MANDELBROT

BIBLEOTECAS UTILIZADAS:
Pthread e OpenMP

ESTRUTURA DE DIRETÓRIOS:
A estrutura do diretório do projeto é organizada da seguinte maneira:

Pasta "codes":
Contém os códigos fonte relacionados ao conjunto de Mandelbrot, incluindo implementações sequenciais e paralelizadas com OpenMP e Pthread. Também armazena os executáveis gerados durante a compilação.

Pasta "metrics":
Esta pasta é destinada ao armazenamento das métricas de desempenho, como os gráficos de Speedup e Eficiência. Além disso, inclui informações sobre a média do tempo de execução e o intervalo de confiança. As métricas são geradas para cada biblioteca (OpenMP e Pthread), levando em consideração diferentes tamanhos de entrada e números de threads.

Pasta "outputs":
Contém os arquivos de texto (".txt") gerados pelos códigos Mandelbrot. Esses arquivos incluem os resultados obtidos a partir das implementações sequenciais e paralelas. A presença de resultados em formato de texto facilita a comparação entre os outputs gerados pelos diferentes códigos, permitindo a verificação da consistência dos resultados.

Programa "metricsgenerator.ipynb":
Trata-se de um Jupyter notebook dedicado à automação dos testes dos códigos Mandelbrot. Sua principal funcionalidade é gerar gráficos representativos das métricas de desempenho, simplificando a análise dos resultados. O notebook é configurado para realizar testes para cada biblioteca (OpenMP e Pthread) e para diversas combinações de tamanhos de entrada e números de threads, proporcionando uma visão abrangente do desempenho do programa.

Descrição do Software:

O propósito deste software é gerar visualizações do conjunto de Mandelbrot, um fractal definido como o conjunto de pontos c no plano complexo. Inicialmente, o código produzia a representação visual no terminal, tornando difícil comparar os resultados entre as implementações sequencial e paralela. Para solucionar essa questão, o código foi modificado para gerar o fractal em um arquivo de texto (.txt), utilizando o nome do arquivo como "output(nome_da_biblioteca).txt". Essa abordagem facilita a comparação usando o comando diff do Linux.

MODIFICAÇÕES NO CÓDIGO:

As adaptações incluíram a obtenção dos valores de max_row, max_column e max_n a partir dos argumentos de execução do programa (argv) em vez da entrada pelo console (cin). Essa modificação simplifica a execução dos programas e facilita os testes de desempenho.

O código principal contém três loops for. O primeiro é responsável pela alocação dinâmica da matriz, o segundo pela geração do fractal e o terceiro pela escrita no arquivo .txt. Os dois loops extremos, de alocação e escrita, não foram paralelizados, pois não constituíam o gargalo de desempenho. Além disso, o terceiro loop é intrinsecamente sequencial devido à necessidade de manter a ordem na escrita do fractal.

PARALELIZAÇÃO EFICIENTE:

A paralelização concentrou-se no segundo loop, que possui um loop for interno e um while. No entanto, observou-se que a paralelização do loop for interno não resultou em diferenças significativas de desempenho, tanto na biblioteca Pthread quanto na OpenMP. A decisão foi, portanto, realizar a paralelização apenas no loop for externo, otimizando a eficiência do código.

Essas modificações visaram melhorar a eficiência do software, garantindo uma execução eficaz e comparável entre as versões sequencial e paralela, enquanto a estrutura do código foi adaptada para facilitar os testes de desempenho e a análise dos resultados.

AMBIENTE DE TESTE:
Hardware: Ryzen 5 5600 6 núcles/12 threads
Software: WSL2 - Ubuntu 22.04

CONFIGURAÇÕES DE ENTRADA UTILIZADAS:

Para avaliar a eficácia da paralelização do programa, foram adotadas as seguintes configurações de entrada durante os testes:

Números de Threads:
1, 2, 4, 8, 12, 16

Tamanhos de Entrada:
Foram utilizados os valores específicos de 100, 200, 400, 800. Nestes experimentos, os parâmetros max_row, max_column e max_n foram definidos de maneira consistente. Por exemplo, para uma entrada de 800, os valores seriam configurados da seguinte forma: max_row = 800, max_column = 800, max_n = 800.

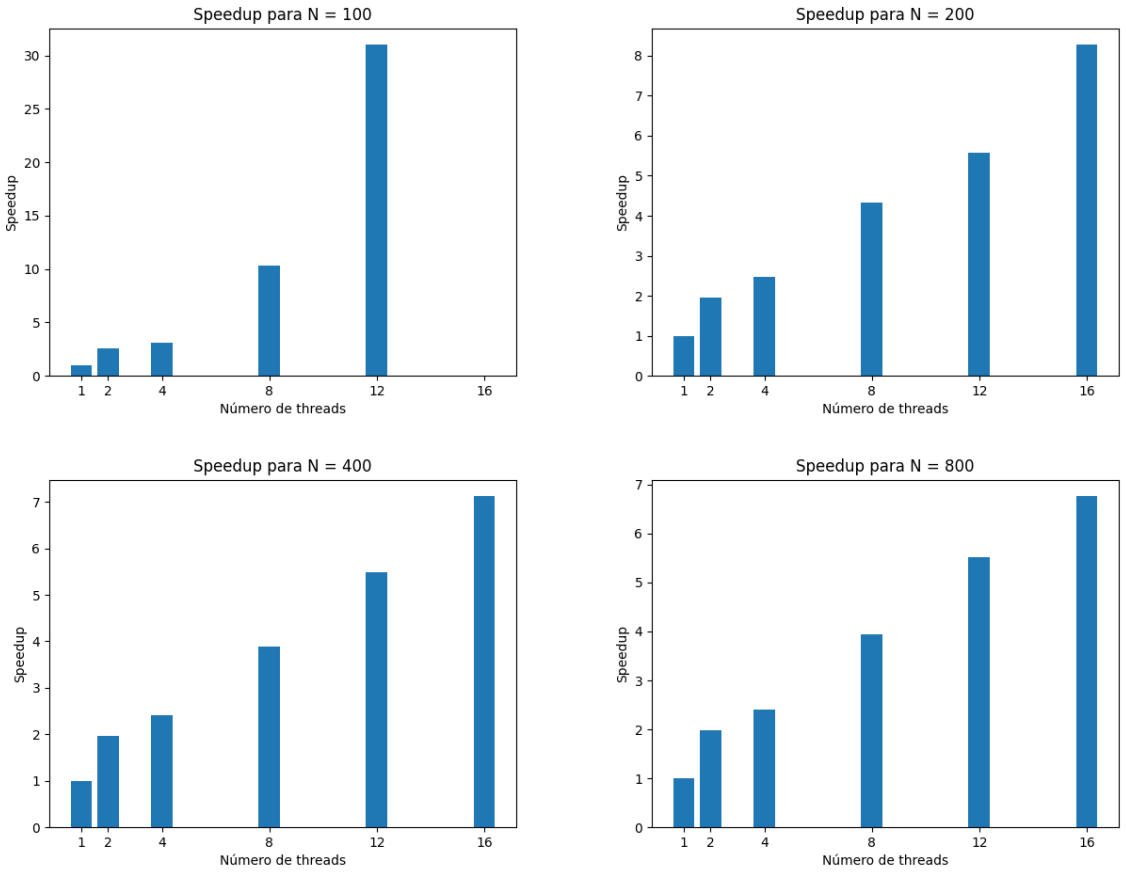
ANÁLISE DOS RESULTADOS OBTIDOS:
OBS: Quando o valores estão muitos pertos de 0. A precisão não é suficiente e assim ocorre um divisão por zero. Portanto, a barra de quando isso acontece não aparece:

ANÁLISE DE SPEED UP

Observando os resultados obtidos, fica evidente que a paralelização do programa demonstrou ser altamente eficaz. Desde os valores mais baixos de entrada (100) até os

maiores (800), o aumento no número de threads contribuiu significativamente para o desempenho do programa. Isso pode ser claramente visualizado nos gráficos de SpeedUp, que demonstram ganhos consideráveis.

A análise desses resultados sugere que o overhead de comunicação não impactou negativamente o desempenho do programa durante a paralelização. Esse aspecto é crucial para garantir que o ganho de desempenho observado seja efetivamente resultado da paralelização e não prejudicado por possíveis custos adicionais de comunicação entre threads. Isso o que foi falado pode ser observado nos gráficos abaixo:



ANÁLISE DA EFICIÊNCIA:

Ao examinar a eficiência do programa, nota-se um benefício claro ao utilizar um número maior de threads para entradas menores, como no caso de 100. O aumento na eficiência nessas situações sugere que a paralelização é eficaz e contribui para um desempenho superior quando o tamanho da entrada é mais modesto.

No entanto, à medida que o tamanho da entrada ultrapassa 100, observa-se uma diminuição na eficiência quando se utiliza um número maior de threads. Especificamente, a eficiência se estabiliza em um patamar um pouco superior à metade do que o programa sequencial oferece, a partir de 4 threads. Notavelmente, a eficiência com 4, 8, 12 e 16 threads permanece bastante similar para valores de entrada superiores a 100.

Essa observação pode ser justificada pelo equilíbrio delicado entre o ganho proporcionado pelo paralelismo e o aumento do overhead de comunicação entre as threads. A partir de um certo ponto, adicionar mais threads pode não resultar em ganhos significativos de eficiência devido a esse equilíbrio. A estabilidade na eficiência para 4, 8, 12 e 16 threads em entradas maiores que 100 sugere uma possível saturação, indicando que o benefício adicional de threads além desse número é limitado em relação ao custo adicional de coordenação entre threads.

Cabe ressaltar que, embora a eficiência mostre uma estabilização para 4, 8, 12 e 16 threads, o SpeedUp continua aumentando até pelo menos 16 threads, indicando que ainda há ganhos de desempenho com a adição de threads, apesar da eficiência se manter relativamente constante. Essa dinâmica pode ser atribuída a fatores específicos da aplicação e da arquitetura do hardware, destacando a complexidade na otimização de programas paralelos em situações práticas.

Isso o que foi falado pode ser observado nos gráficos abaixo: