**Introdução**

Este trabalho tem como objetivo a implementação e coleta de dados de execução para análise de matrizes, tanto de forma sequencial como paralela.

O primeiro código, “singlethread.c”, implementa uma forma de multiplicação de matrizes simples, sem utilizar paralelismo. O paralelismo utilizado neste código serve somente para o preenchimento da matriz alocada dinamicamente, uma vez que os objetivos são obter o speedup da multiplicação de matrizes somente, não havendo necessidade de incluir o tempo do preenchimento.

O segundo código, “multithread.c”, implementa tanto o paralelismo no preenchimento das matrizes como na multiplicação das mesmas. O paralelismo escolhido foi o paralelismo de dados, pois assim cada thread poderá executar o mesmo código de multiplicação em partes diferentes do código.

Os dois códigos recebem parâmetros pela linha de comando, sendo o primeiro o tamanho da matriz e o segundo, o número de threads que serão utilizadas.

**Multiplicação de Matrizes em Paralelo**

Para a implementação do paralelismo, foi utilizado como base o código anteriormente feito “singlethread.c”, aplicando técnicas de paralelismo onde se fez necessário. A principal diferença entre os dois códigos se dá no comando for da linguagem C que percorre a matriz.

Enquanto na versão singlethread um único thread percorre todas as linhas e colunas da matriz efetuando a multiplicação, na versão paralelizada, cada thread executa linhas específicas da matriz de acordo com seu identificador passado na criação do thread com o comando pthread\_create. Ou seja, supondo que temos 10 threads criadas, a thread identificada como thread 0 irá executar as linhas 0, 10, 20, 30 e assim por diante, enquanto a thread 1 irá executar as linhas 1, 11, 21, 31 etc. da matriz. Portanto, cada thread começa os cálculos na linha de seu identificador e a variável de controle é incrementada de acordo com o número total de threads criadas.

**Resultados**

As configurações das máquinas utilizadas para obtenção dos dados se encontram na tabela 1.

|  |  |
| --- | --- |
| **Processador** | **Memória** |
| Ryzen 5 5600x | 16gb |
| Ryzen 3 5400U | 8 gb |
| I5 5200U | 8 gb |

**Tabela 1. Configurações utilizadas**

Os gráficos speed up foram obtidos utilizando matrizes quadradas de tamanhos 1000, 3000 e 5000. Os desempenhos dos códigos foram calculados somente após o preenchimento das matrizes, ou seja, apenas as multiplicações tiveram seus tempos calculados.

Figura 1. Gráfico speedup para matriz quadrada de tamanho 1000

Figura 2. Gráfico speedup para matriz quadrada de tamanho 5000

É possível notar com as figuras que, quanto maior o tamanho da matriz, maior o destaque do paralelismo, principalmente em CPUs com mais processadores lógicos e, com mais poder de processamento, como é o caso do Ryzen 5 5600x, se mantendo bem próximo a linha speedup ideal até um pouco mais que 4 threads de execução.

Figura 3. Gráfico de eficiência de cada processador para matriz quadrada de tamanho 1000

Figura 4. Gráfico de eficiência de cada processador para matriz quadrada de tamanho 5000

É bem claro o decaimento constante de eficiência, sendo de fácil entendimento a relação de que, quando a eficiência começa a cair, principalmente de forma significativa, o speedup do programa está aumentando lentamente ou começando a cair.

**Conclusão**

Com os resultados obtidos, foi possível alcançar um nível relativamente satisfatório em relação ao speedup devido a sua proximidade a linha ideal em estágios iniciais. No entanto, após todos os testes realizados, também é possível concluir que um maior número de threads não necessariamente implica em um maior desempenho. Em todos os gráficos de speedup, foi possível notar que, após atingir um número máximo de threads de determinado processador, o ganho de desempenho praticamente não existe mais e, às vezes até diminui. Isso ocorre porque, a partir do momento em que o processador atinge o máximo de threads disponíveis, ele começa a intercalar os threads e, a paralelização não faz mais tanto sentido.

Nota-se também que, independentemente do tamanho da matriz, a eficiência diminui à medida que mais processadores são usados. Isso pode ser devido ao fato de que, quanto mais processadores são utilizados, a complexidade da divisão de trabalho entre eles se torna muito alta, reduzindo então seu tempo dedicado a resolver o problema.

**Referências**

Pacheco, P. (2011). An Introduction to Parallel Programming. Morgan Kaufmann Publishers.

SS Wiki. (n.d.). Parallel efficiency - simple approach. Universidade de Magdeburg. Acesso em 15/07/2023, <https://wikis.ovgu.de/lss/doku.php?id=guide:parallel_efficiency>.