UNIVERSIDADE FEDERAL DE LAVRAS
Departamento de Computação Aplicada

Atividade de análise de desempenho - REO 6

Código e Nome: GCC177 – Programação Paralela e Concorrente
Nome do Aluno: Mateus Carvalho Gonçalves

Ambiente de teste

Processador: *Intel® Core™ i7-7500U CPU @ 2.70GHz*
Número de núcleos de processamento: *2*
Número de threads: *4*

Resultados obtidos

Foram realizados testes com número de passos ‘n’ variando entre 10^2 e 10^9, como mostram as Tabelas 1 e 2. Para os diferentes valores de n, o código foi executado 10 vezes por instância para os números de threads T=1, T=2, T=4, T=8, e calculada a média do tempo de execução e o desvio padrão, mostrado nas Tabelas 1 e 2.

É importante ressaltar que para n=10^2, o resultado de pi obtido foi igual a 3,1416; já para n>=10^{10}, o resultado não era condizente com o número real de pi, configurando um erro; e para outros valores entre os previamente citados o resultado era igual a 3,14159.

Tabela 1. Média dos tempos de execução (formatação científica)

	Número de threads			
Número de Trapézios	***1***	***2***	***4***	***8***
n=10^2	5,91E-03	6,52E-05	2,19E-02	3,26E-04
n=10^4	4,08E-04	4,33E-04	2,21E-02	1,16E-03
n=10^6	3,00E-02	1,61E-02	3,01E-02	1,23E-02
n=10^8	7,11E-01	4,03E-01	2,68E-01	3,28E-01
n=10^9	6,97E+00	3,55E+00	2,16E+00	2,12E+00

Tabela 2. Desvio padrão dos tempos de execução (formatação científica)

	Número de threads			
Número de Trapézios	***1***	***2***	***4***	***8***
$n=10^2$	3,29E-03	4,37E-05	5,52E-03	2,99E-04
$n=10^4$	3,64E-05	4,89E-05	1,03E-02	3,40E-04
$n=10^6$	2,92E-03	3,41E-03	4,44E-03	4,42E-03
$n=10^8$	1,57E-02	4,31E-02	4,86E-02	5,24E-02
$n=10^9$	3,89E-02	3,33E-02	1,07E-01	1,27E-01

Pode-se perceber que ao aumentar o tamanho de n a média do tempo de execução aumentou porque essa variável define o número de repetições dentro de o loop para calcular o valor aproximado de pi. Já ao aumentar o número de threads obteve-se um comportamento não linear, para todos os números de passos houve uma quantidade de threads que executou em menor tempo. Isso quer dizer que o tempo gasto com a comunicação entre threads, a medida em que se aumenta as variáveis de teste, também não é linear na máquina utilizada. Outras condições externas ao teste também podem ter influenciado no resultado, como a execução de outros programas.

Em seguida foram calculados o speedup (gráfico da Figura 1), a eficiência (gráfico da Figura 2) e a fração sequencial definida experimentalmente (Métrica de Karp-Flat), mostrada na Tabela 3.

Figura 1. Speedup x número de passos

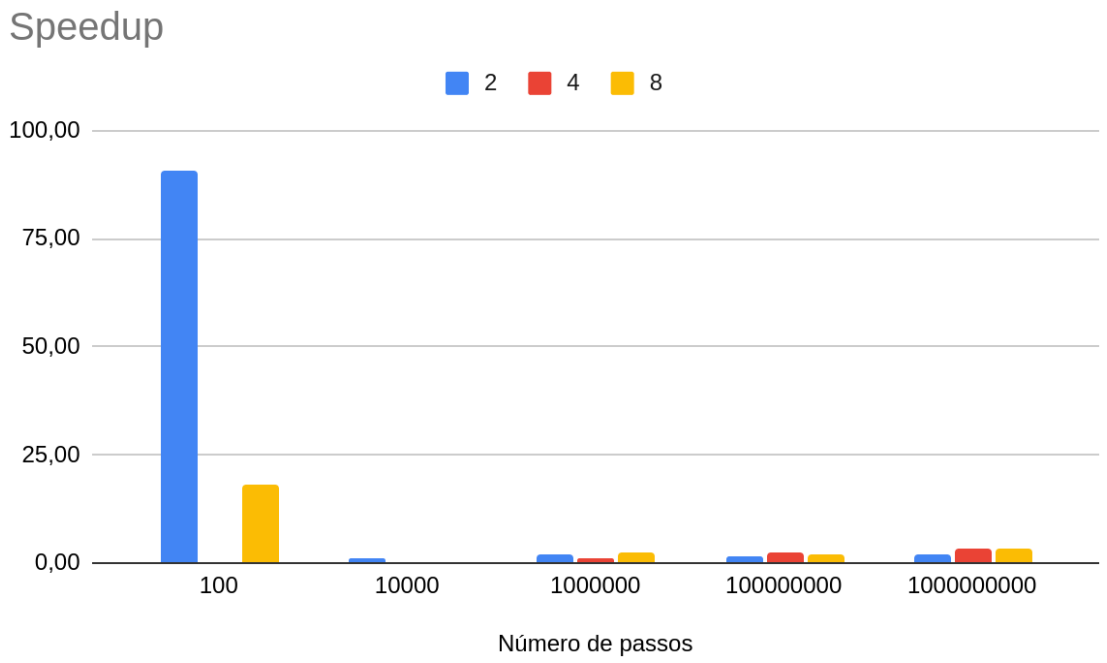

Figura 2. Eficiência das threads de acordo com o número de passos

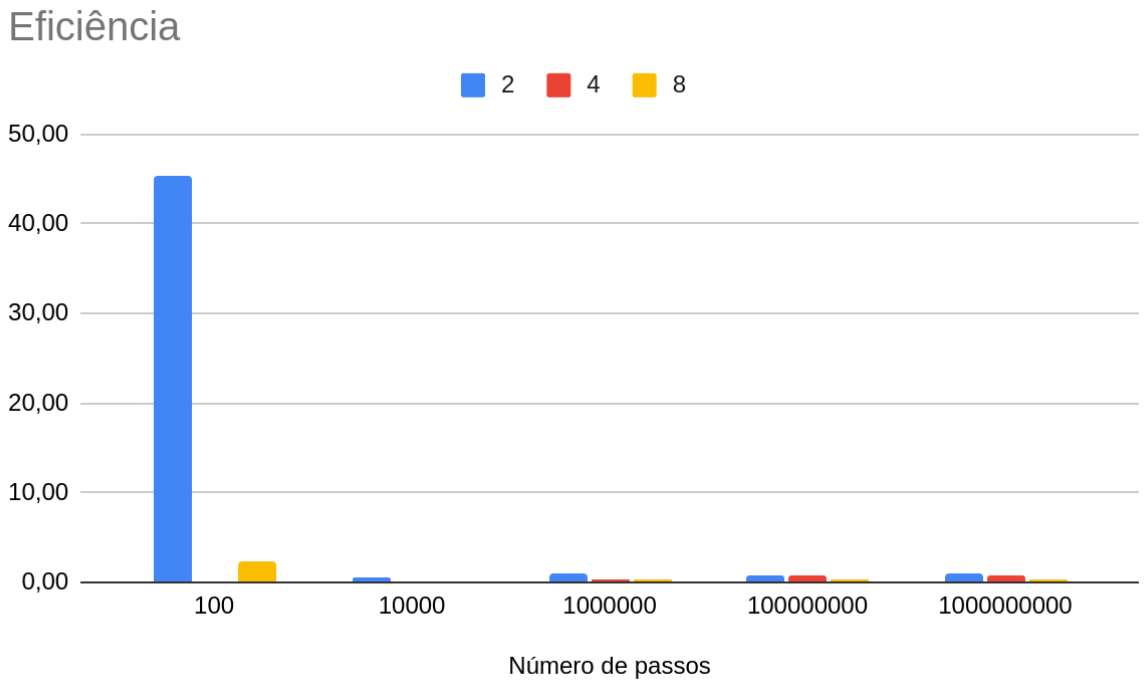

Tabela 3. Valores obtidos da métrica de Karp-Flat

	Número de threads		
Número de Trapézios	***2***	***4***	***8***
$n=10^2$	-0,98	4,61	-0,08
$n=10^4$	1,12	72,12	3,10
$n=10^6$	0,08	1,01	0,32
$n=10^8$	0,13	0,17	0,38
$n=10^9$	0,02	0,08	0,02

A partir do gráfico de Speedup percebe-se que a partir de 10000 passos, a medida em que se eleva o número de passos, há uma melhora considerável na execução, e quanto maior o número de passos o speedup de execuções com mais threads foi maior.

Tanto no gráfico de Speedup quanto no de Eficiência, ao executar o programa com 100 passos houve um comportamento “anômalo”, com 2 threads obtendo uma melhoria gigantesca, 4 threads piora a execução em relação a uma única thread, e 8 threads obtendo uma melhora grande também. É entendível que 2 threads tenham uma boa execução por causa de haver menos tempo gasto com comunicação, gerando um overhead paralelo muito menor, mas o comportamento de 4 e 8 threads foi imprevisível ao meu ver.

Já o gráfico da Eficiência mostra que a execução com 2 threads obteve melhor desempenho das threads, isso provavelmente se deve ao fato de que a máquina possui 2

núcleos de processamento. Em contrapartida, 8 threads possuiu a menor eficiência porque o tempo ocioso, devido à espero de resultados de outras threads, foi grande.

Já a fração sequencial determinada experimentalmente não foi eficaz para n=100 e n=10000, uma vez que apresentou valores negativos e maiores do que 1. Para 4 threads, a fração paralela sempre apresentou valores mais altos de execução sequencial, enquanto para n=1000000 a o código pode sofrer melhorias para tentar diminuir o tempo de execução com mais threads.

Conclusão

Após a análise dos dados obtidos a partir da execução dos testes, pode-se concluir que a paralelização do algoritmo de cálculo do valor de Pi paralelo usando OpenMP garantiu melhor velocidade e eficiência para execução com 2 threads, já para outros números de threads, a velocidade pode até aumentar, mas a eficiência de computação deixa a desejar, sendo assim o recurso consumido por essas threads poderia ser melhor distribuído para outras funções na máquina.

Referências

Rocha Ricardo, Programação Paralela e Distribuída: Métricas de Desempenho. Disponível em: https://www.dcc.fc.up.pt/~ricroc/aulas/0708/ppd/apontamentos/metricas.pdf. Acesso em: 15/02/2021.