Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD1116 - COMPUTAÇÃO DE ALTO DESEMPENHO - T01 (2025.1)
Tarefa 4: Aplicações limitadas por memória ou CPU

Docente: SAMUEL XAVIER DE SOUZA
Discente: Iago Gabriel Nobre de Macedo (20220037927)
Link do repositório da atividade no Github: https://github.com/IagoGMacedo/IMD1116-Computacao-de-Alto-Desempenho/blob/main/exercicios/calculos_bound/Main.c

Implementação

Realizei a implementação dos dois programas, onde: possuo primeiro um laço que realizo o cálculo de operações complexas para o CPU Bound. O número de iterações dessa operação é determinado no começo do programa pela variável "operationNumbers". Após isso tenho um outro laço for que realiza a soma de diferentes marizes para o Memory bound. Ambos os laços são paralelizados com openmp. A quantidade de threads que o programa irá executar é determinado como argumento de execução do programa.

Resultados Obtidos

O número de operações que determinei para o teste é de **100000000,** assim podemos ter ideia da performance obtida. Variando o número de threads para ambos os programas obtive o seguinte resultado:

Número de threads	Tempo CPU Bound	Tempo Memory Bound
1	5.5057	0.8095
5	2.3600	0.7980
10	2.4579	1.4516
100	2.5165	0.8996

Considerações

A análise dos resultados revela comportamentos distintos entre os programas CPU-bound e memory-bound conforme o número de threads aumenta. No caso CPU-bound, observa-se uma melhoria significativa no tempo de execução ao passar de 1 para 5 threads (5.5057s para 2.36s), indicando que a paralelização eficientemente distribuiu a carga de cálculos intensivos entre os núcleos disponíveis. Contudo, ao aumentar para 10 e 100 threads, o tempo sofre ligeira degradação (2.4579s e 2.5165s), sugerindo que o overhead de gerenciamento de threads e a competição por recursos da CPU superam os ganhos de paralelismo. Isso ocorre porque, em aplicações compute-bound, threads adicionais além da capacidade física de núcleos geram trocas de contexto frequentes, reduzindo a eficiência do pipeline de execução.

Já para o programa memory-bound, o desempenho apresenta maior volatilidade. Com 1 thread, o tempo é 0.8095s, reduzindo marginalmente para 0.798s com 5 threads. Entretanto, ao atingir 10 threads, o tempo dispara para 1.4516s, indicando um gargalo crítico no acesso à memória. Nesse cenário, a concorrência excessiva por largura de banda da RAM limita os ganhos de paralelização, já que múltiplos threads disputam acesso simultâneo aos mesmos recursos de memória. Com 100 threads, há uma leve recuperação (0.8996s), possivelmente devido ao escalonamento do sistema operacional otimizar a alocação, mas ainda sem superar o desempenho de 5 threads. Isso ilustra que aplicações memory-bound são sensíveis à saturação do barramento de memória, onde mais threads não implicam em maior eficiência.

Ao analisar os resultados obidos podemos entender que, em aplicações memory-bound, a paralelização pode ser útil até um ponto em que a largura de banda da memória é aproveitada sem contenção excessiva. Porém, em aplicações compute-bound, a competição por ciclos de CPU e a falta de recursos físicos suficientes podem degradar o desempenho. Assim, o balanceamento entre o número de threads e a capacidade do hardware é crucial: programas intensivos em CPU beneficiam-se de threads alinhados aos núcleos físicos, enquanto programas dependentes de memória exigem moderação para evitar gargalos. Essa dualidade reforça a importância de adaptar a estratégia de paralelização ao tipo de operação dominante no código.