Relatório da Tarefa 04

Aplicações limitadas por memória ou CPU

20210093987 - Francisca Paula de Souza Braz
Data da entrega: 04/04/2025
Disciplina: DCA3703 - PROGRAMAÇÃO PARALELA - T01 (2025.1)

Introdução

Neste relatório, foram implementados dois programas paralelos em linguagem C utilizando OpenMP, um limitado por memória (*memory-bound*), com operações simples de soma entre vetores e outro limitado por processamento (*compute-bound*), com cálculos matemáticos intensivos. Ambos foram paralelizados com a diretiva #pragma omp parallel for, e o número de threads foi controlado via variável de ambiente OMP_NUM_THREADS.

O objetivo da análise é observar como a variação no número de threads afeta o desempenho de cada tipo de programa, identificando os pontos em que a execução melhora, estabiliza ou piora. Além disso, foi feita uma reflexão sobre o impacto do multithreading de hardware (como o Hyper-Threading) em diferentes tipos de carga de trabalho, destacando os benefícios em aplicações *memory-bound* e os possíveis prejuízos em aplicações *compute-bound* devido à competição por recursos da CPU.

Metodologia

Para esta análise, foram desenvolvidos dois programas em linguagem C utilizando a API OpenMP para paralelização com base em laços de repetição. Os programas seguem características distintas:

- **Programa memory bound:** realiza somas simples entre elementos de dois vetores, sendo limitado principalmente pelo acesso à memória.

- **Programa CPU bound**: realiza operações matemáticas como funções trigonométricas, que são mais custosas, e limitado pelo poder de processamento da CPU.

Ambos os códigos foram paralelizados com a diretiva #pragma omp parallel for, que permite dividir automaticamente a iteração do laço entre múltiplas threads.

Para controlar a quantidade de threads utilizadas durante a execução, foi empregada a variável de ambiente OMP_NUM_THREADS, configurada com os valores 1, 2, 4 e 8 por meio do comando: export OMP_NUM_THREADS=N,

A OMP_NUM_THREADS é uma **variável de ambiente** reconhecida pelo OpenMP. Ela informa à *runtime* do OpenMP quantas threads devem ser usadas nas regiões paralelas. O comando export é usado no terminal para tornar essa variável visível ao programa que será executado em seguida.

Além de variar o número de threads, foi considerada a execução com apenas **1 thread** (export OMP_NUM_THREADS=1) como referência de desempenho sequencial. Em vez de remover as diretivas OpenMP, o código permaneceu paralelizado com #pragma omp parallel for, também foi compilados os programas sem definição de threads. Todas as execuções foram cronometradas utilizando funções de medição de tempo em C, considerando apenas o tempo de execução da parte principal do código.

Resultados

Os testes foram realizados com 1, 2, 4 e 8 threads, tanto para o programa *memory-bound* quanto para o programa *compute-bound*, mantendo a paralelização com OpenMP em todos os casos. O tempo de execução foi medido em segundos. Abaixo, os resultados obtidos:

Número de Threads	**Tempo de Execução (s)**
1 (Sequencia)l	1.436711
2	1.369001
4	1.306822
8	1.462361
com diretiva e sem definição de threads	**1.334504**

Programa Memory-Bound Resultados

Número de Threads	Tempo de Execução (s)
1 Sequencial	0.941350
2	1.106454
4	1.013057
8	1.021461
com diretiva e sem definição de threads	**0.854680**

Programa Compute-Bound Resultados

Ao executar dois programas paralelizados com OpenMP e variar o número de threads usando export OMP_NUM_THREADS, foi possível observar diferentes comportamentos de desempenho:

Quando o desempenho melhora, estabiliza ou piora, por exemplo, no programa da memory-bound, o desempenho melhorou levemente com 4 threads, apresentando o melhor tempo entre as configurações paralelas.mas quando usamos mais de 4 threads, o desempenho piorou, devido à contenção por acesso à memória com apenas 1 ou 2 threads, o tempo foi pior que o sequencial, pois houve overhead da paralelização sem ganho efetivo.

Já no programa da CPU-bound o desempenho piorou em todas as versões paralelas, sendo o melhor tempo obtido na versão sequencial. A paralelização trouxe overhead e causou competição direta por recursos da CPU, o que prejudicou o desempenho.
Em **programas memory-bound**, o multithreading de hardware (como o Hyper-Threading): Pode **ajudar discretamente**, pois permite que outra thread continue executando enquanto uma espera dados da memória. No entanto, se houver **muitas threads**, ocorre **contenção de memória** e o ganho se perde.
Em **programas compute-bound**, o multithreading tende a **prejudicar o desempenho**, pois as threads **competem pelos mesmos recursos de processamento**, como unidades

de ponto flutuante e cache. Isso **satura a CPU**, não há paralelismo real e o overhead se torna maior que qualquer benefício.

Conclusão

A paralelização com OpenMP pode oferecer ganhos significativos de desempenho, mas seus efeitos variam conforme a natureza da aplicação.

Nos testes realizados nessa atividade, o programa *memory-bound* apresentou leve melhora ao utilizar até 4 threads, mas teve desempenho prejudicado com 8 threads devido à contenção no acesso à memória, ou seja podemos inferir que é um indicativo de que há um limite de paralelismo efetivo imposto pela latência da memória entre outros.

Por outro lado, o programa *compute-bound* não se beneficiou da paralelização. O uso de múltiplas threads resultou em pior desempenho, causado pelo **overhead** da criação e sincronização de threads, além da **competição por recursos da CPU**.

Além disso, observou-se que o uso de apenas 1 thread com código paralelizado, serve como boa referência para comparar os impactos do paralelismo sem a necessidade de reescrever o código.

E esses resultados demonstram que a escolha de paralelizar um programa deve considerar o tipo de limitação envolvida (memória ou CPU), a arquitetura da máquina e o equilíbrio entre custo e benefício da paralelização. Em alguns casos, menos threads resultam em melhor desempenho, o que reforça a importância do entendimento da natureza da aplicação.