Análise de Escalabilidade: CPU-Bound vs. Memory-Bound

Um Estudo Experimental com OpenMP

Werbert Arles de Souza Barradas

Universidade Federal do Rio Grande do Norte (UFRN)
Disciplina de Programação Paralela - DCA3703

20 de agosto de 2025

Introdução

O Gargalo do Desempenho Paralelo

O desempenho em arquiteturas multicore é frequentemente limitado por um de dois fatores: a capacidade de processamento (**CPU**) ou a velocidade de acesso à memória (**RAM**).

CPU-Bound

Aplicações cujo tempo é dominado por cálculos intensivos.

Memory-Bound

Aplicações cuja performance é limitada pela largura de banda e latência da memória.

Objetivo do Estudo

Demonstrar e analisar experimentalmente o comportamento de escalabilidade destes dois tipos de carga de trabalho ao variar o número de threads.

Metodologia: Programas de Teste

Programa 1: Memory-Bound

Projetado para saturar a memória combinando duas técnicas:

- **Acesso por Colunas:** Gera alta taxa de *cache misses* ao criar grandes "saltos" (strides) na memória.
- **Acesso Aleatório:** A ordem das colunas é embaralhada para derrotar o *hardware prefetcher* da CPU, maximizando a latência.

Metodologia: Programas de Teste

Programa 2: CPU-Bound

Projetado para saturar a CPU:

- **Dados na Cache:** O vetor de dados é pequeno o suficiente para caber na cache, eliminando a memória como gargalo.
- **Cálculo Intensivo:** Executa uma função com 100.000 iterações de operações matemáticas complexas (‘sin‘, ‘cos‘, ‘sqrt‘) por elemento.

Metodologia: Estrutura do Código

Memory-Bound: Acesso Ineficiente

```
// Loop para testar com diferentes números de threads
for (int num_threads = 1; num_threads <= 16; num_threads *= 2) {
        omp_set_num_threads(num_threads);
        double start_time = omp_get_wtime();
        // O loop externo percorre o VETOR DE ÍNDICES DE
        // COLUNAS embaralhado.
        #pragma omp parallel for
        for (int k = 0; k < DIM; k++) {
            int j = col_indices[k]; // Pega uma coluna aleatória
            // O loop interno ainda percorre as linhas,
            // causando o acesso column-major.
            for (int i = 0; i < DIM; i++) {
                c[IDX(i, j)] = a[IDX(i, j)] + b[IDX(i, j)];
            }
        }
        double end_time = omp_get_wtime();
    }
```

Metodologia: Estrutura do Código

CPU-Bound: Cálculo Intensivo

```
// Funcao com calculos pesados.
double intensive_calculation(double val) {
    double result = val;
    for (int i = 0; i < 100000; i++) {
        result = sin(result)*cos(val) + sqrt(fabs(result));
    }
    return result;
}
// Loop para testar com diferentes números de threads
    for (int num_threads = 1; num_threads <= 16; num_threads *= 2) {
        omp_set_num_threads(num_threads);
        double start_time = omp_get_wtime();
        // Seção paralela. O gargalo aqui é a capacidade de processamento d
        #pragma omp parallel for
        for (long i = 0; i < VECTOR_SIZE; i++) {
            data[i] = intensive_calculation(data[i]);
        }
        double end_time = omp_get_wtime();
    }
```

Comparativo de Desempenho: Memory-Bound vs. CPU-Bound

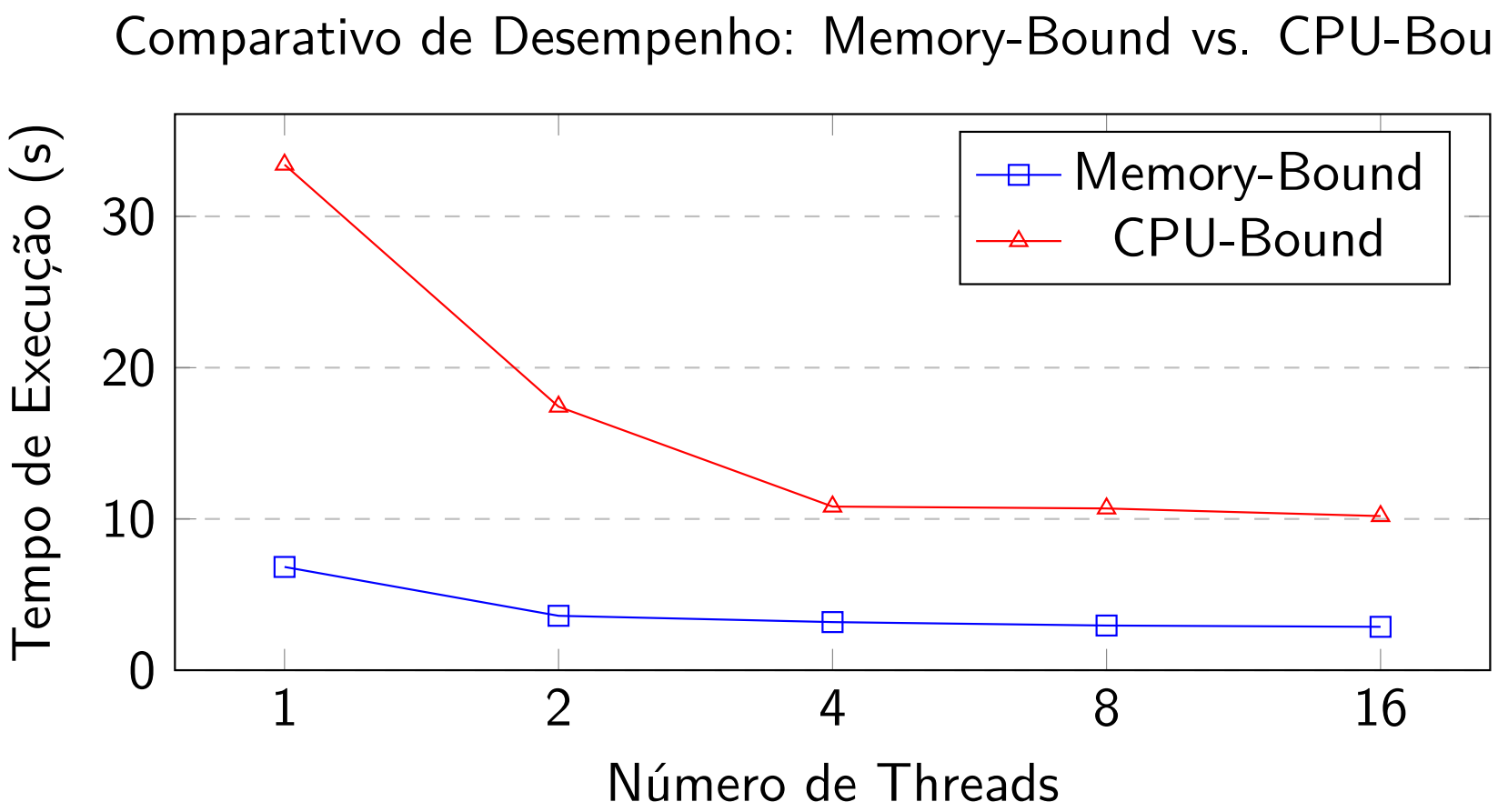

Figure: Gráfico comparativo do tempo de execução pelo número de threads.

Análise dos Resultados

CPU-Bound (Linha Vermelha)

O gráfico mostra uma queda acentuada no tempo de execução até 8 threads, indicando um excelente *speedup* com o aumento de núcleos. A partir de 8 threads, a melhoria se torna mínima e a curva se achata, o que é consistente com a saturação dos núcleos físicos e a contenção de recursos do SMT (Hyper-Threading).

Memory-Bound (Linha Azul)

A curva de melhoria é muito menos acentuada. Embora haja um ganho inicial significativo (de 1 para 2 threads), o desempenho estabiliza rapidamente. O gargalo deixa de ser a CPU e passa a ser a largura de banda da memória, que é saturada com poucas threads.

O Papel Duplo do Multithreading de Hardware

"Atrapalha" em Cargas CPU-Bound

- Duas threads lógicas competem pelos mesmos recursos de cálculo (ALUs, FPUs) de um único núcleo físico.
- Como os recursos já estão 100
- **Efeito Prático:** A performance do programa CPU-Bound estagna quando o número de threads ultrapassa o de núcleos físicos.

"Ajuda" em Cargas Memory-Bound

- Uma thread passa a maior parte do tempo parada (*stalled*), aguardando dados da RAM, o que deixa o núcleo ocioso.
- O SMT aproveita este tempo ocioso para executar a segunda thread no mesmo núcleo.
- Este processo **"esconde" a latência da memória**, melhorando a utilização geral do processador.
- **Efeito Prático:** Pequenos ganhos de performance

Conclusão

Perfis de Escalabilidade Distintos

O estudo demonstrou experimentalmente os dois perfis de escalabilidade.

- A performance do programa **CPU-Bound** é limitada pelo **número de núcleos de processamento físicos** disponíveis. Ele escala bem até atingir este limite, mas ganha pouco com SMT.
- A performance do programa **Memory-Bound** é limitada pela **largura de banda da memória**. Atinge um platô de desempenho muito antes, quando o barramento de memória é saturado.

Implicação Prática

Compreender se uma aplicação é CPU-bound ou Memory-bound é crucial para otimizar o desempenho. Para programas CPU-bound, o foco deve ser em algoritmos mais eficientes e paralelismo de tarefas. Para programas Memory-bound, a otimização deve focar em melhorar a localidade de dados e reduzir o tráfego de memória.