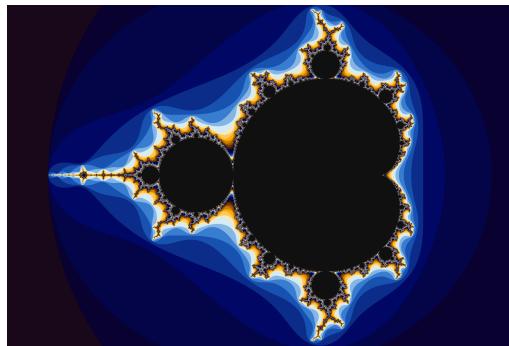


INSTITUTO DE MATEMÁTICA E ESTATÍSTICA - USP

MAC5742-0219

Relatório do EP3

Cálculo Distribuído do Conjunto de Mandelbrot



Integrantes do Grupo:

Bruno Ferrero N° USP: 3690142

Marcelo Galdino N° USP: 10120500

Willy Reis N° USP: 7694112

Professor: Alfredo Goldman

Monitor: Pedro Bruel

S Ã O P A U L O

Julho de 2017

Sumário

1	Introdução	2
2	MPI x OpenMP	3
3	Experimentos	4
3.1	Implementação	4
3.2	Experimentos	5
4	Resultados dos Experimentos	6
4.1	10 medições com 1 iteração	6
4.2	1 medição com 10 iterações	9
4.3	Comparativo entre os experimentos	11
5	Conclusão	14

1. Introdução

O presente EP trata de uma família de fractal denominada de *Conjunto de Mandelbrot*¹. O objetivo deste EP é explorar a característica “*facilmente paralelizável*” do problema e com isso utilizar técnicas e recursos de paralelização (*Pthreads* e *OpenMP*) para comparar o desempenho de cada uma das tecnologias.

Abaixo ilustramos as 4 figuras indicando as regiões do fractal que serão calculadas pelos códigos implementados.

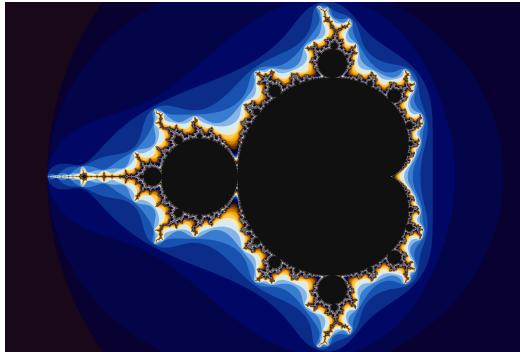


Figura 1: *Full Picture*

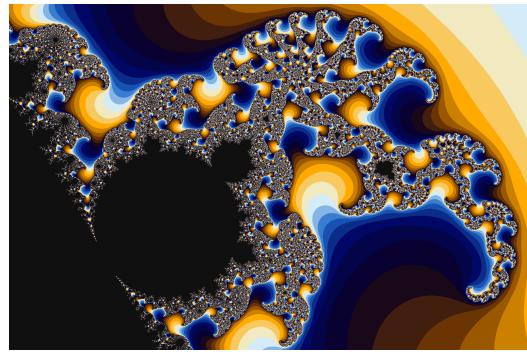


Figura 2: *Elephant Valley*

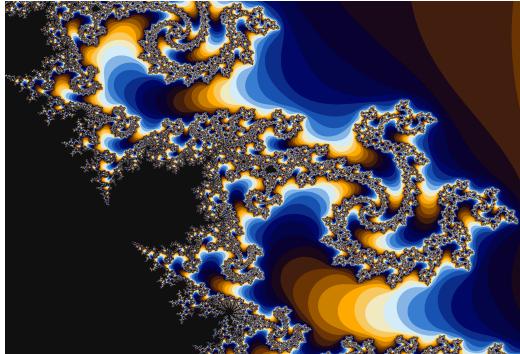


Figura 3: *Triple Spiral Valley*

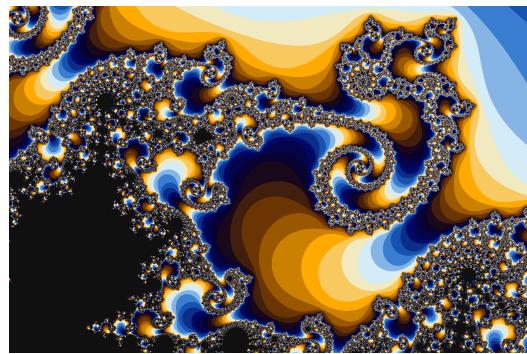


Figura 4: *Seahorse Valley*

Figura 5: Regiões do Conjunto de Mandelbrot

Diferentemente dos primeiro EP, neste trabalho é abordado o uso de computação distribuída e paralela através do padrão MPI e das diretivas de compilação OpenMP.

O objetivo deste trabalho é comparar o tempo gasto para gerar o Conjunto de Mandelbrot de maneira distribuída (sem compartilhamento de memória) e de maneira local (com compartilhamento de memória), levando em consideração um número fixo de processadores utilizados, além da comparação do algoritmo sequencial com o algoritmo distribuído.

¹https://en.wikipedia.org/wiki/Mandelbrot_set

2. MPI x OpenMP

OpenMP é uma interface de programação (API), portável, baseada no modelo de programação paralela de memória compartilhada para arquiteturas de múltiplos processadores.

É composto por três componentes básicos:

- Diretivas de Compilação;
- Biblioteca de Execução;
- Variáveis de Ambiente.

OpenMP está disponível para uso com os compiladores C/C++ e Fortran, podendo ser executado em ambientes Unix e Windows (Sistemas Multithreads). Definido e mantido por um grupo composto na maioria por empresas de hardware e software, denominado como OpenMP ARB (Architecture Review Board).

MPI Message Passing Interface é um padrão que especifica um modelo (um conjunto de interfaces) e não sua implementação (como). O modelo de programação de passagem de mensagem é bastante poderoso por ser fácil de usar e possui alta portabilidade. A facilidade de uso se justifica pelo uso de interfaces simples claramente especificadas quanto ao seu comportamento, uso e parâmetros para operação. A portabilidade pelo fato de que com o uso passagem de mensagem, é possível escrever programas que irão funcionar em máquinas de tipos variados como por exemplo multiprocessadores com memória distribuída, redes de estações de trabalhos ou ambientes híbridos dos mais variados tipos. A idéia geral é permitir escrever aplicações que usem passagem de mensagem como modelo de programação de forma padronizada possibilitando interoperabilidade entre os programas e consequente aumento de escalabilidade. Atualmente, o padrão é utilizado principalmente em máquinas paralelas e clusters (conjuntos) de estações de trabalho.

3. Experimentos

Neste trabalho foi realizada a implementação do problema do Conjunto de Mandelbrot, utilizando MPI e OpenMP.

Foram realizados quatro tipos de experimentos, variando o número de instâncias da máquina virtual n1-standard do Google Cloud Engine e o número de processadores por instâncias, mas mantendo fixo o número de processadores totais.

3.1. Implementação

O problema do *Conjunto de Mandelbrot*² é um problema embaraçosamente ou perfeitamente paralelizável, isto é, pode ser paralelizado de maneira trivial ou complexa, uma vez que cada ponto da imagem é calculado de maneira independente, não havendo necessidade de comunicação entre as tarefas paralelas.

Dada a trivialidade do problema, em questão de paralelização, foi realizada a divisão da estrutura de dados em partes de tamanho equivalente, para computação de maneira independente distribuidamente e paralelamente quando possuindo mais de uma instância ou mais de um processador por instância.

A implementação utilizando MPI foi feita com comunicação ponto a ponto através das funções *MPI_Send* e *MPI_Recv*. Dada a entrada do problema, contendo a área, o tamanho da do Conjunto de Mandelbrot a ser calculado e o número de instâncias (ou pontos) e *threads*, o programa aloca espaço para a imagem a ser criada de maneira sequencial.

Para cada instância então é alocado um espaço em memória equivalente a uma fração do tamanho da imagem real, divididos igualmente entre as instâncias. Para cada instância, o subproblema ainda é dividido de acordo com o número de processadores desta utilizando a implementação em OpenMP do EP1.

Assim que cada instância termina seu processamento, o conteúdo gerado é enviado para a instância mestre, responsável por transferir esse conteúdo para o espaço reservado para o resultado.

Uma alteração importante nesta implementação foi a alteração da estrutura de dados em matriz para uma representação em um vetor, cujos dados referentes as cores (RGB) de cada pixel são alocados juntos e em sequência.

²B. B. Mandelbrot - The Fractal Geometry of Nature, Ed. Freeman

3.2. Experimentos

Foram feitos duas variações do mesmo experimento. Em ambas foram feitas 10 execuções com uma entrada de tamanho 8192×8192 para cada uma das áreas do conjunto de Mandelbrot e para cada uma das configurações a seguir:

- Sequencial
 - 1 instância n1-standard-8
- Paralela
 - 1 instância n1-standard-8
 - 2 instâncias n1-standard-4
 - 4 instâncias n1-standard-2
 - 8 instâncias n1-standard-1

A diferença entre as execuções se dá na maneira como ela foi realizada. Na primeira foram realizadas 10 medições do perf, com 1 iteração, o que nos permitiu ter dados para a geração de boxplots. Na outra foi realizada uma medição com 10 iterações do perf, nos gerando os dados para o cálculo do desvio padrão.

4. Resultados dos Experimentos

4.1. 10 medições com 1 iteração

Neste teste analisamos principalmente a variação de tempo nas execuções.

As figuras 6 mostra a variação de tempo para a execução sequencial. Neste algoritmos, como não há comunicação e nem trocas de contexto, houve pouca variação nos resultados obtidos, com mais variação de menos de 0.5 segundos em mais de 50% das execuções.

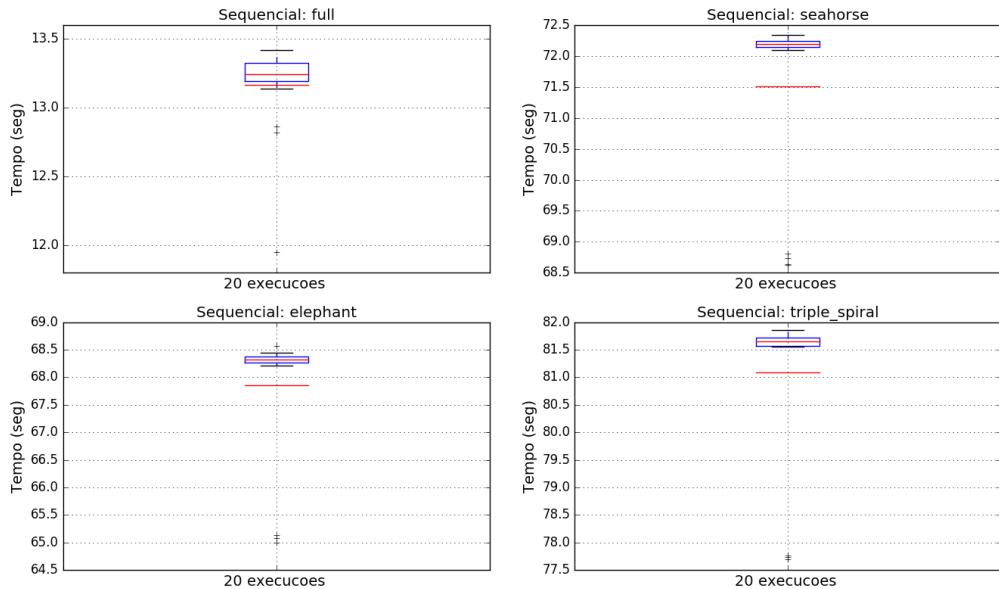


Figura 6: Boxplot do tempo de execução do experimento sequencial.

Nas próximas figuras são analisados o desempenho de cada configuração dos experimento por área.

Na área completa, 7 é possível notar que a maioria das execuções, independente da configuração da execução, ficou com um desempenho semelhante (dentro de 0.3 segundos), com exceção do teste para 2 instâncias de 4 cores, que possui uma maior variação dos principais 50%. Em comparação com o algoritmo sequencial, o comportamento dos algoritmos quanto a variação é similar.

Já na área elephant valley 8 e na triple spiral 10, a variação entre o primeiro e o terceiro quartis são maiores do que 1 segundo, com exceção do experimento com duas instâncias. Na figura 9, a variação ficou menor do que 1 segundo.

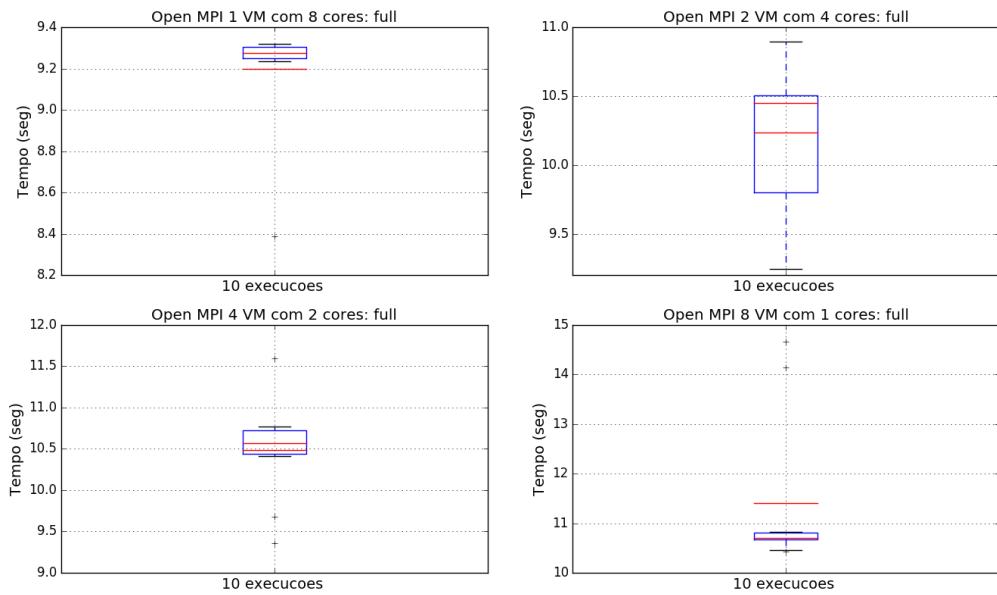


Figura 7: Boxplot do tempo de execução para a área completa

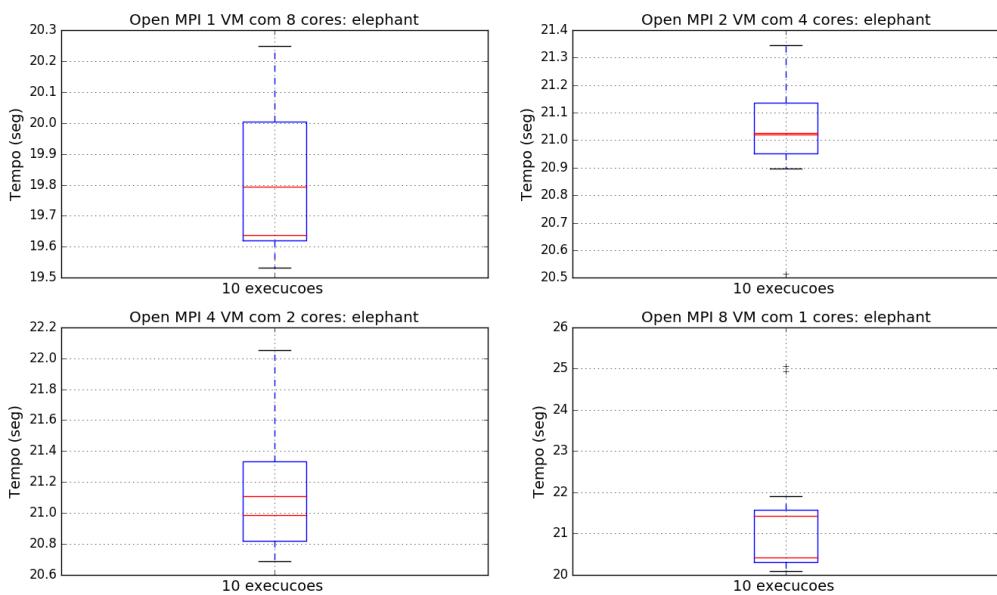


Figura 8: Boxplot do tempo de execução para a área elephant valley

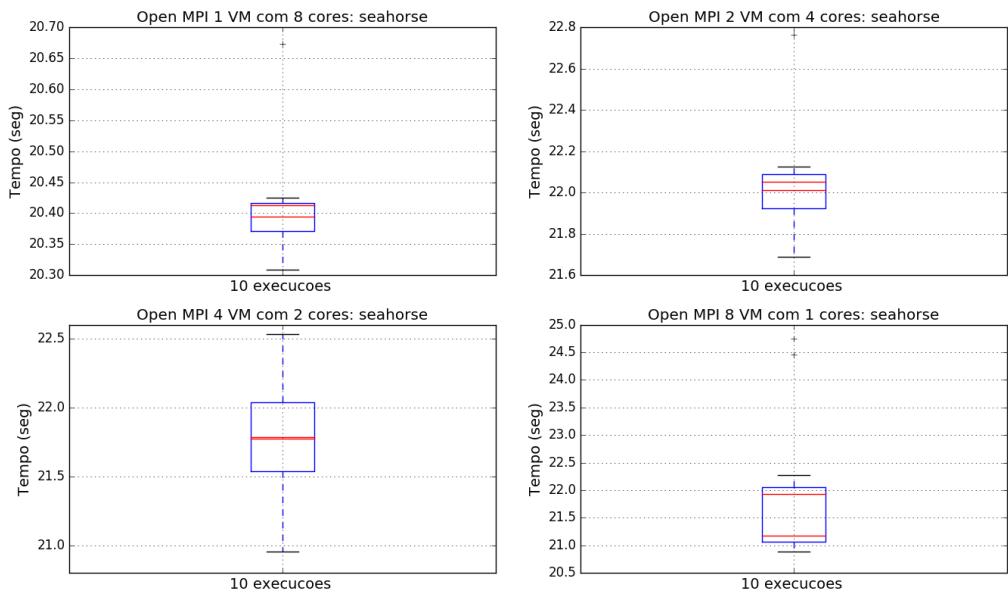


Figura 9: Boxplot do tempo de execução para a área seahorse valley

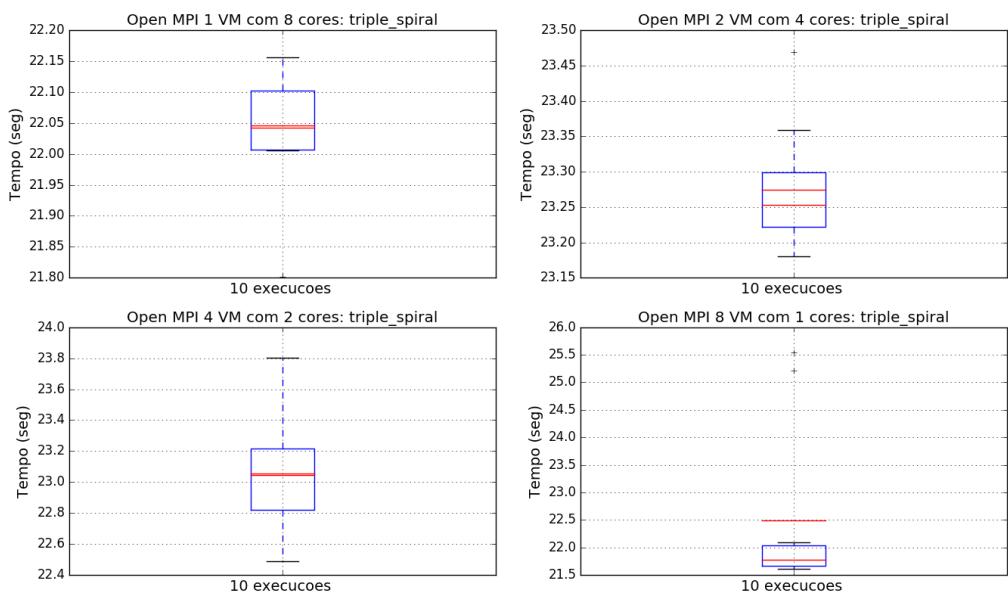


Figura 10: Boxplot do tempo de execução para a área triple spiral

4.2. 1 medição com 10 iterações

Nesses experimentos, com a obtenção do desvio padrão dos resultados, foi possível obter a variação do tempo de execução de acordo com cada uma das variações do experimento e sua respectiva variação no desvio padrão (representados pelas cores azul como limite superior e verde como limite inferior).

Na figura 11, vemos os experimentos realizados com mais de uma instância ficaram com um tempo de execução abaixo do experimento com apenas uma instâncias e paralelo. Isso pode acontecer devido à comunicação necessária entre as instâncias. Ouve uma leve melhora na execução com 8 instâncias de um processador, embora o alcance do desvio padrão também tenha sido maior.

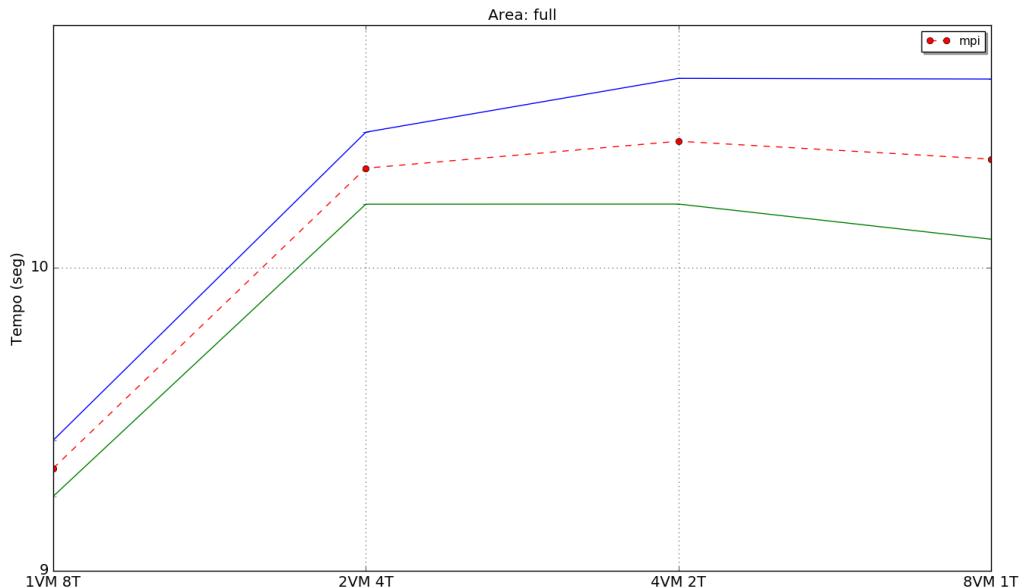


Figura 11: Variação do tempo de execução para a área full valley

No teste da área elephant valley, figura 12, vemos novamente um melhor tempo de execução do experimento em uma instância, e uma melhora do experimento de 8 instâncias em detrimento aos outros experimentos distribuídos. O mesmo comportamento se repete na área seahorse valley, figura 13.

No entanto, no experimento da área triple spiral, figura 14, o algoritmo totalmente distribuído teve o melhor desempenho. superando o algoritmo totalmente paralelo, inclusive considerando o desvio padrão.

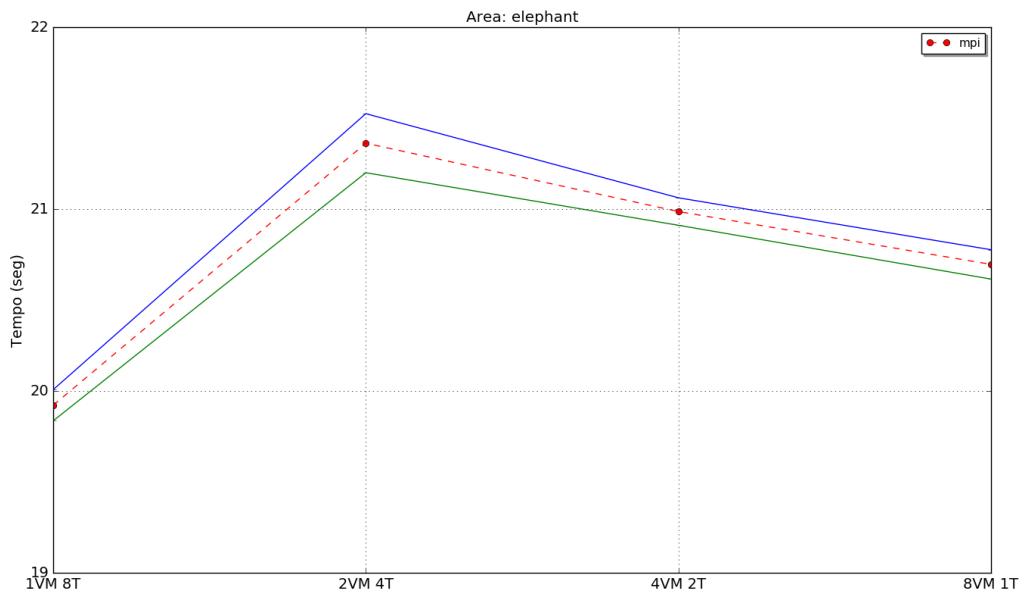


Figura 12: Variação do tempo de execução para a área elephant valley

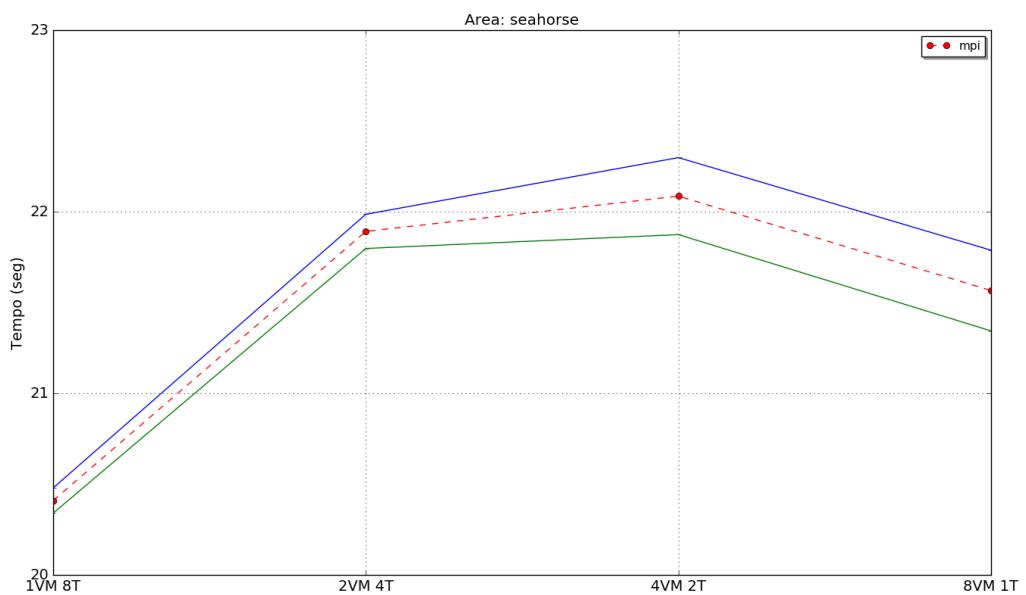


Figura 13: Variação do tempo de execução para a área seahorse valley

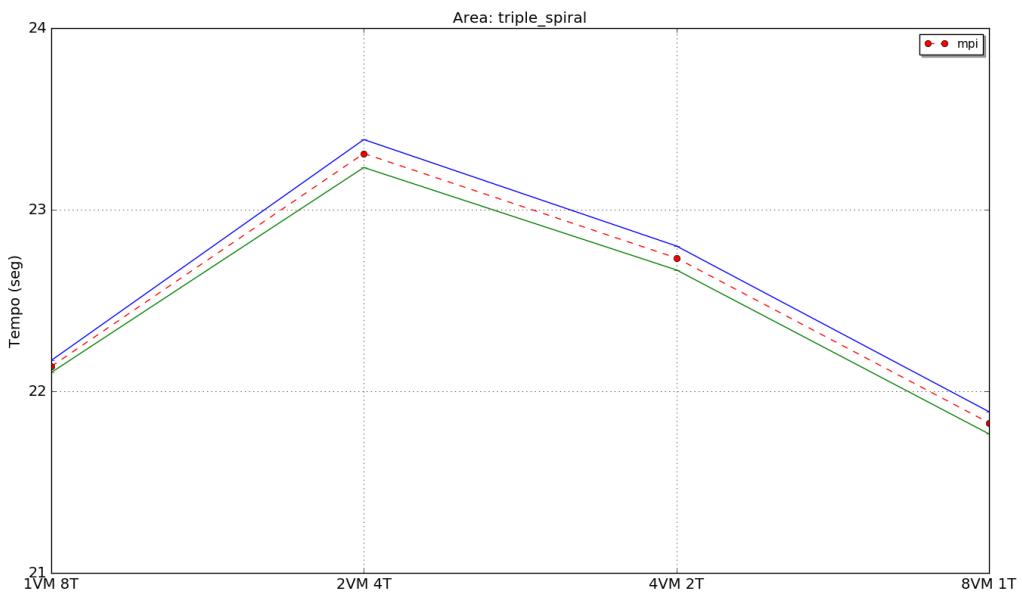


Figura 14: Variação do tempo de execução para a área triple spiral

4.3. Comparativo entre os experimentos

Nos experimentos das figuras 15 a 18, analisamos se há alguma diferença no tempo médio de execução levando em consideração a forma como os experimentos foram realizados. No geral o comportamento é similar.

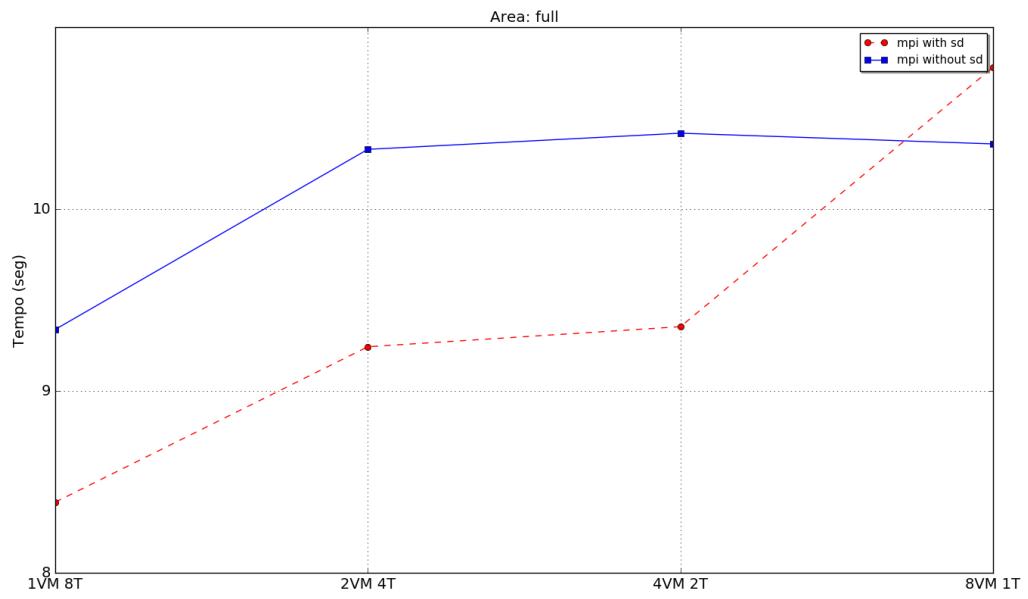


Figura 15: Variação do tempo de execução para a área full valley

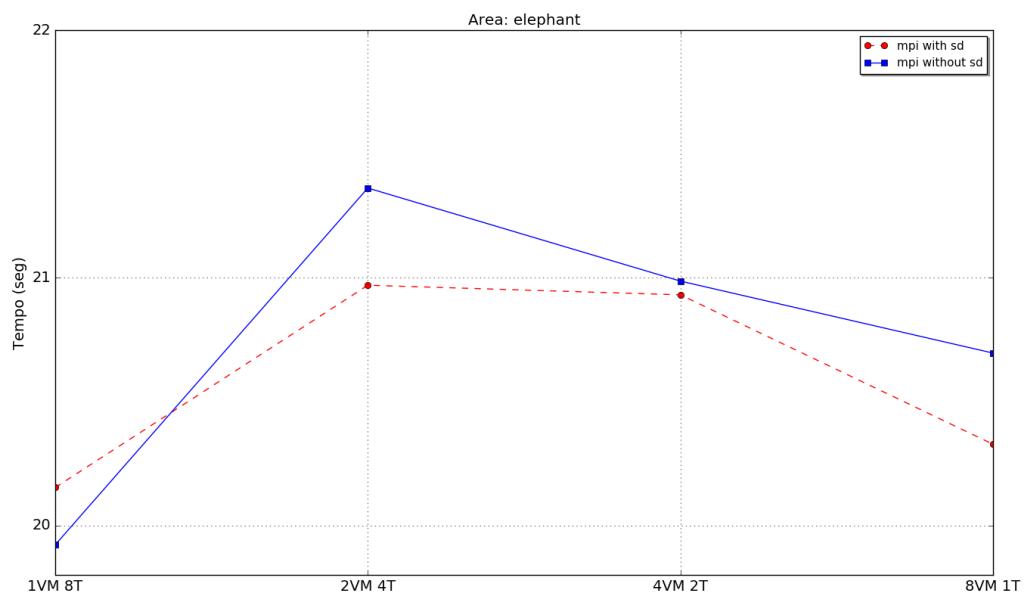


Figura 16: Variação do tempo de execução para a área elephant valley

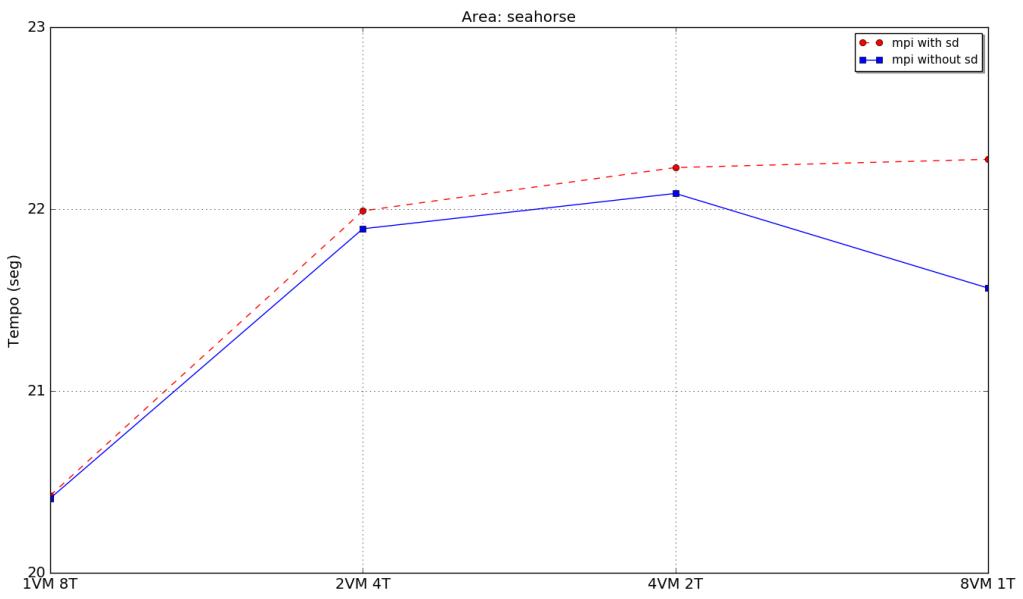


Figura 17: Variação do tempo de execução para a área seahorse valley

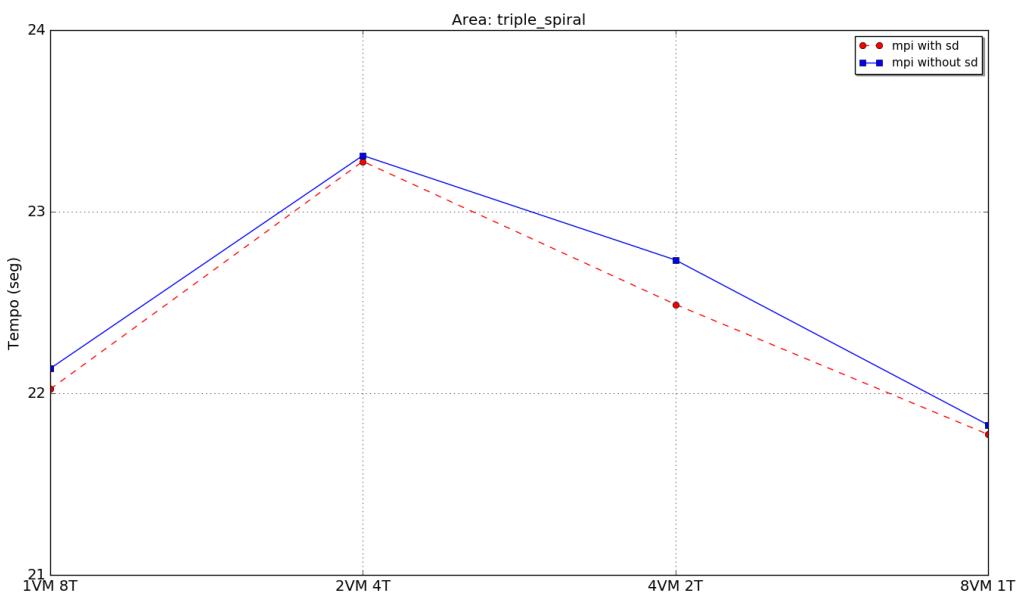


Figura 18: Variação do tempo de execução para a área triple spiral

5. Conclusão

Dada a natureza de incerteza da paralelização de *threads*, é possível, e bastante provável, que execuções distintas ocorrerão de maneira distintas no processador, no sentido de que a ordem de disponibilização de recursos computacionais para execução da *thread* será totalmente diferente a cada execução. Por esse motivo, um maior número de execuções propicia uma melhor confiabilidade nos resultados obtidos, uma vez que é possível reduzir as chances de execuções estranhas e discrepantes do normal.

O problema do Conjunto de Mandelbrot é um problema facilmente paralelizável, o que significa que não há a necessidade de troca de mensagens por *threads* distintas. Em outras palavras, cada *thread* pode executar o seu trabalho de maneira independente.

Com os experimentos realizados não foi possível obter um resultado satisfatório sobre o desempenho geral de um algoritmo paralelo e distribuído com o mesmo poder computacional.

No geral o algoritmo totalmente paralelo foi um pouco superior, mas o algoritmo totalmente distribuído possuía um bom desempenho também, e em um caso foi superior ao paralelo.