Tarefa 12: Análise de Escalabilidade e Otimização Incremental

Aplicando os Modelos de Amdahl e Gustafson

Werbert Arles de Souza Barradas

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

03 de outubro de 2025

Agenda

1. Teoria da Escalabilidade
2. Evolução das Implementações
3. Análise dos Heatmaps

Teoria da Escalabilidade: Lei de Amdahl

A Pergunta: Qual o speedup máximo para um problema de tamanho fixo?

A Lei de Amdahl foca em acelerar uma tarefa existente (Escalabilidade Forte).

A Ideia Central:

- O ganho de desempenho é **limitado pela porção serial** do código.
- Não importa quantos processadores você adicione, o tempo total nunca será menor que o tempo da parte sequencial.

A Fórmula:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

- $S(N)$: Speedup com N processadores
- P: Fração paralelizável do código
- N: Número de processadores

Conclusão de Amdahl

O speedup tem um limite máximo rígido, definido pela parte serial.

Teoria da Escalabilidade: Lei de Gustafson

A Pergunta: E se aumentarmos o problema junto com os processadores?

A Lei de Gustafson foca em resolver problemas **maiores** no mesmo tempo (Escalabilidade Fraca).

A Ideia Central:

- Para problemas grandes, a porção serial **se torna insignificante**.
- Com mais poder computacional, os cientistas aumentam a precisão ou o tamanho da simulação, não apenas rodam o problema antigo mais rápido.

A Fórmula:

$$S(N) = N - P \times (N - 1)$$

- $S(N)$: Speedup com N processadores
- P: Fração serial do código
- N: Número de processadores

Conclusão de Gustafson

O speedup pode escalar de forma quase linear com o número de processadores.

1. Versão Ingênua

Estratégia e Gargalo

Aplicar `#pragma omp parallel for/sections` em cada laço de trabalho de forma independente. **Gargalo:** Alto overhead de criar e destruir threads (fork/join) a cada etapa.

```
// Loop de tempo PRINCIPAL - SERIAL
for (int t = 0; t < NT; t++) {
    // 1. Cria e destr i threads aqui
    #pragma omp parallel for
    for (int i = 1; i < NX-1; i++) { /*...*/ }

    // 2. Cria e destr i threads novamente aqui
    #pragma omp parallel sections
    { /* Condicoes de contorno */ }
}
```

2. Otimização Final (com `collapse`)

Estratégia Final

Envolver todo o laço de tempo em uma região paralela única para eliminar o overhead de fork/join, e usar `collapse(2)` para otimizar o balanceamento de carga do kernel.

```
// Threads sao criadas apenas UMA VEZ aqui
#pragma omp parallel
{
    // Loop de tempo PRINCIPAL - PARALELO
    for (int t = 0; t < NT; t++) {
        // Distribuicao de trabalho com baixo custo
        #pragma omp for collapse(2) schedule(static)
        for (int i = 1; i < NX-1; i++) {
            for (int j = 1; j < NY-1; j++) { /*...*/ }
        }
        // ... demais diretivas work-sharing
    }
} // Threads sao destruidas apenas UMA VEZ aqui
```

Configuração do PaScal Analyzer

Uso do NPAD

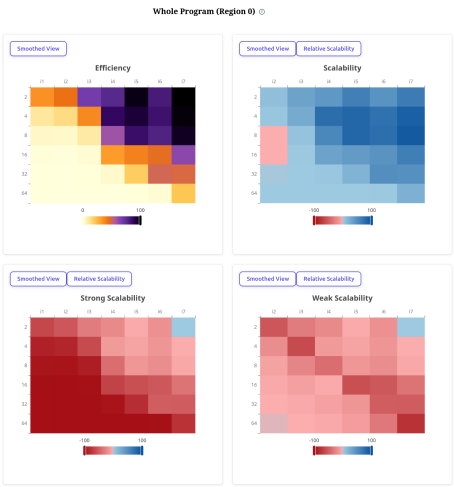
Foi configurado 1 nó (Amd 512) com 64 cores

Flags do Pascalanalyzer:

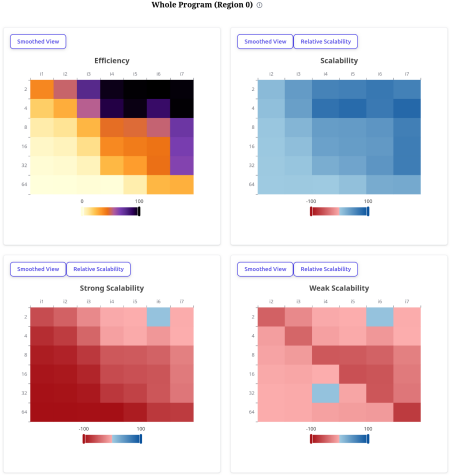
- -c 1,2,4,8,16,32,64
- -i 32,64,128,512,1024,2048,4096
- –inst aut -g -o OUTPUT.json

Análise: Comparativo Visual e Configurações

Versão Ingênua



Versão Otimizada



Análise: Diagnóstico Comparativo

Diagnóstico da Versão Ingênua

- **Causa:** O overhead massivo de *fork-join* a cada iteração do laço de tempo.
- **Efeito:** Eficiência quase nula e **escalabilidade forte negativa** (vermelho escuro), resultando em *slowdown*. O programa fica mais lento com mais processadores.

Diagnóstico da Versão Otimizada

- **Causa:** O overhead foi eliminado com uma região paralela única.
- **Efeito:** O *slowdown* desaparece. A escalabilidade forte, embora não seja perfeita (vermelho claro), mostra um comportamento saudável de **retornos decrescentes**, como previsto pela Lei de Amdahl.

Conclusões Finais

- A estratégia de maior impacto foi a **redução de overhead**, consolidando o trabalho dentro de uma **região paralela única**. Isso corrigiu a escalabilidade negativa.
- A otimização é um processo incremental: a remoção do gargalo do **modelo de programação** (fork-join) revelou os gargalos inerentes ao **algoritmo e hardware** (comunicação, acesso à memória).
- A cláusula `collapse(2)` foi importante para maximizar a eficiência do kernel, melhorando o balanceamento de carga para a computação na grade 2D.
- A **Lei de Amdahl** foi demonstrada na prática: após a otimização, o desempenho geral continuou limitado pela porção do código que não escalava perfeitamente.