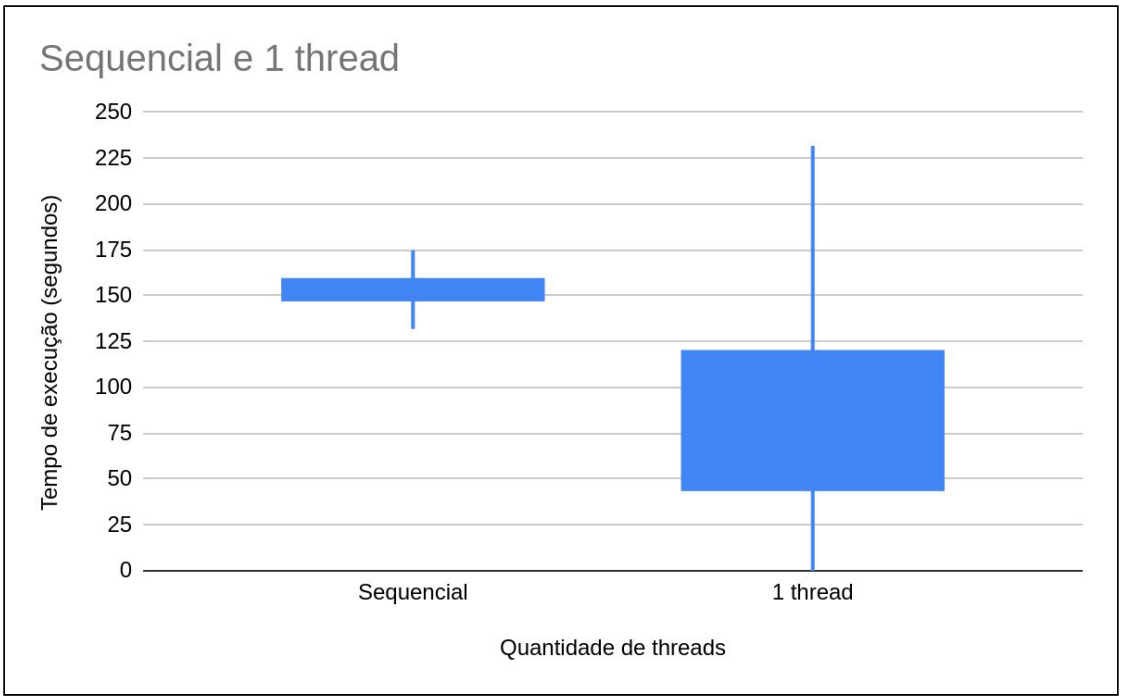


Gráfico 2. Gráficos boxplot para os testes com as implementações sequencial e com 1 thread.

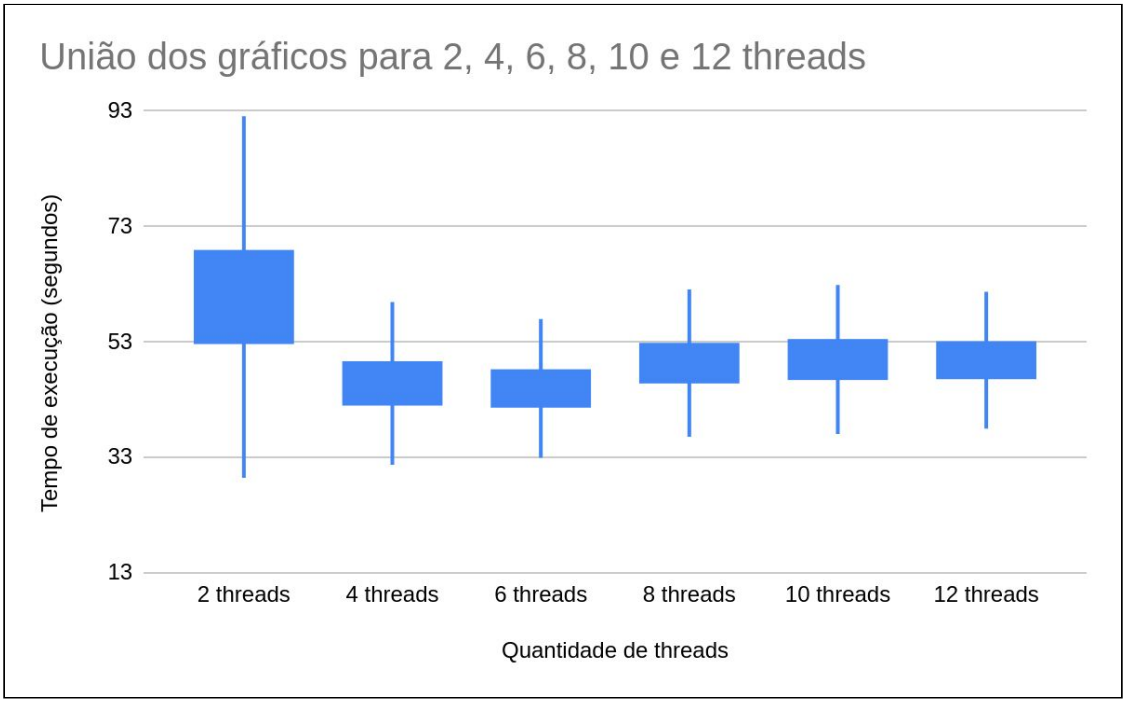


Gráfico 3. Gráficos boxplot para os testes com 2, 4, 6, 8, 10 e 12 threads.

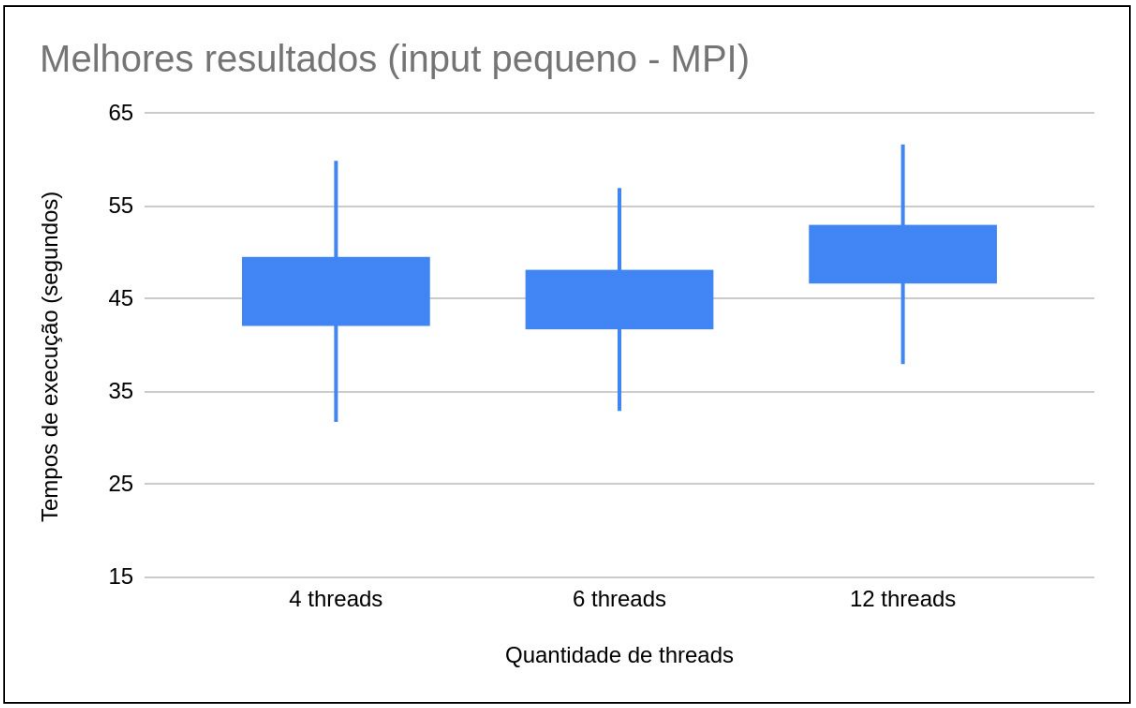


Gráfico 4. Gráficos boxplot para os testes com melhores resultados.

Os gráficos gerados para cada quantidade de threads com o tamanho de entrada grande (32768) no ambiente 1 são apresentados a seguir.

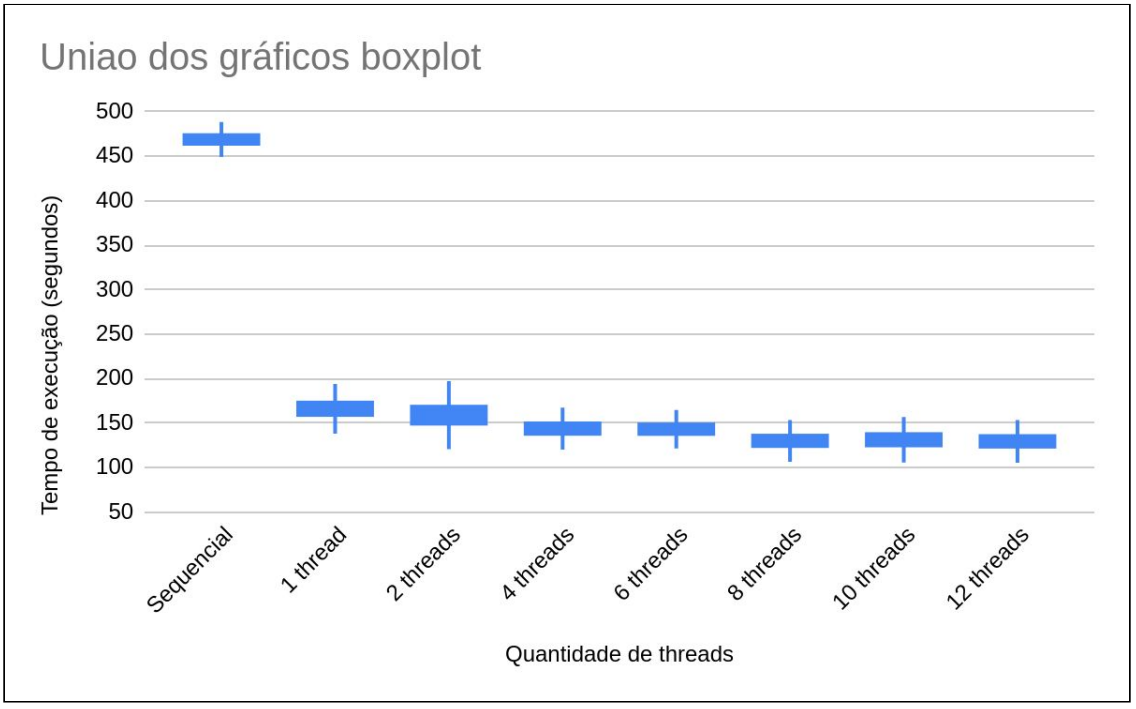


Gráfico 5. União de todos os gráficos boxplot dos testes.

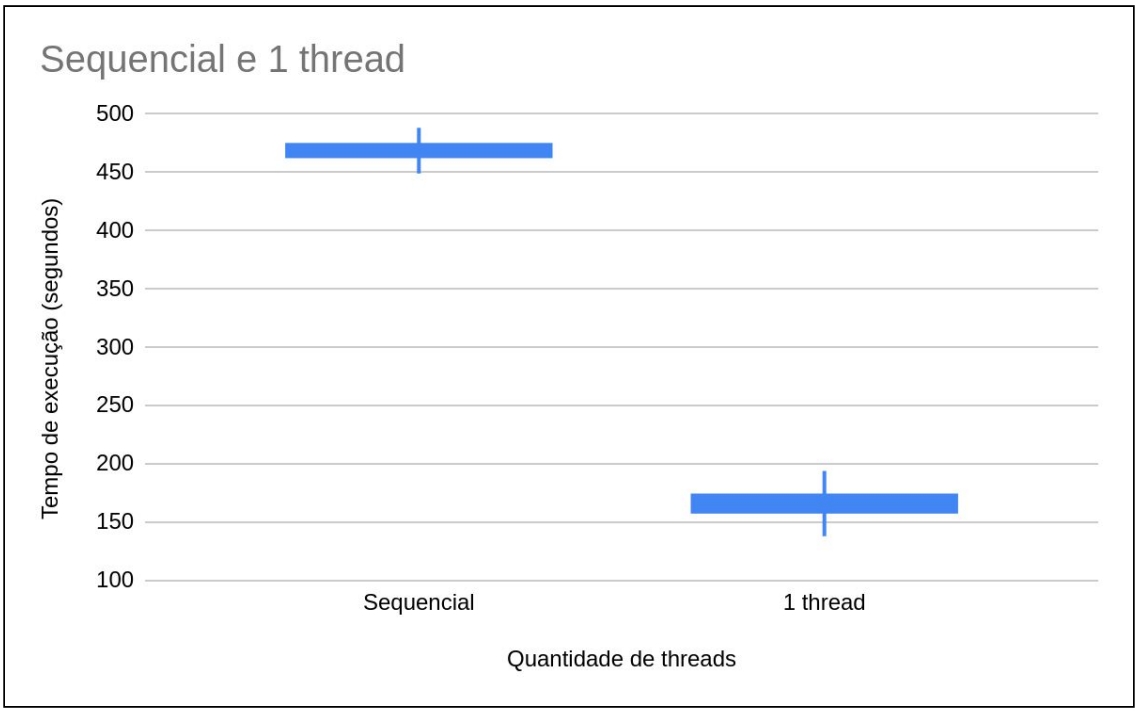


Gráfico 6. Gráficos boxplot para os testes com as implementações sequencial e com 1 thread.

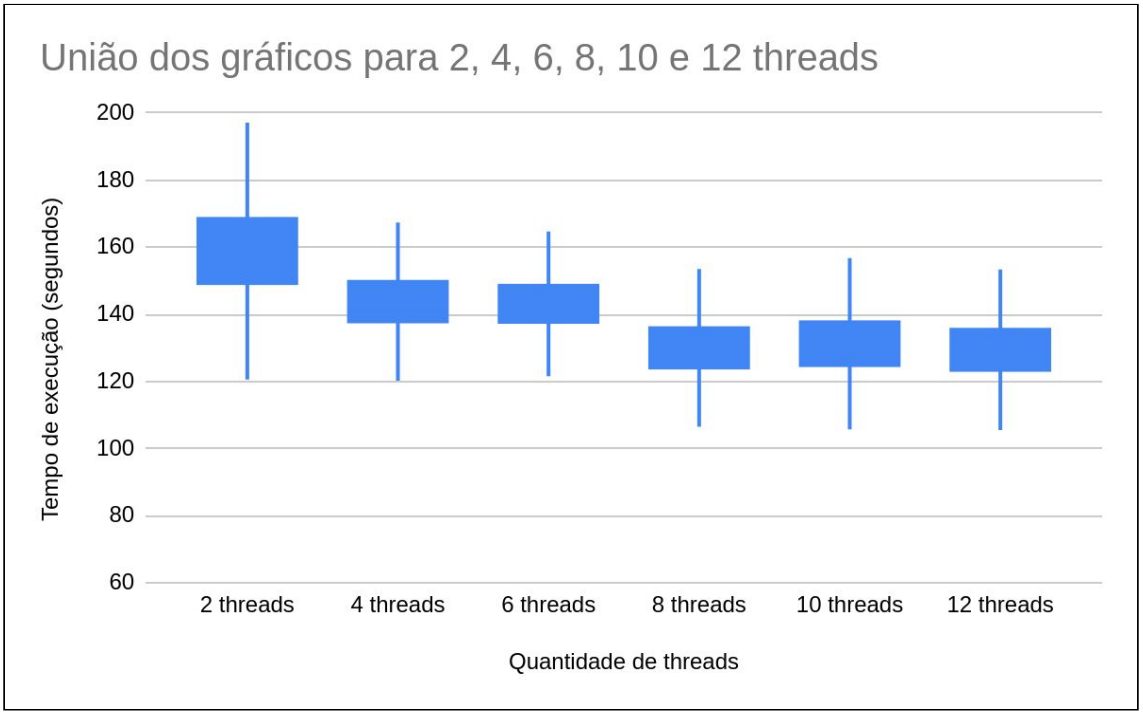


Gráfico 7. Gráficos boxplot para os testes com 2, 4, 6, 8, 10 e 12 threads.

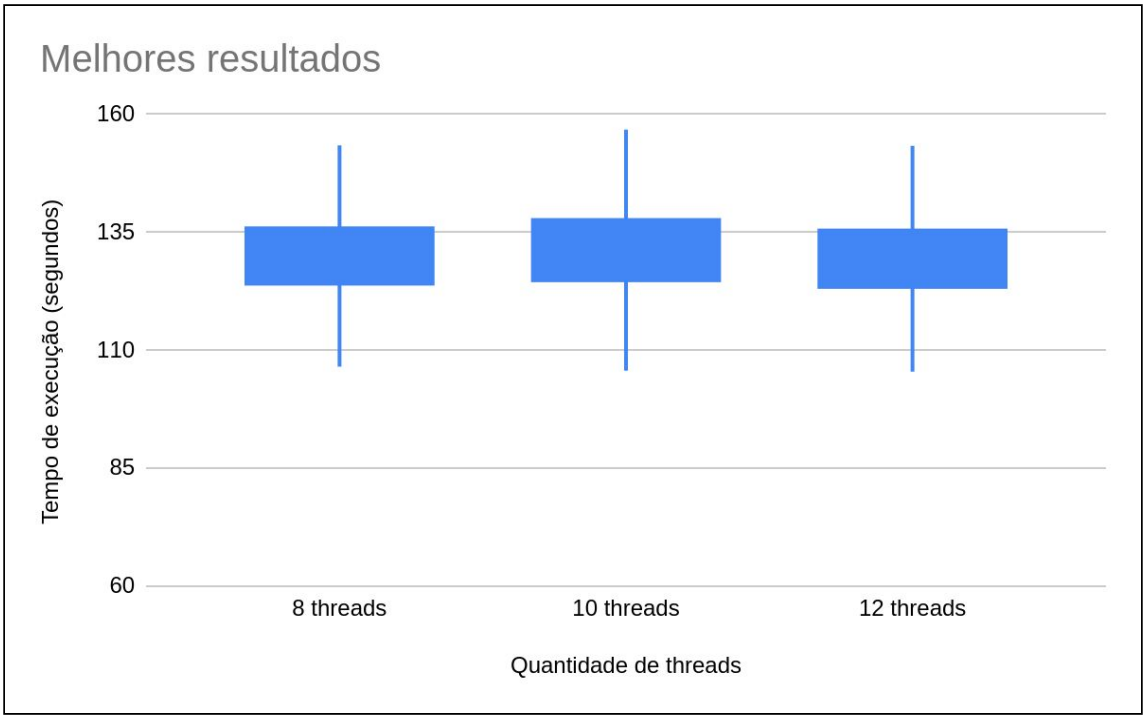


Gráfico 8. Gráficos boxplot para os testes com melhores resultados.

O ambiente de avaliação 2 é composto por um processador Intel Core, modelo i5-8265U, com clock de 1,60 GHz, 4 cores físicos e 8 lógicos. O ambiente de avaliação 2 utiliza o sistema operacional Linux Mint, versão 19. A versão do GCC utilizada foi a 7.5, e a do open MPI foi a 2.1.1.

Os gráficos boxplot construídos para os testes com o tamanho de entrada pequeno (16384) no ambiente 2 são apresentados abaixo.

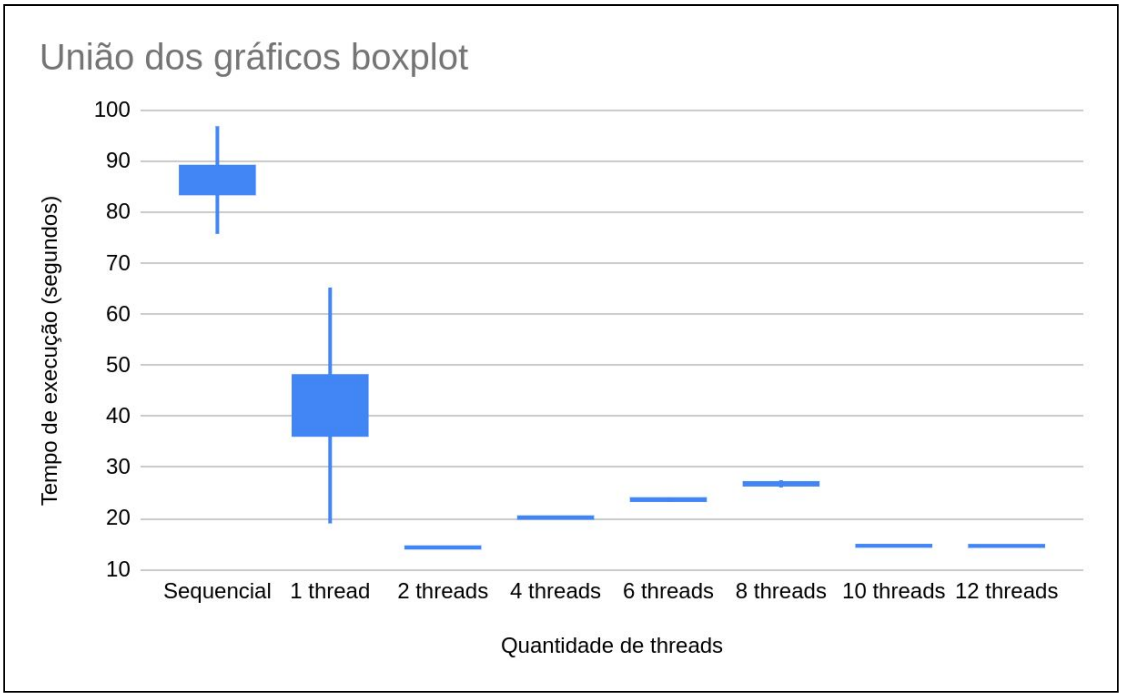


Gráfico 9. União de todos os gráficos boxplot dos testes.

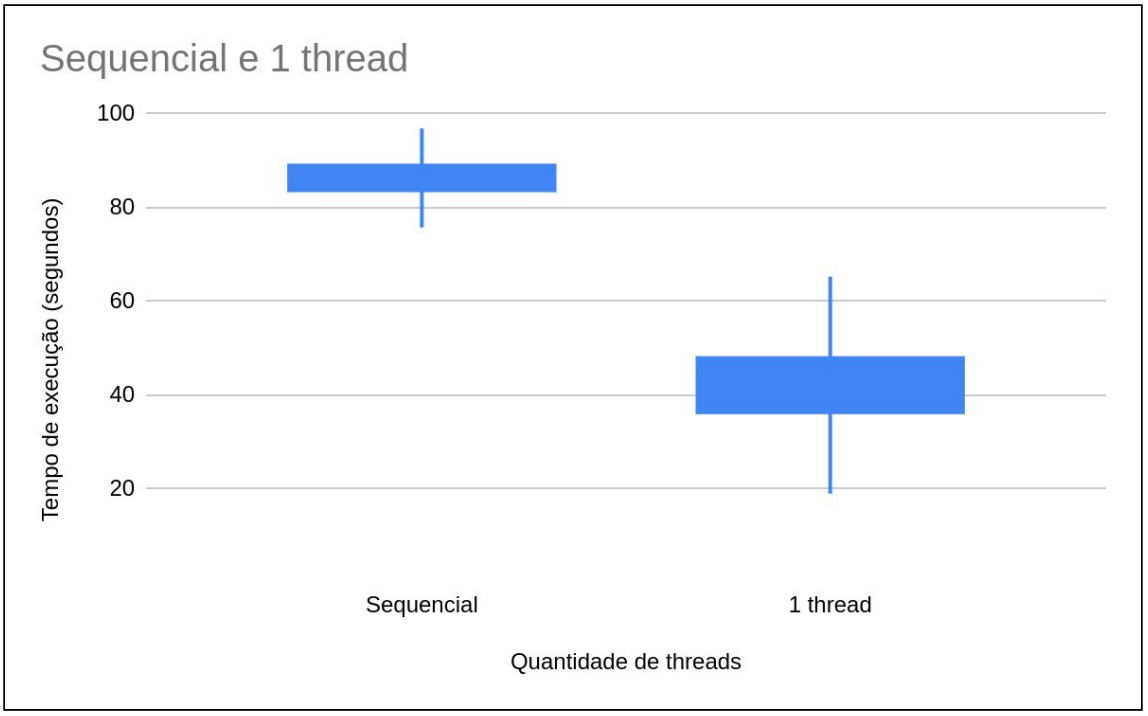


Gráfico 10. Gráficos boxplot para os testes com as implementações sequencial e com 1 thread.

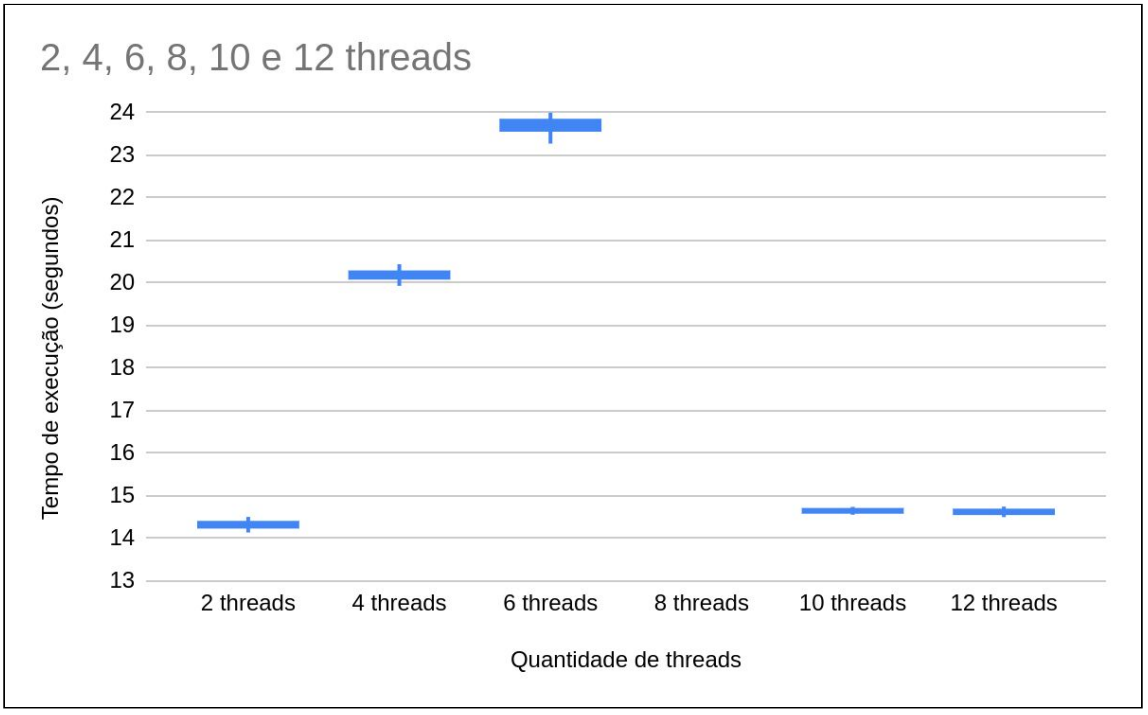


Gráfico 11. Gráficos boxplot para os testes com 2, 4, 6, 8, 10 e 12 threads.

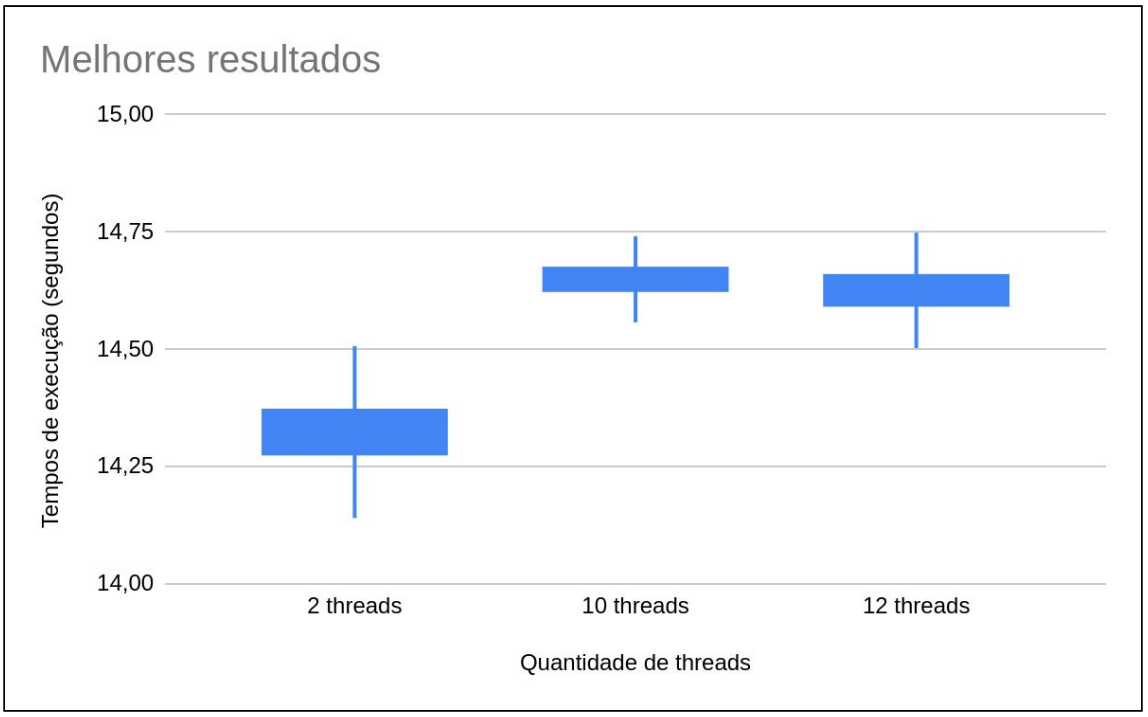


Gráfico 12. Gráficos boxplot para os testes com melhores resultados.

Os gráficos boxplot construídos para os testes com o tamanho de entrada grande (32768) no ambiente 2 são apresentados abaixo.

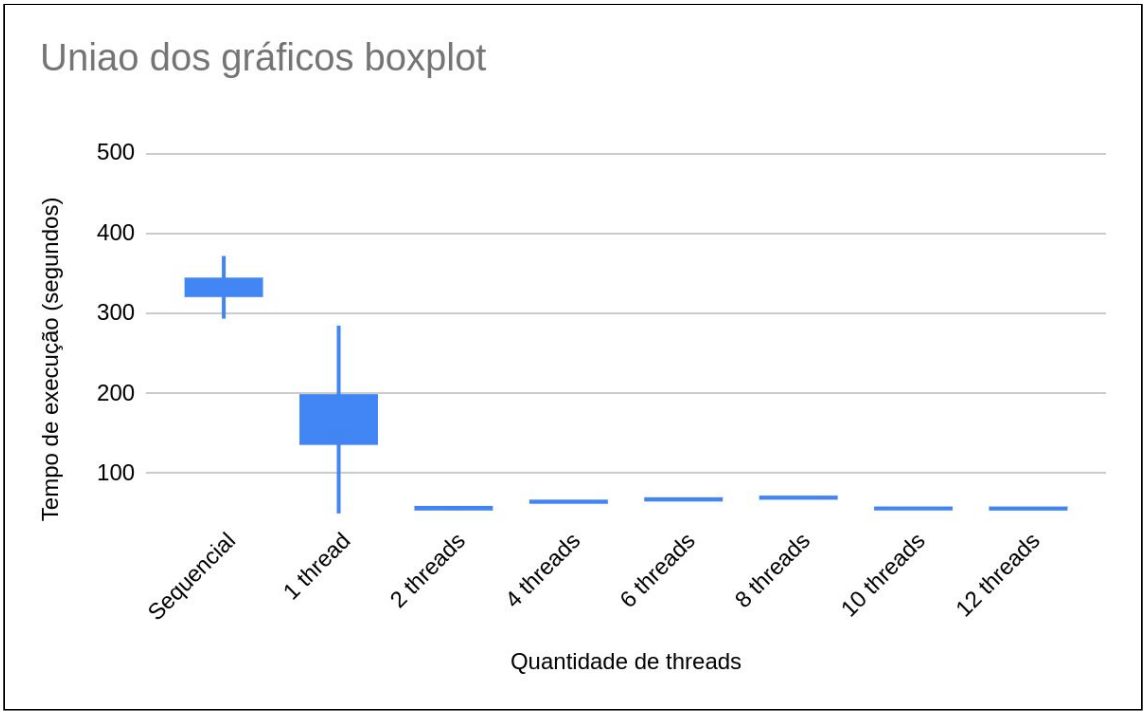


Gráfico 13. União de todos os gráficos boxplot dos testes.

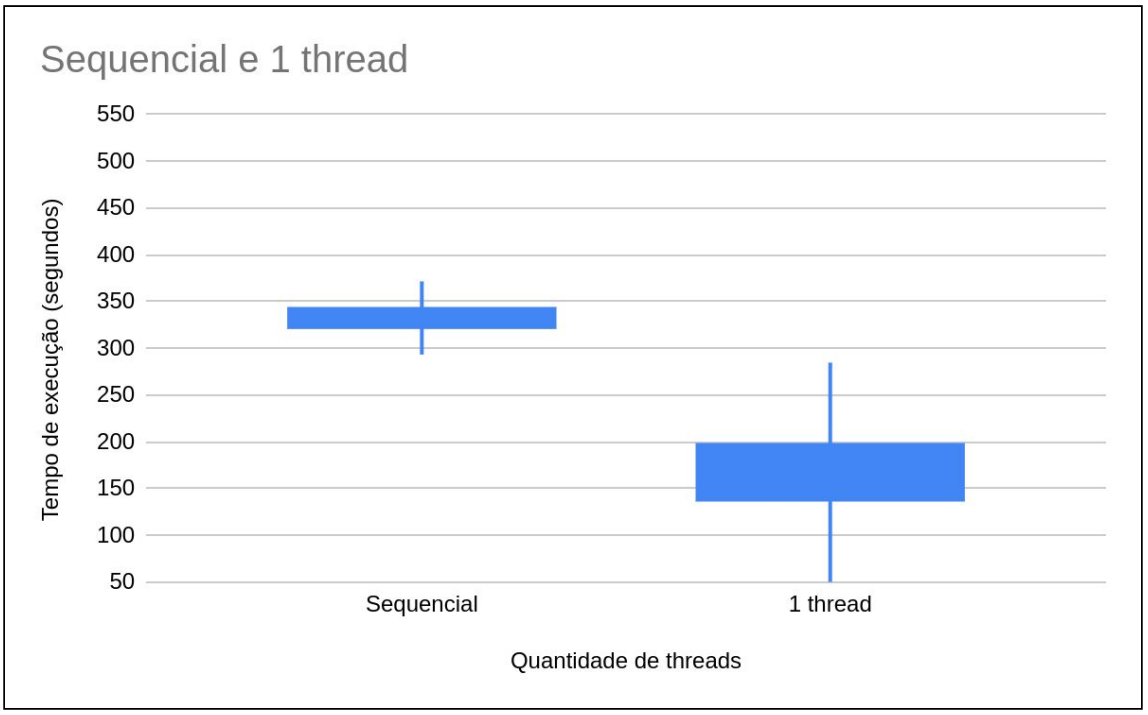


Gráfico 14. Gráficos boxplot para os testes com as implementações sequencial e com 1 thread.

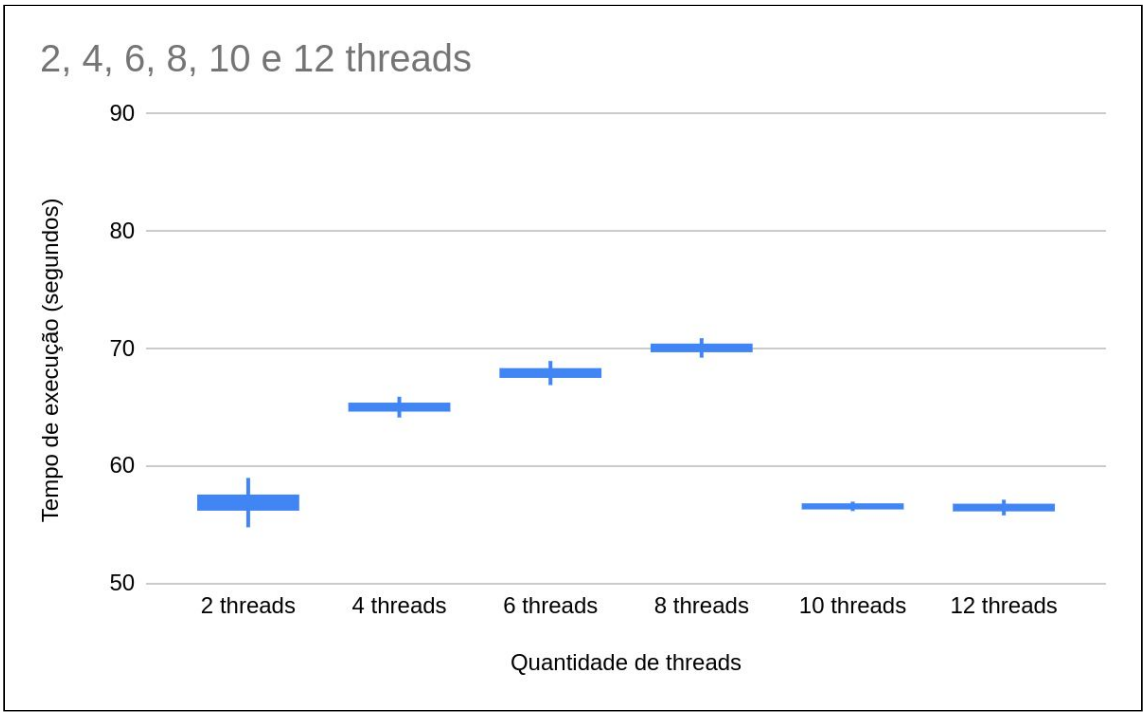


Gráfico 15. Gráficos boxplot para os testes com 2, 4, 6, 8, 10 e 12 threads.

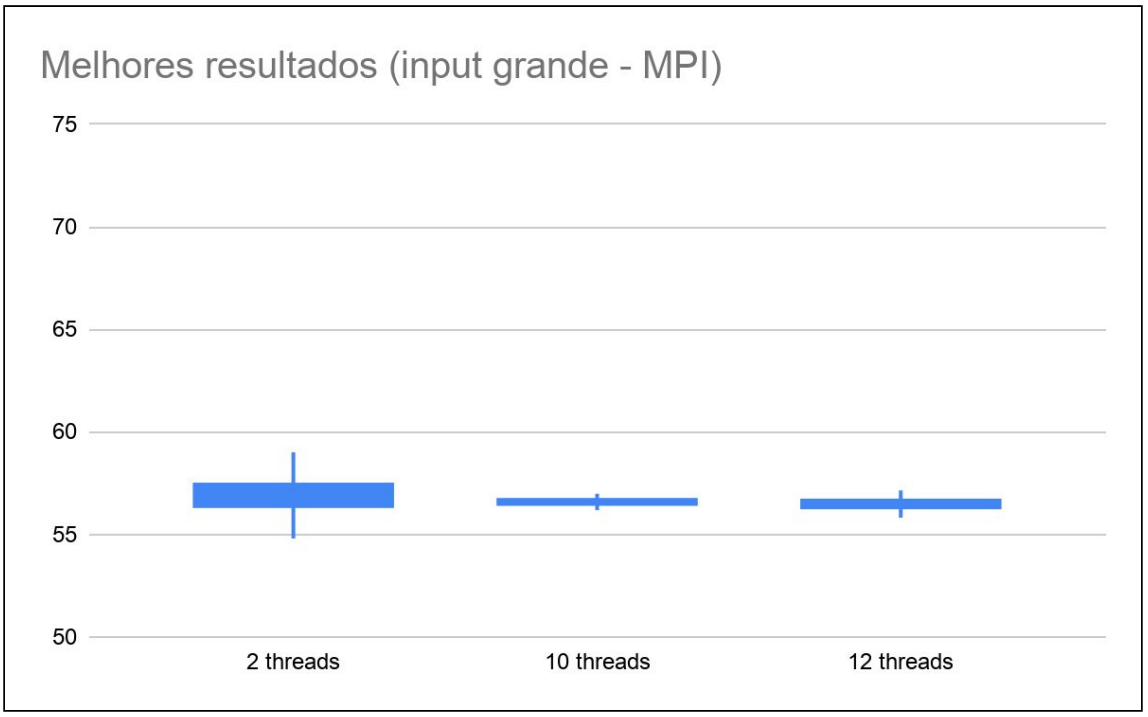


Gráfico 16. Gráficos boxplot para os testes com melhores resultados.

Tempo médios

Os tempos médios estão dispostos em duas tabelas e em dois gráficos, onde é possível visualizar as médias obtidas pelos dois ambientes utilizados durante os testes. Nos testes feitos utilizando o MPI nenhum conjunto de testes foi desconsiderado, pois todos os resultados foram considerados representativos por aderirem à uma curva normal.

Média dos tempos de execução para quantidade de threads								
Ambiente	**Sequencial**	**1 thread**	**2 threads**	**4 threads**	**6 threads**	**8 threads**	**10 threads**	**12 threads**
1	152,786	86,067	60,698	44,257	43,606	48,924	48,301	48,261
2	86,050	43,418	14,319	20,183	23,685	26,806	14,648	14,631

Tabela 1. Média em segundos obtida nos dois ambientes para entrada pequena.

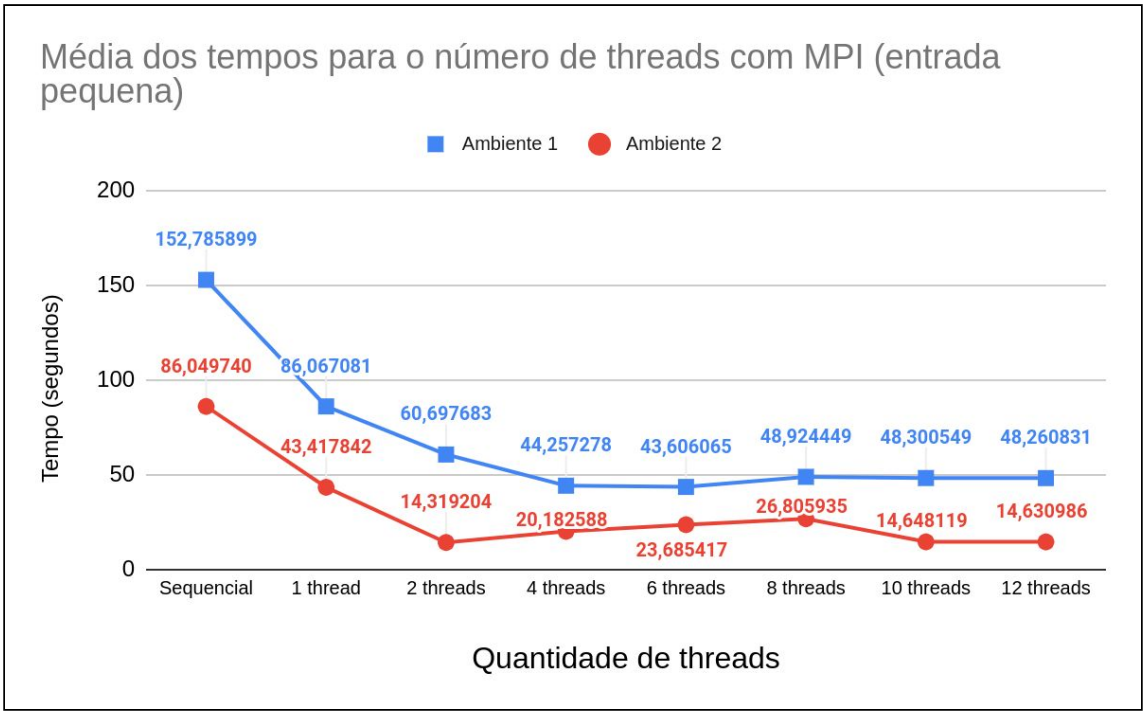


Gráfico 17. Comparativo entre as médias nos ambientes de teste.

Média dos tempos de execução para quantidade de threads								
Ambiente	**Sequencial**	**1 thread**	**2 threads**	**4 threads**	**6 threads**	**8 threads**	**10 threads**	**12 threads**
1	468,361	165,845	154,578	141,317	142,295	129,489	128,992	128,097
2	331,995	162,095	56,993	65,069	68,032	70,049	56,573	56,462

Tabela 2. Média em segundos obtida nos dois ambientes para entrada grande.

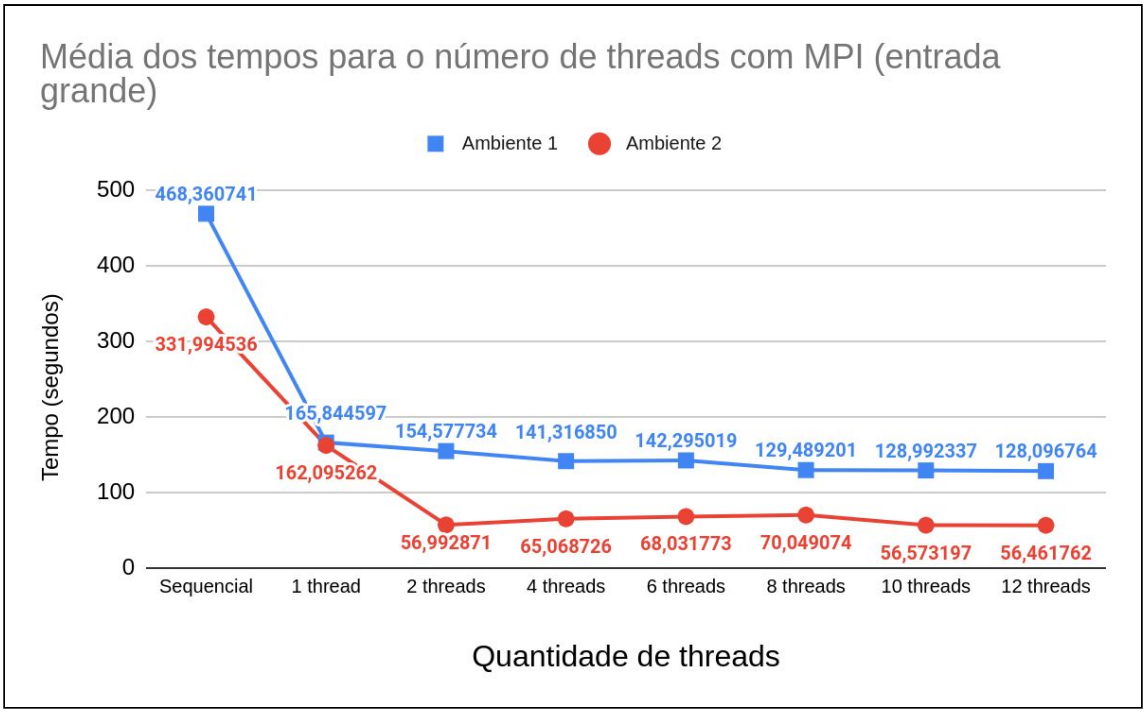
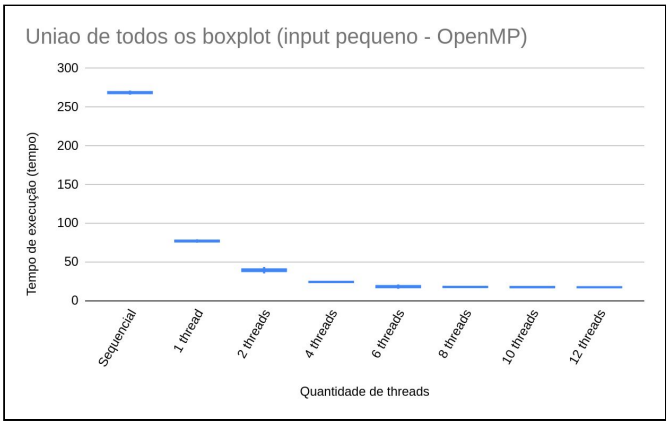


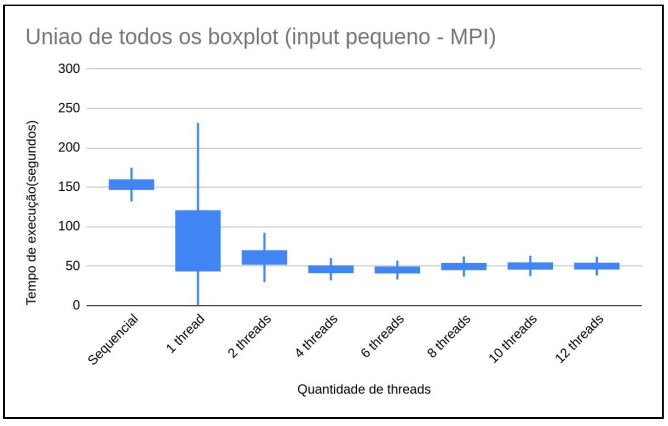
Gráfico 18. Comparativo entre as médias nos ambientes de testes.

Comparação OpenMP e OpenMP + MPI

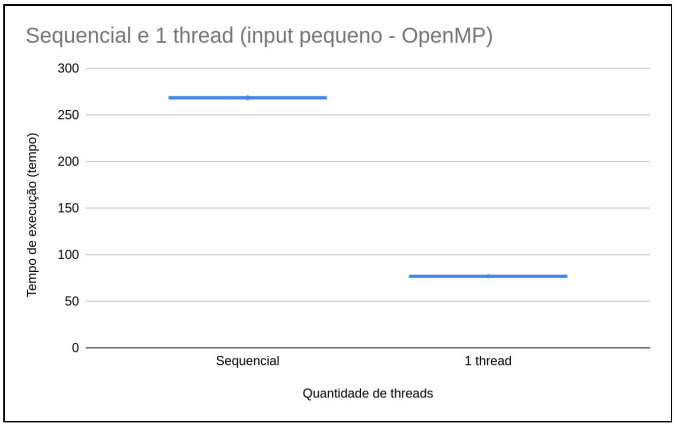
Para comparar as duas implementações utilizamos fortemente os gráficos desenvolvidos durante o primeiro trabalho e o atual. Todos os gráficos foram colocados com a mesma escala no eixo y, eixo do tempo de execução, que é o eixo que muda entre eles.

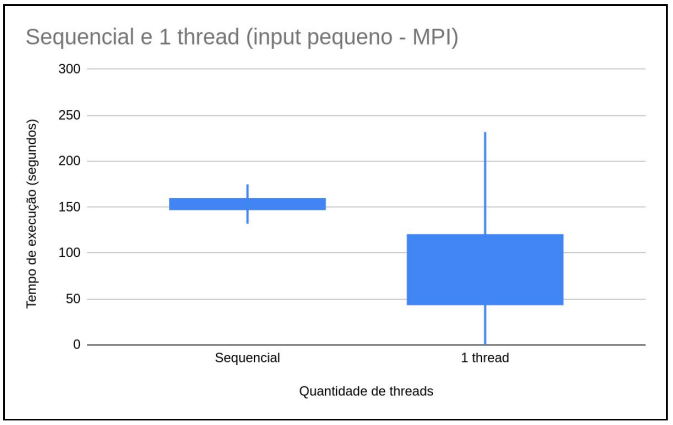
Abaixo temos os gráficos apresentados anteriormente neste trabalho e os gráficos gerados no primeiro trabalho para o ambiente 1. Primeiramente são apresentados os gráficos para entrada pequena (16384) e após os gráficos para entrada grande (32768).



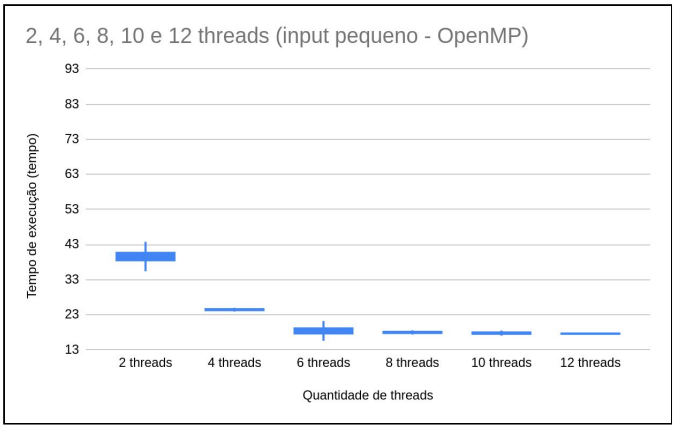


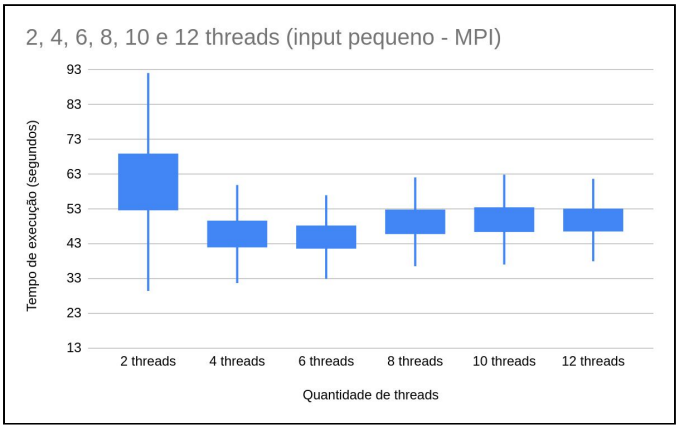
Gráficos 19 e 20. Todos os gráficos boxplot das duas implementações.



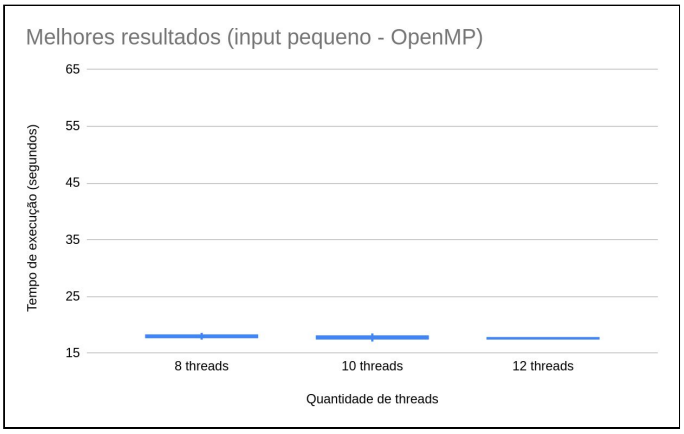


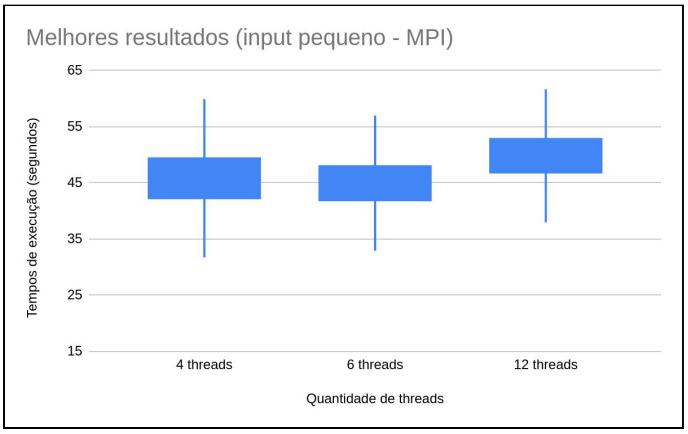
Gráficos 21 e 22. Resultados dos testes sequenciais e com 1 thread, para as 2 implementações.





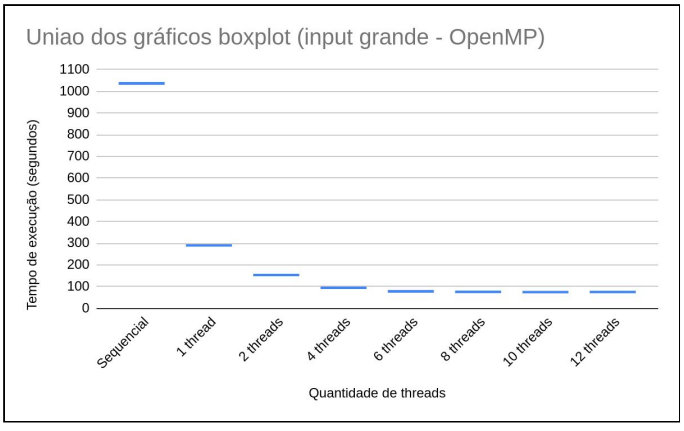
Gráficos 23 e 24. Resultados dos testes com mais de 1 thread para as duas implementações.

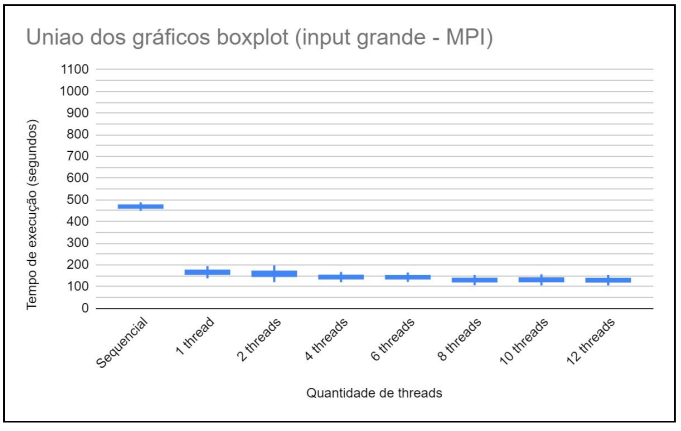




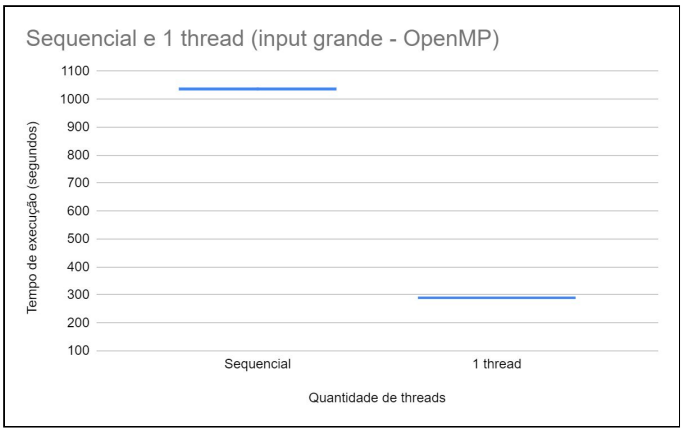
Gráficos 25 e 26. Resultados dos melhores testes das duas implementações.

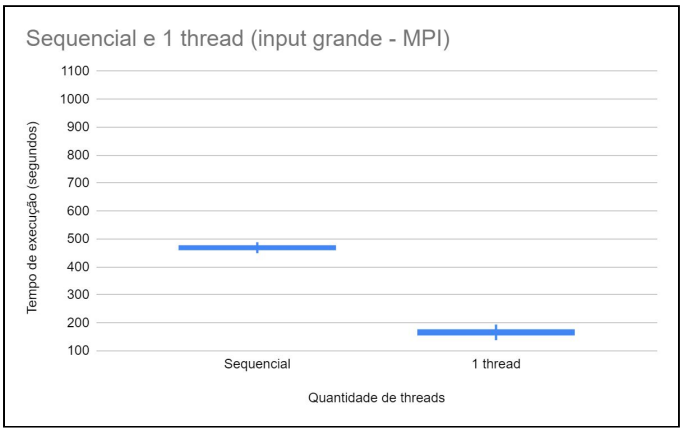
A seguir os gráficos relativos aos testes para o input grande no ambiente 1.



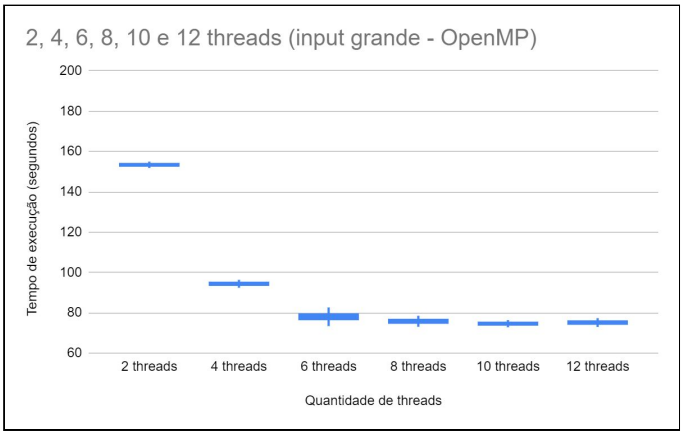


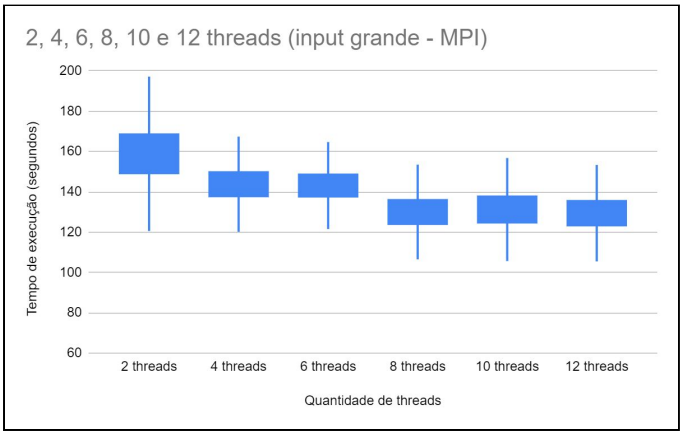
Gráficos 27 e 28. Todos os gráficos boxplot das duas implementações.



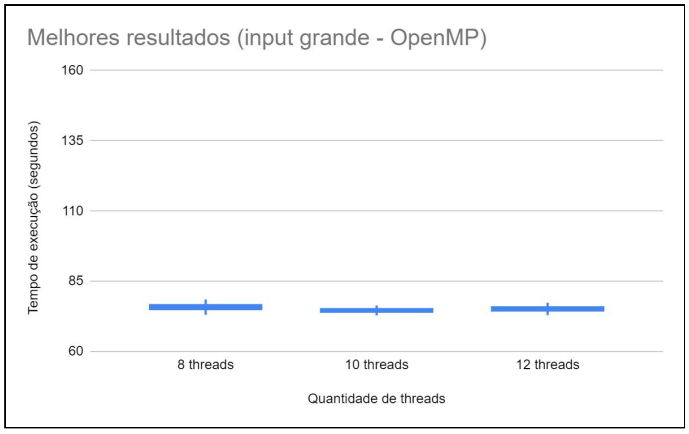


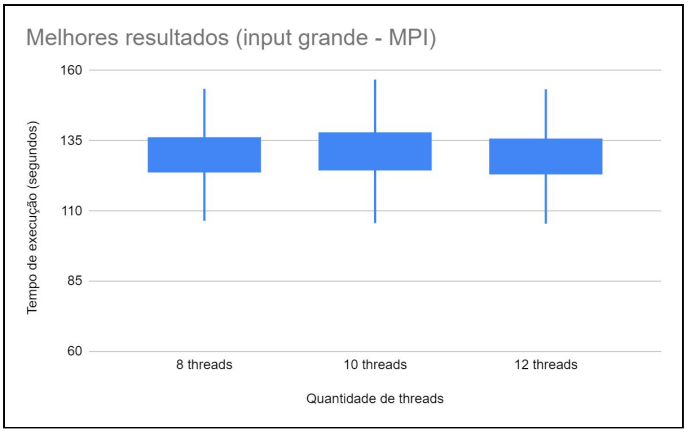
Gráficos 29 e 30. Resultados dos testes sequenciais e com 1 thread, para as 2 implementações.





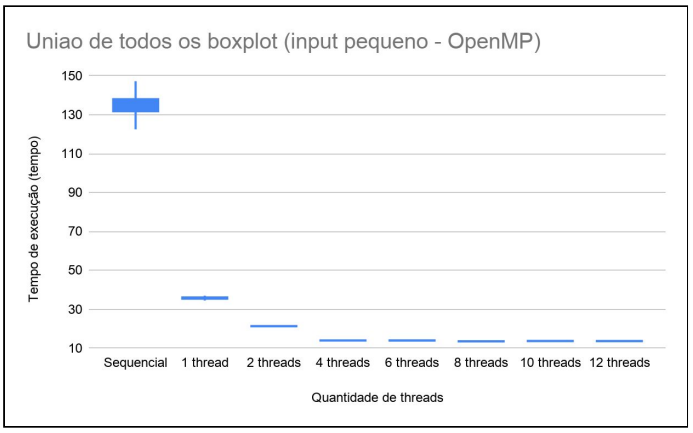
Gráficos 31 e 32. Resultados dos testes com mais de 1 thread para as duas implementações.

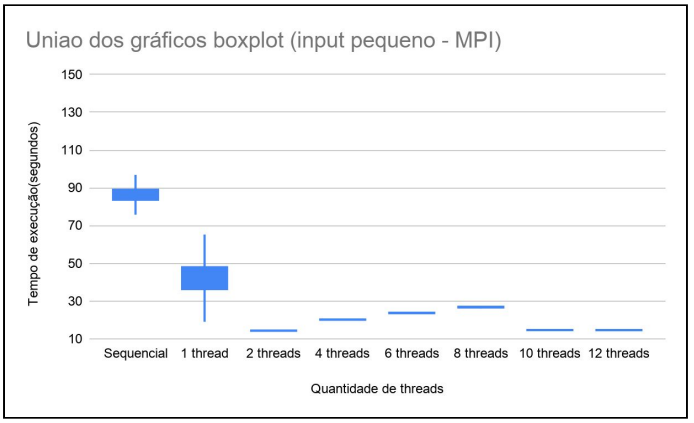




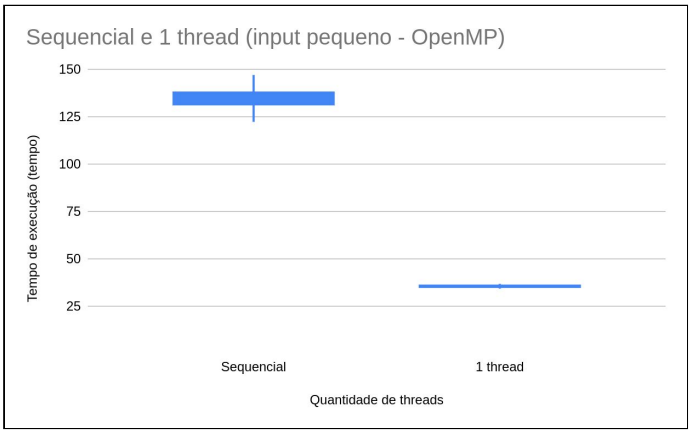
Gráficos 33 e 34. Resultados dos melhores testes das duas implementações.

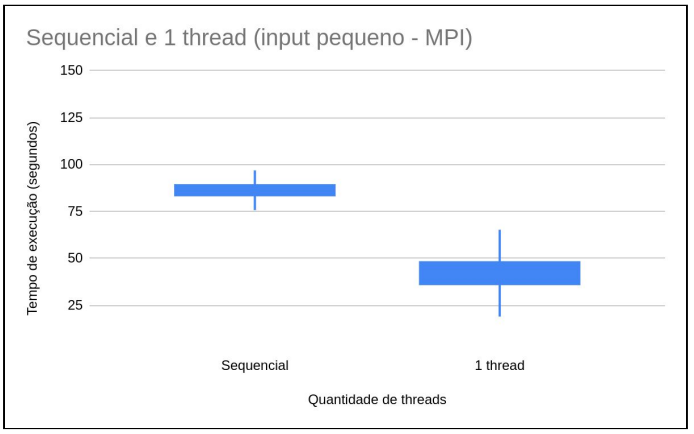
No ambiente 2, os gráficos boxplot construídos com todos os testes, dos dois trabalhos, são mostrados abaixo. Primeiro os resultados dos testes com tamanho de entrada pequena e depois com tamanho de entrada grande.



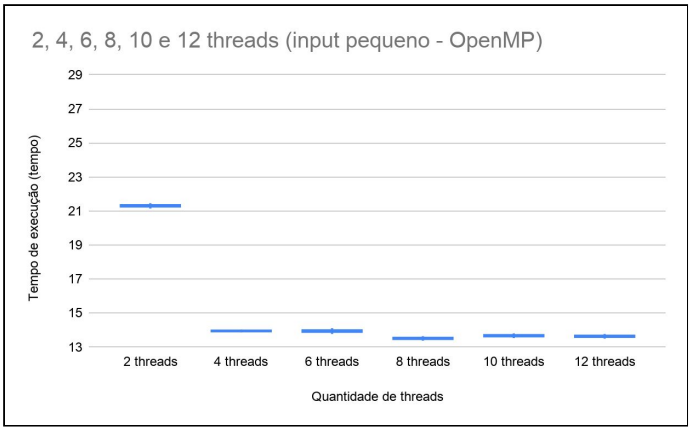


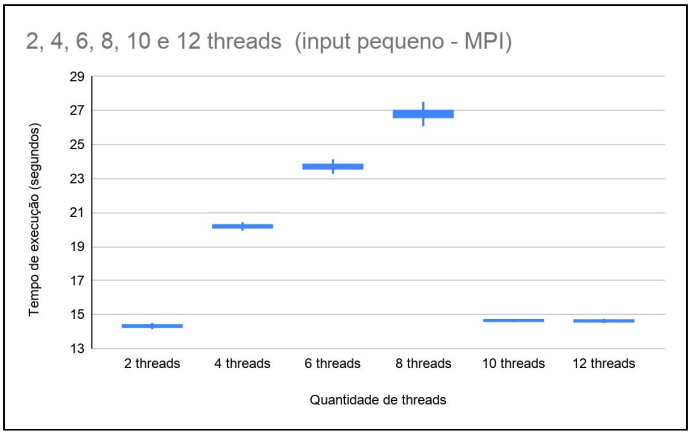
Gráficos 35 e 36. Todos os gráficos boxplot das duas implementações.



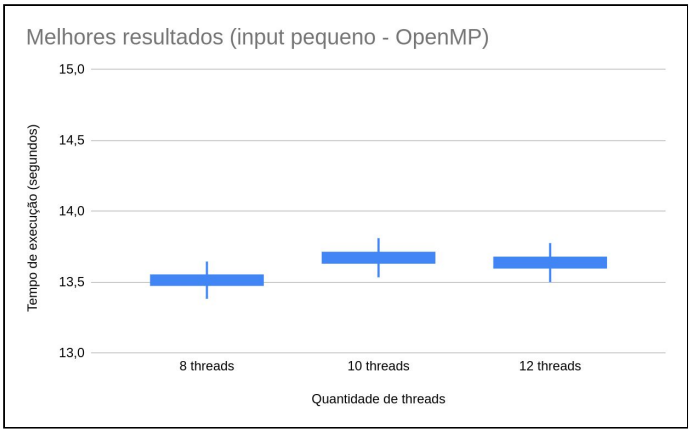


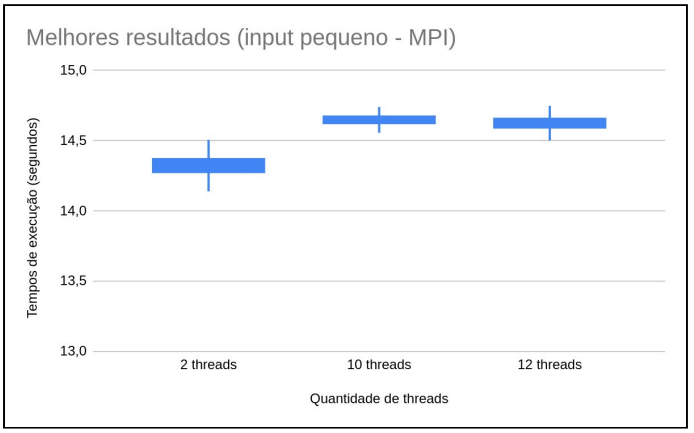
Gráficos 37 e 38. Resultados dos testes sequenciais e com 1 thread, para as 2 implementações.





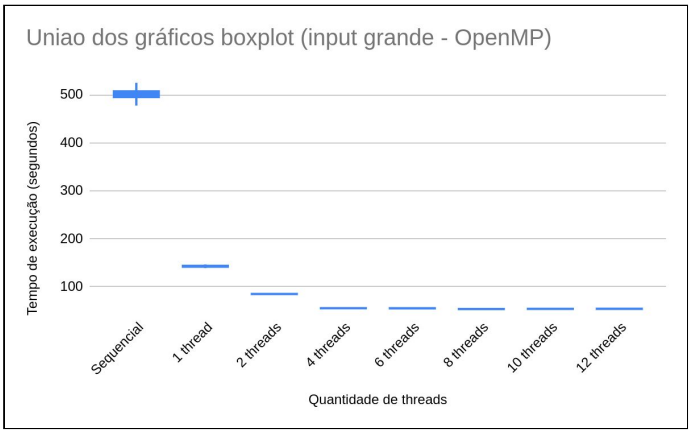
Gráficos 39 e 40. Resultados dos testes com mais de 1 thread para as duas implementações.

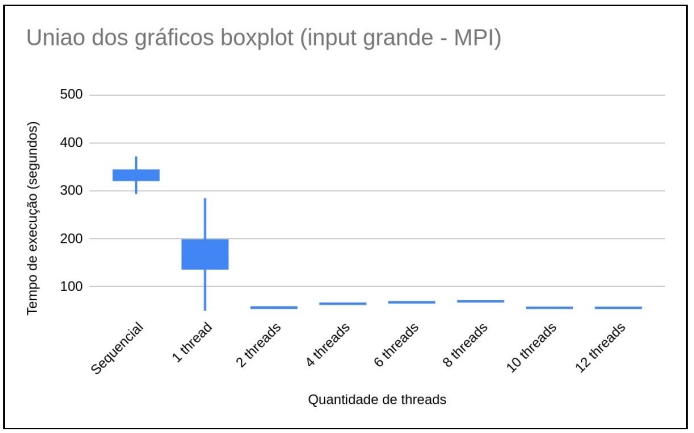




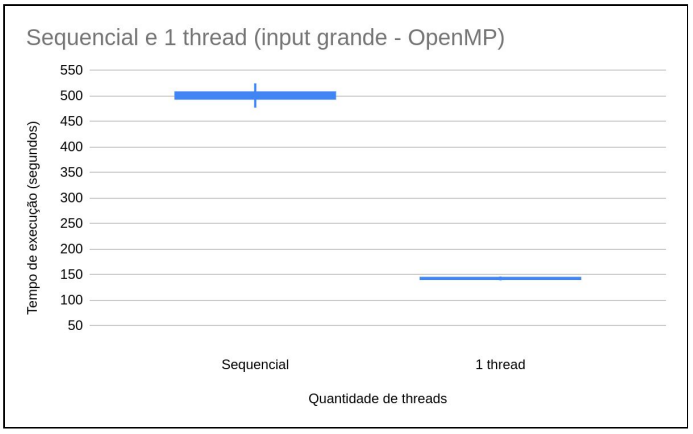
Gráficos 41 e 42 Resultados dos melhores testes das duas implementações.

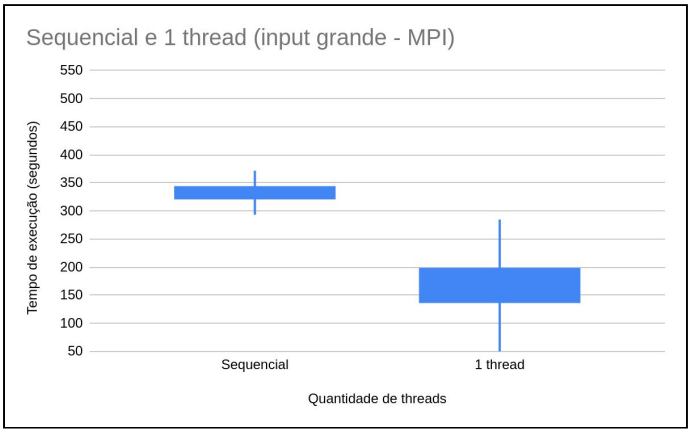
Abaixo são apresentados os gráficos relativos aos testes para o input grande.



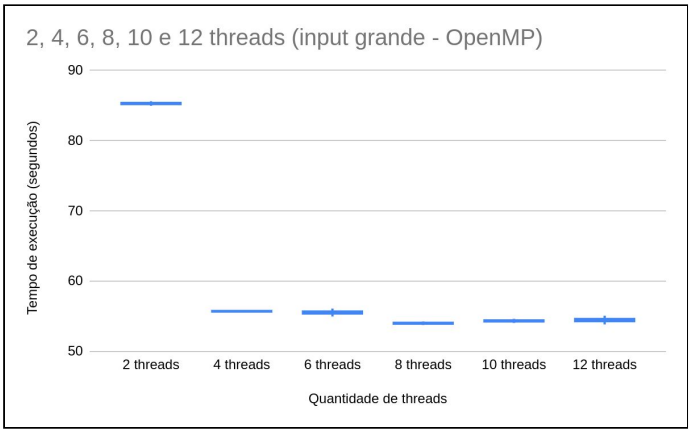


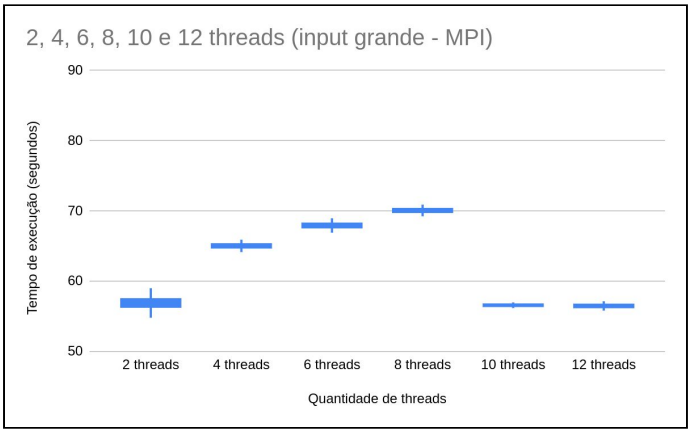
Gráficos 43 e 44.. Todos os gráficos boxplot das duas implementações.



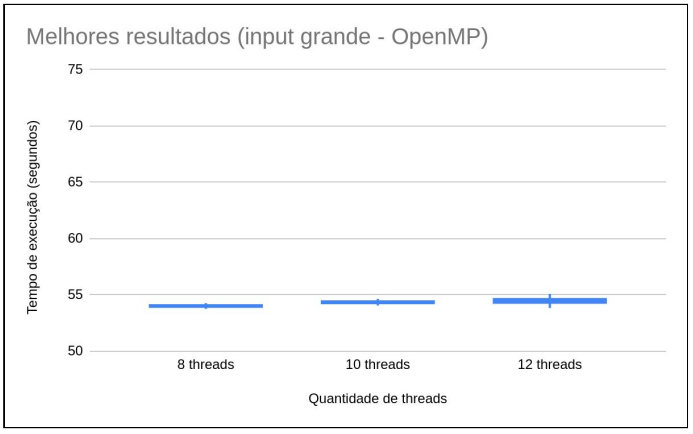


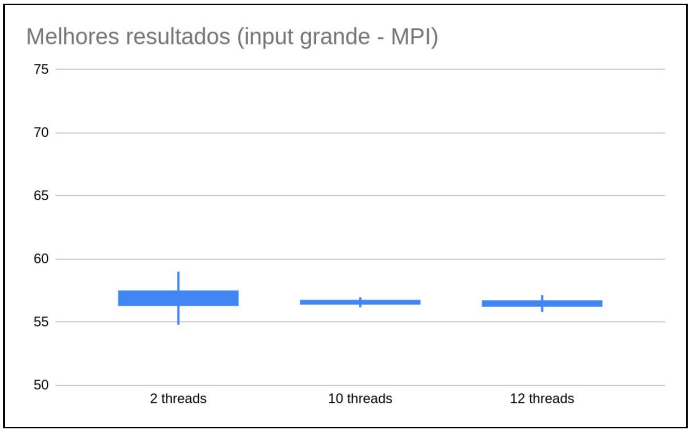
Gráficos 45 e 46. Resultados dos testes sequenciais e com 1 thread, para as 2 implementações.





Gráficos 47 e 48. Resultados dos testes com mais de 1 thread para as duas implementações.





Gráficos 49 e 50. Resultados dos melhores testes das duas implementações.

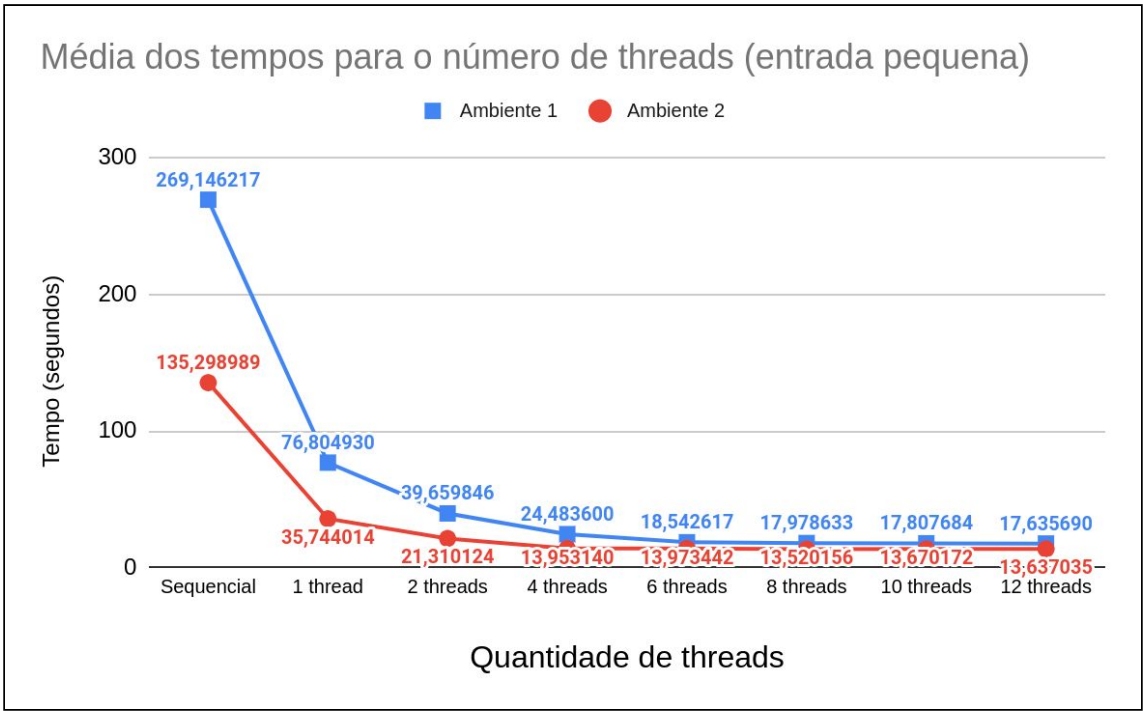


Gráfico 51. Médias dos tempos dos testes para o número de threads da implementação apenas com OpenMP.

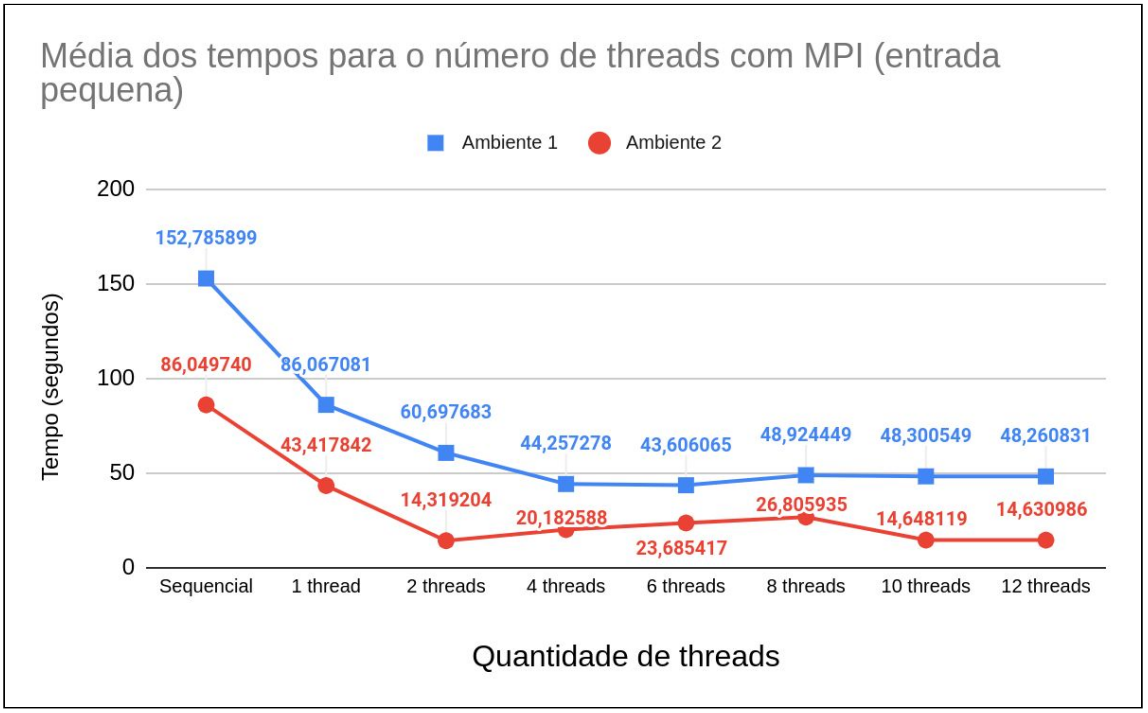


Gráfico 52. Médias dos tempos dos testes para o número de threads da implementação com MPI.

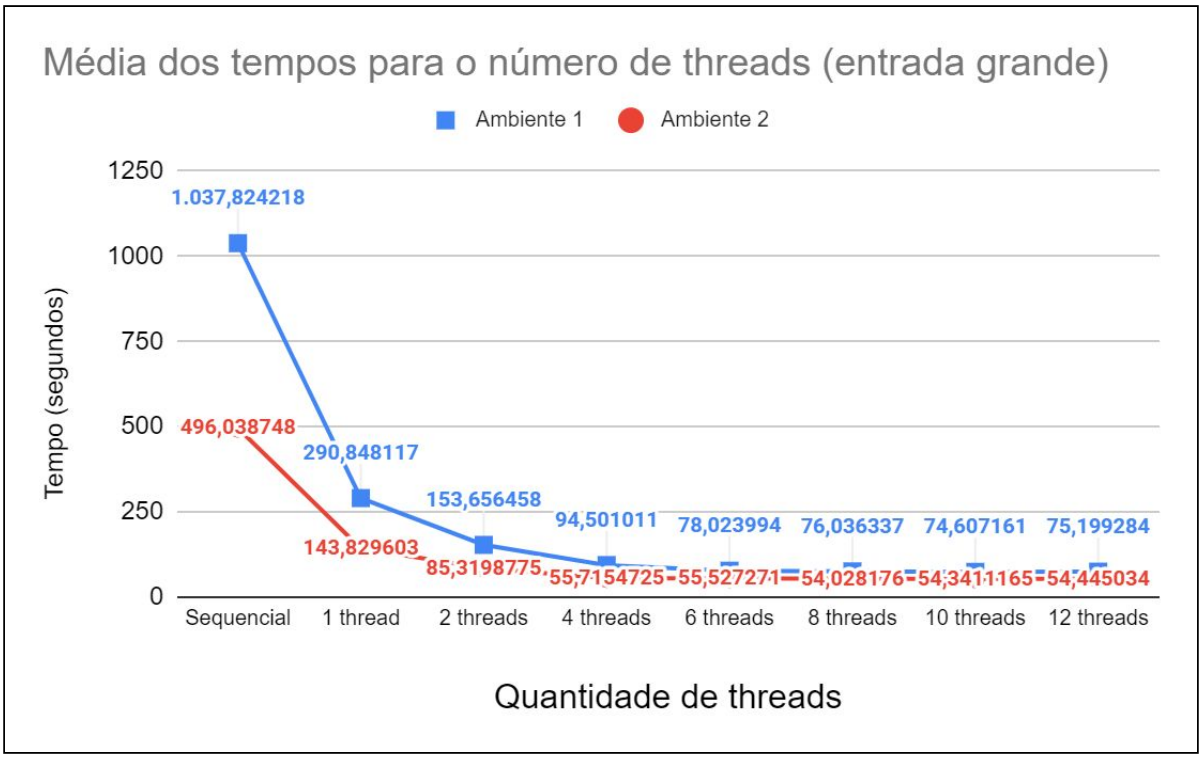


Gráfico 53. Médias dos tempos dos testes para o número de threads da implementação apenas com OpenMP.

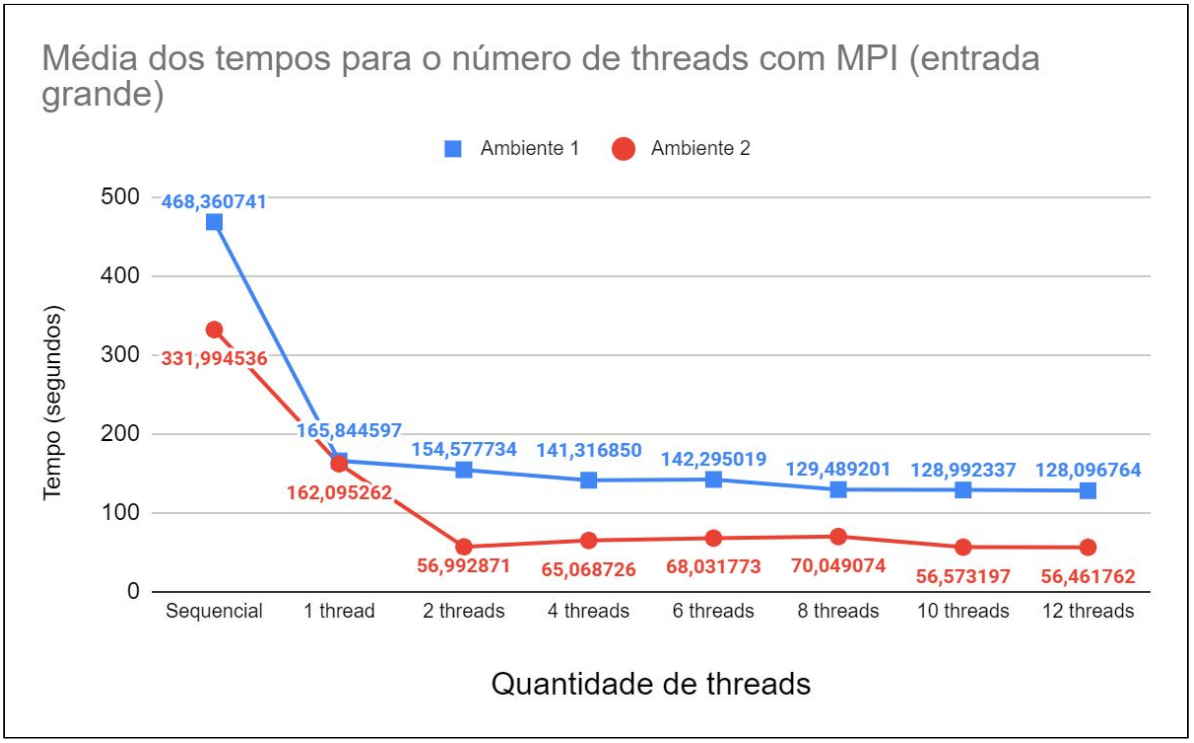


Gráfico 54. Médias dos tempos dos testes para o número de threads da implementação com MPI.

Observando os gráficos acima é possível notar que a quantidade de processos MPI acaba influenciando negativamente o desempenho do programa conforme o número de threads aumenta, pois fica evidente que não existe um ganho alto conforme se adiciona mais threads ao programa, sendo que para os testes com input pequeno em ambos ambientes os melhores resultados médios foram com menos threads, 6 threads e 2 threads, respectivamente. Acreditamos que a razão da piora possa ser a mesma do porque, no trabalho um, o aumento do número de threads não necessariamente melhorou o desempenho do programa, sendo que atribuímos essa piora ao overhead gerado pelas trocas de contexto que acontecem entre as threads, e aqui neste trabalho ainda se tem o adicional de troca de contexto entre processos o que gera um overhead ainda maior.

Um caso especial foi os testes com uma thread. Nos dois ambientes este teste obteve variações de tempo bem grandes. É possível notar isto ao observar o gráfico 22, onde a caixa do gráfico para uma thread é muito maior do que o de costume ou observado em outros gráficos. Isto ocorre pois os resultados foram muito distribuídos. Após muita reflexão, não conseguimos concluir o motivo. Possivelmente tem a ver com o escalonador do SO, sendo a maneira como ele está distribuindo parcelas de tempos para cada processo utilizar os recursos um fator importante nessa oscilação entre os tempos, pois nesse caso até mesmo a prioridade que o SO da para os processos gerados pelo MPI podem influenciar bastante nos resultados finais. Para exemplificar com números, considerando o tamanho de input pequeno e o ambiente de execução 1, várias vezes o tempo de execução foi de 120 segundos, porém existiram vários tempos abaixo de 90 segundos, e até abaixo de 60 segundos. No ambiente de execução 2 isso aconteceu com menor frequência mas também aconteceu.

Também é possível observar pelos gráficos que existe uma melhora de tempo para 10 e 12 threads. Em nossa reflexão sobre os resultados achamos que possivelmente essa melhora ocorra devido a dois fatores, sendo o primeiro a quantidade de threads disparadas pelos processos, o que faz com que mais de um processo execute em paralelo utilizando um bom número de threads, e, como segundo fator, o tamanho da entrada/tempo necessário para processar toda entrada. Esse segundo fator fica mais evidente para o input grande, onde naturalmente é mais demorado o processamento, sendo possível notar isso no gráfico 54, os resultados obtidos conforme aumenta o número de threads e isso pode evidenciar, não temos certeza, um amortecimento das trocas de contexto geradas pelo escalonador do SO em função do crescimento do número de threads utilizadas. É complicado pontuar exatamente a razão da performance obtida tendo em vista que não estamos focando no trabalho de um escalonador e MPI não foi concebido para processamento unicamente local e sim distribuído.

Testes de hipótese

Os melhores resultados, levando em conta o input pequeno e grande, utilizando MPI foram com 12 threads, por esse motivo vamos supor como hipótese que, para 12 threads, os dois programas são equivalentes (com ou sem MPI).

Sejam M_1 a média de tempo de execução do programa do trabalho anterior com 12 threads para o tamanho de entrada pequeno e M_2 a média de tempo de execução do programa do trabalho atual (com MPI) com 12 threads para o tamanho de entrada pequeno,

H0: $M_1 = M_2$
H1: $M_1 \neq M_2$

Utilizando o teste T de student, tendo uma variância combinada de aproximadamente 0,9433 no ambiente 1 e 0,01574 no ambiente 2, nível de significância de 0,05 e grau de liberdade 58, têm-se o valor do cálculo da seguinte forma:

$t_0 = (M_1 - M_2)/ \sqrt{s^2_{1,2} * (1/N_1 + 1/N_2)}$,

onde $s^2_{1,2}$ é a variância combinada das amostras do programa sem MPI com 12 threads e do programa com MPI com 12 threads, e N_1 e N_2 são os tamanhos das amostras.

Desta forma t_0 = 472,9779518 para o ambiente 1 e t_0 = 118,9811434 para o ambiente 2.

Com o grau de liberdade 58 e nível de significância de 0,05, o valor crítico do teste bilateral é de 2,0017 (positivo ou negativo). Sendo assim, o teste falha para os dois ambientes para o tamanho de entrada pequeno, significando que as médias de tempo de execução não são iguais.

Da mesma forma, para o tamanho de entrada grande:

Sejam M_1 a média de tempo de execução do programa do trabalho anterior com 12 threads para o tamanho de entrada grande e M_2 a média de tempo de execução do programa do trabalho atual (com MPI) com 12 threads para o tamanho de entrada grande,

H0: $M_1 = M_2$
H1: $M_1 \neq M_2$

Utilizando o teste T de student, tendo uma variância combinada de aproximadamente 1,810308885 no ambiente 1 e 0,05338879356 no ambiente 2, nível de significância de 0,05 e grau de liberdade 58.

Desta forma t_0 = 589,7254909 para o ambiente 1 e t_0 = 130,9222439 para o ambiente 2.

Com o grau de liberdade 58 e nível de significância de 0,05, o valor crítico do teste bilateral é de 2,0017 (positivo ou negativo). Novamente, em ambos os ambientes a hipótese é provada falsa, sendo assim os tempos não são equivalentes.

Observando nos gráficos e nos dados dos testes, é possível perceber que a implementação sem MPI possui tempo de execução médio menor para todas as combinações testadas, sendo assim, é a implementação com melhor performance em relação ao tempo de execução.

Conclusões

Neste trabalho nos propomos a implementar uma versão do problema NBody utilizando a ferramenta MPI em conjunto com a ferramenta OpenMP. Após a implementação e os testes foi possível concluir que a adição da ferramenta MPI não necessariamente aumenta o desempenho do programa. Nos casos testados o aumento de desempenho aconteceu apenas nos que antes eram sequenciais ou com apenas uma thread utilizando OpenMp, pois nesses casos a segmentação do problema em 4 processos gerou paralelização do programa, o que trouxe ganhos na execução. Para outros casos, com mais threads no pool do OpenMP, a adição de MPI piorou o desempenho. Isto ocorre pois existe um overhead grande para o sistema em lidar com 4 processos ao invés de um só como era na implementação anterior.

Gostaríamos de destacar, porém, que a ferramenta MPI poderia sim aumentar o desempenho do programa se fosse utilizada em uma rede de computadores distribuídos, e não rodando localmente como foi esta implementação, porém não era o foco do trabalho esta implementação em outro host, o que dificultaria a comparação com o trabalho anterior em OpenMP. Contudo, nos baseando nos resultados da implementação com OpenMP é bastante plausível pressupor que se nossa implementação com OpenMP + MPI fosse executada em um modelo de rede distribuído o desempenho certamente seria superior ao que obtemos utilizando apenas OpenMP, pois em um cenário distribuído cada processo seria um nó diferente na rede que iria executar uma parte do programa NBody utilizando o paralelismo informado através do número de threads setadas no OpenMP, assim sendo, cada nó executaria um único processo com o paralelismo de threads informadas ao programa, sendo este cenário idêntico ao que obtemos bons resultados em nosso primeira implementação, isso nos leva a crer que em um cenário ideal a implementação atual seria superior a anterior.

Contudo, nós não podemos afirmar o quão superior seria essa implementação/execução, pois existem outros fatores de overhead que não temos conhecimento de quanto influenciam, por exemplo, o overhead de comunicação na rede. O que podemos concluir é que localmente o uso de processos MPI na implementação inicial OpenMP prejudicou ao invés de melhorar os tempos de execução. Como ideia futura, fica a curiosidade de testar esta implementação em um cenário distribuído.

Todos os testes, tabelas e gráficos feitos foram produzidos em uma planilha que encontra-se disponível aqui. A planilha não possui explicações, apenas os dados brutos dos testes.