UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
IDCA3703 - PROGRAMAÇÃO PARALELA

TARFA 4 - APLICAÇÕES LIMITADAS POR MEMÓRIA OU CPU
RELATÓRIO DE EXECUÇÃO

ERNANE FERREIRA ROCHA JUNIOR

NATAL/RN, 31 DE MARÇO DE 2025

SUMÁRIO

1. INTRODUÇÃO

A computação paralela tem se tornado uma ferramenta essencial para aproveitar ao máximo os recursos de hardware disponíveis nos sistemas modernos. Com o aumento do número de núcleos em processadores e o suporte ao multithreading, é possível dividir tarefas computacionais em múltiplas threads e, assim, reduzir significativamente o tempo de execução de programas intensivos.

Entretanto, nem todo programa se beneficia da paralelização da mesma forma. A natureza do problema — se é limitado por memória (memory-bound) ou por processamento (compute-bound) — influencia diretamente na escalabilidade e no desempenho obtido ao se utilizar múltiplas threads. Programas memory-bound sofrem mais com latência de acesso à memória e largura de banda limitada, enquanto programas compute-bound são fortemente afetados pela capacidade de cálculo dos núcleos e pela contenção de recursos compartilhados.

Este relatório investiga e compara o comportamento de dois programas paralelizados com OpenMP: um com operações em vetores (memory-bound) e outro com cálculos matemáticos pesados (compute-bound). O objetivo é analisar como o desempenho de cada tipo de aplicação varia com o aumento do número de threads e entender os limites e gargalos impostos pelo hardware.

2. METODOLOGIA

Para avaliar o impacto da paralelização em diferentes tipos de cargas de trabalho, foram implementados dois programas em linguagem C utilizando a biblioteca OpenMP:

- **Memory-bound**: realiza somas simples entre dois vetores e armazena o resultado em um terceiro vetor. O acesso à memória foi propositalmente dificultado utilizando um stride (salto de índices), reduzindo a localidade espacial e aumentando o número de faltas de cache.
- **Compute-bound**: executa uma série de operações matemáticas complexas (como seno, logaritmo, cosseno, exponencial e raiz quadrada) sobre um grande número de elementos, acumulando os resultados. Essas operações exigem alto uso de CPU e pouca dependência de acesso à memória.

Ambos os programas foram paralelizados com a diretiva #pragma omp parallel for, e o número de threads foi variado de 1 até 32, independentemente do número de núcleos físicos disponíveis. A captura de tempo foi realizada com a função gettimeofday, permitindo medir com precisão o tempo de execução de cada programa.

A execução foi feita em um ambiente com um processador Apple M2, que possui 8 núcleos físicos, mas suporta múltiplos threads via hardware multithreading. As execuções foram feitas sequencialmente, com cada programa sendo executado com o mesmo número de threads em cada rodada.

As saídas de tempo foram salvas em um arquivo CSV com três colunas: número de threads, tempo de execução do programa memory-bound e tempo de execução do programa compute-bound. Posteriormente, os dados foram analisados por meio de gráficos e observação direta dos tempos medidos, a fim de identificar padrões de melhoria, estabilização ou degradação no desempenho.

3. RESULTADOS

Com os experimentos realizados, foi possível observar claramente os impactos da paralelização nos dois tipos de programas: um com gargalo em acesso à memória (memory-bound) e outro limitado por cálculos intensivos (compute-bound). Os tempos de execução foram registrados variando o número de threads de 1 a 32, e a partir desses dados, foram calculadas as métricas de speedup e eficiência.

A primeira observação relevante diz respeito ao tempo de execução conforme o número de threads aumenta. Como ilustrado na Figura 1, ambos os programas apresentaram queda significativa no tempo à medida que mais threads foram utilizadas. O programa memory-bound teve uma redução acentuada até cerca de 8 threads, ponto a partir do qual o tempo começa a se estabilizar, refletindo o limite físico dos núcleos disponíveis no processador. O compute-bound, por sua vez, também teve uma melhora inicial, porém os ganhos se tornaram rapidamente marginais após 4 a 6 threads, sugerindo que os cálculos pesados começam a competir por recursos internos da CPU, como FPU e cache, limitando o desempenho.

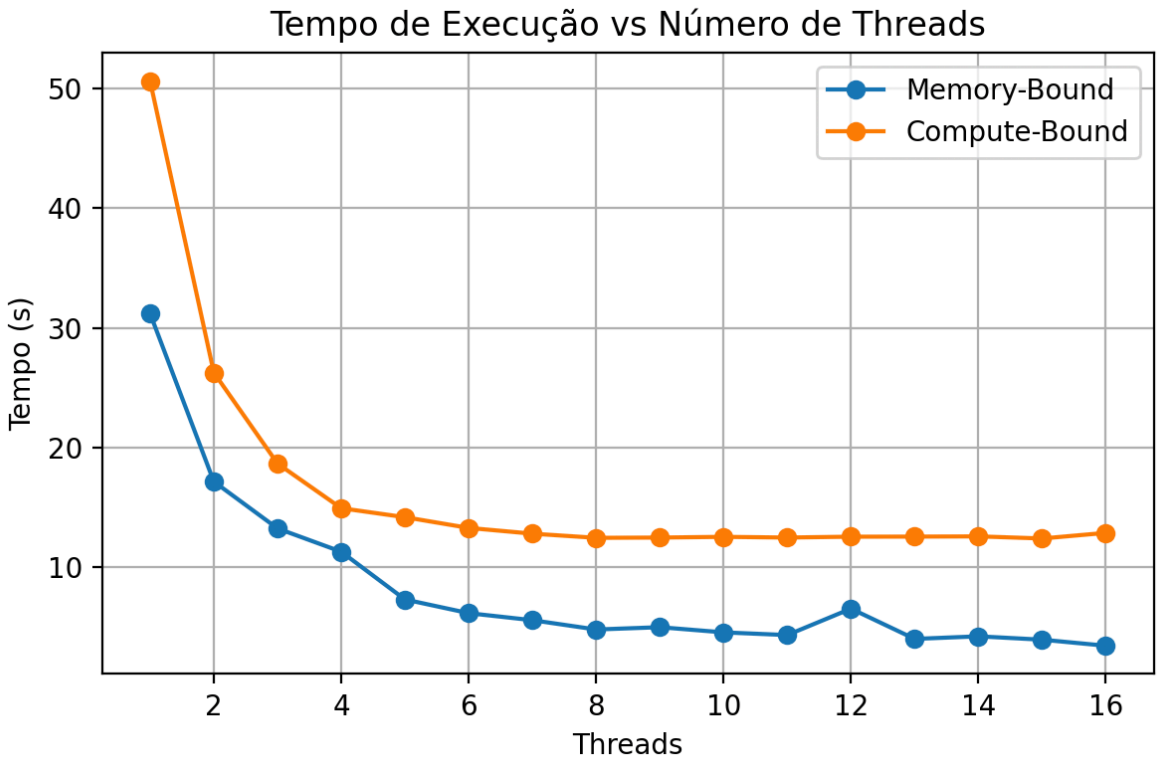

Figure 1: Tempo de Execução vs Número de Threads

A Figura 2 mostra o speedup obtido por cada programa. O programa limitado por memória chega próximo de um speedup linear nos primeiros 8 threads, o que é um indicativo de boa escalabilidade nesse intervalo. A partir desse ponto, embora continue melhorando levemente com mais threads, os ganhos são menores. Já o programa compute-bound apresenta uma curva de speedup que cresce bem nos primeiros núcleos, mas rapidamente se estabiliza, evidenciando que o aumento no número de

threads não se converte diretamente em ganhos proporcionais de desempenho, principalmente por causa da contenção de recursos internos do processador.

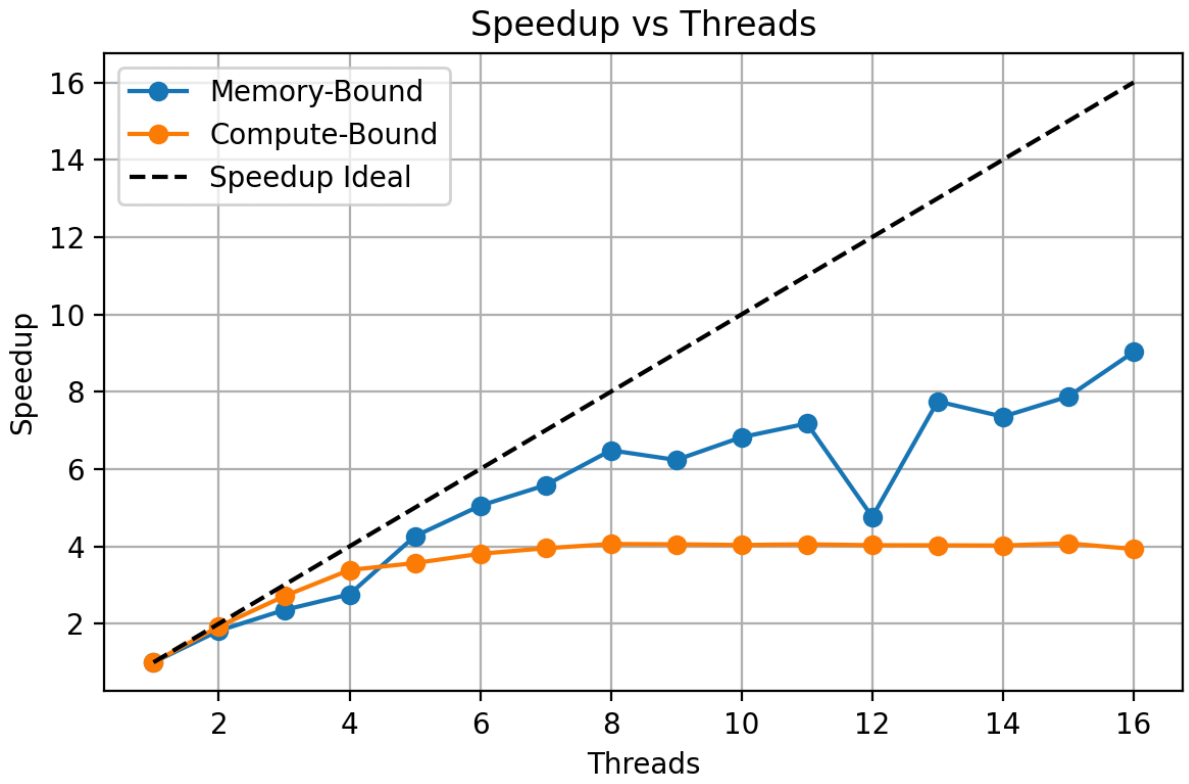

Figure 2: Speedup vs Threads

Esse comportamento é reforçado pela análise da eficiência, apresentada na Figura 3. Observa-se que o programa memory-bound mantém uma boa eficiência até 8 threads, caindo de forma progressiva depois disso, mas ainda obtendo algum benefício com o uso de mais threads — possivelmente graças ao suporte a hardware multithreading (como o SMT dos chips M1/M2). Em contrapartida, o programa compute-bound perde eficiência rapidamente, indicando que, após certo ponto, adicionar mais threads causa mais sobrecarga do que ganho real.

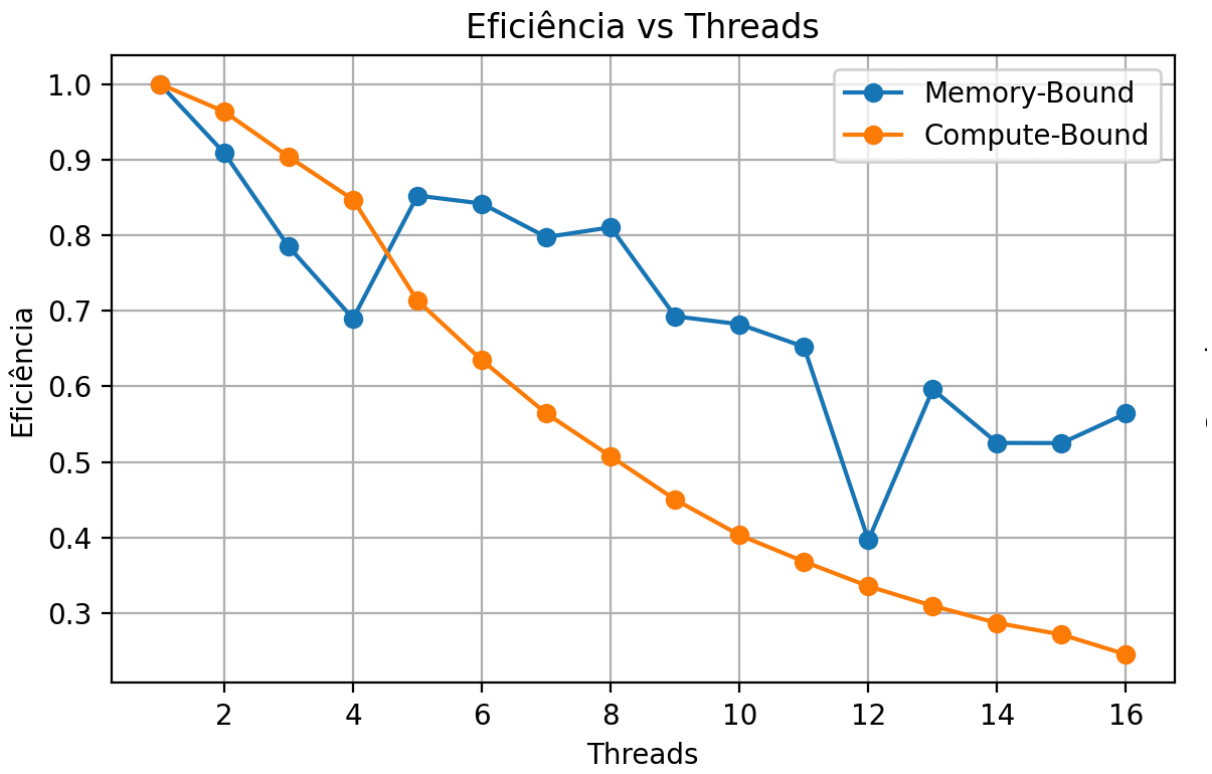

Figure 3: Eficiência vs Threads

Com base nesses resultados e nos gráficos analisados, é possível compreender com mais profundidade como diferentes tipos de aplicações respondem à paralelização, especialmente em relação ao número de threads disponíveis e à arquitetura subjacente do processador.

4. CONCLUSÃO

A partir da implementação e análise dos programas memory-bound e compute-bound, foi possível observar na prática os efeitos do paralelismo sobre diferentes tipos de carga computacional. O uso de OpenMP permitiu distribuir o trabalho entre múltiplas threads, revelando padrões distintos de escalabilidade e eficiência para cada caso.

No cenário memory-bound, a paralelização trouxe ganhos significativos até o limite de threads físicas disponíveis no processador, com alguma continuidade nos ganhos mesmo após esse ponto, graças ao suporte a multithreading por hardware. Isso indica que, quando o gargalo está no acesso à memória, o sistema pode se beneficiar de mais threads concorrentes, já que há espaço para sobrepor acessos e manter os núcleos ocupados.

Já no programa compute-bound, o ganho com múltiplas threads se mostrou limitado. Apesar de uma melhora inicial nos tempos de execução, rapidamente se observou uma saturação no desempenho. Isso ocorre porque cálculos matemáticos intensivos competem fortemente por recursos internos da CPU, como registradores, unidades de ponto flutuante e cache, tornando o paralelismo menos eficiente após um certo ponto.

Esses resultados evidenciam que o aumento no número de threads nem sempre se traduz em ganho proporcional de desempenho. Mais do que apenas "paralelizar", é fundamental compreender o tipo de tarefa em execução e como ela interage com a arquitetura da máquina. Assim, o uso eficiente de recursos depende tanto da natureza do problema quanto da forma como ele é escalonado em paralelo.

Essa análise prática ilustra a importância de medir e observar o comportamento real dos programas paralelizados, reforçando que a escolha do número de threads deve ser feita com base em testes e conhecimento da arquitetura, e não apenas em valores arbitrários.

5. ANEXOS

- Repositório no Github com o programa desenvolvido: https://github.com/ErnaneJ/parallel-programming-dca3703