Relatório - Análise de Desempenho de Programas em C - João Comini César de Andrade

Atividade 01 - Análise de Desempenho

Conceitos

Localidade de memória e cache

Os programas executam de forma mais eficiente quando acessam dados próximos na memória, aproveitando os níveis de cache do processador. Isso se chama localidade espacial e temporal. Acessos sequenciais (STRIDE=1) são mais eficientes, enquanto acessos com saltos grandes (ex.: STRIDE=16) causam mais cache misses.

Predição de desvio

Instruções condicionais (if) criam pontos de decisão. Processadores modernos usam predição de desvio para tentar adivinhar o resultado e manter o pipeline cheio. Se errar, há perda de desempenho. Versões sem if podem ser mais rápidas.

Ponto flutuante vs. inteiros

Operações em ponto flutuante exigem unidades específicas (FPU) e geralmente são mais custosas que operações inteiras. Apesar de CPUs modernas possuírem boas FPUs, em alguns casos substituir float por int pode trazer ganhos.

Otimizações de compilador (-O0 vs -O2)

O nível -O0 gera código direto e pouco otimizado, enquanto -O2 aplica técnicas como eliminação de redundâncias, unrolling e vectorization. Isso reduz bastante o tempo de execução, embora torne a depuração mais difícil.

Questões sobre o código

a) Onde ocorre o uso de ponto flutuante (float) e de inteiros (int)?

- O código utiliza float quando está no modo MODE=0 ou MODE=1, pois as matrizes e operações matemáticas (multiplicação, somas) são feitas com números de ponto flutuante.
- Já em MODE=2, o programa substitui as operações em ponto flutuante por int, realizando cálculos apenas com inteiros.

b) Onde aparece o desvio condicional (if) e como ele pode afetar o desempenho?

- O desvio aparece no laço principal, dentro do cálculo (modo MODE=0). O if verifica uma condição para decidir qual operação aplicar ao elemento da matriz.
- Esse desvio pode prejudicar a predição de desvio da CPU, especialmente se os resultados da condição forem imprevisíveis. Quando a predição falha, a CPU perde ciclos e precisa reexecutar instruções, aumentando o tempo total de execução.

c) Como o parâmetro STRIDE altera o padrão de acesso à memória?

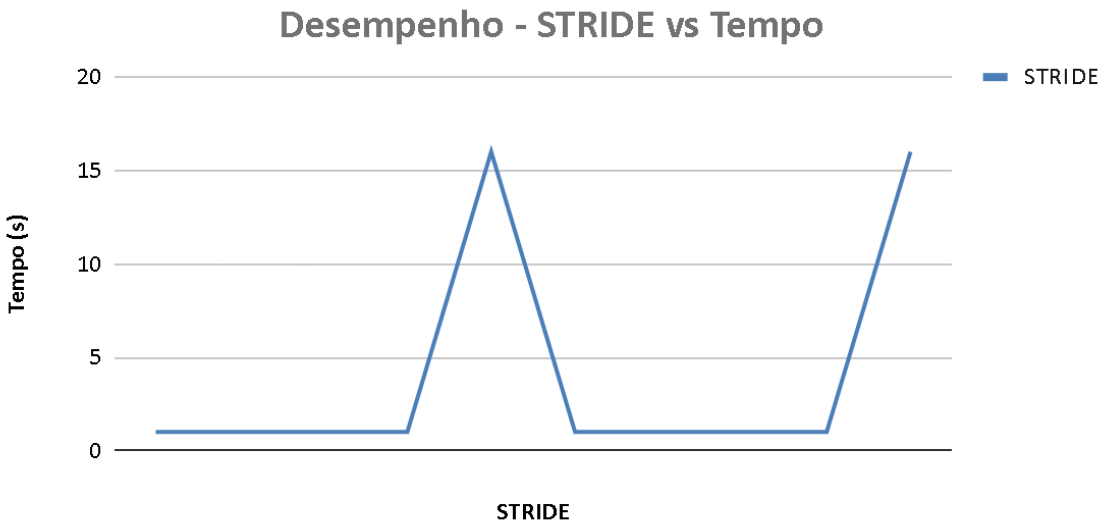
- O parâmetro STRIDE define o passo de acesso aos elementos da matriz.
- Quando STRIDE=1, o acesso é sequencial (boa localidade espacial de cache → mais rápido).
- Quando STRIDE é maior (ex.: 16 ou 32), os acessos ficam salteados, o que gera mais cache misses, reduzindo o aproveitamento da hierarquia de memória e aumentando o tempo de execução.

Coleta de Dados

Tabela_1

N	REPEAT	STRIDE	MODE	Flag (-O0/-O2)	Tempo (s)
4096	3	1	0	-O0	0.68
4096	3	1	0	-O0	0.35
4096	3	1	1	-O0	0.80
4096	3	1	2	-O0	0.48
8192	6	16	1	-O0	0.54
4096	3	1	0	-O2	0.36
4096	3	1	0	-O2	0.18
4096	3	1	1	-O2	0.44
4096	3	1	2	-O2	0.18
8192	6	16	1	-O2	0.47

Gráfico STRIDE vs TEMPO


Desempenho - STRIDE vs Tempo
20
15
10
5
0
Tempo (s)
STRIDE
STRIDE

Atividade 02 - Escalabilidade e Programação Paralela (OpenMP)

Conceitos

Processos e threads

Um processo é uma instância independente de um programa em execução, com seu próprio espaço de endereçamento. Uma thread é uma unidade menor de execução dentro de um processo, que compartilha a memória e os recursos do processo.

Speedup

Mede o ganho de desempenho da versão paralela em relação à sequencial. Se T1 é o tempo com 1 thread e Tp com p threads, Speedup = T1/Tp.

Eficiência

É o speedup dividido pelo número de threads (Eficiência = Speedup/p). Mede quão bem os recursos paralelos estão sendo usados.

Questões sobre o código

a) O que faz a diretiva #pragma omp parallel for ?

- Essa diretiva instrui o compilador a dividir automaticamente as iterações de um laço for entre várias threads.
- Cada thread executa uma parte do laço em paralelo, explorando o paralelismo do hardware (multicore).

b) Qual o comando no código que modifica o número de threads?

- Normalmente, o código usa a função omp_set_num_threads(n) para definir quantas threads o programa vai usar.
- Em alguns casos, o número de threads também pode ser controlado por variáveis de ambiente (OMP_NUM_THREADS).

c) Quantos processadores/núcleos possui o computador em uso e qual seria o número ideal de threads?

- Isso depende do seu computador: em Linux/macOS pode ser visto com lscpu ou nproc, no Windows pelo Gerenciador de Tarefas → Desempenho → CPU.
- O número ideal de threads geralmente é igual ao número de núcleos físicos ou lógicos (com hyper-threading).

    - Exemplo: se sua CPU tem 8 núcleos/16 threads, o ideal é usar até 16 threads.

    - Se usar muito mais do que os núcleos disponíveis, o sistema pode sofrer overhead de escalonamento, piorando o desempenho.

Coleta de Dados

Tabela_2

Threads	N	t_best	t_avg	Checksum	Speedup	Eficiência
1	1048576	0.7	523773	1	1	
2	1048576	0.3	523773	2.025	1.012	
4	1048576	0.17	523773	3.9	0.98	
8	1048576	0.09	523773	7.5	0.93	
1	524288	0.34	261886	1	1	
2	1048576	0.35	1047546	0.98	0.5	
4	2097152	0.36	1047546	0.95	0.24	
8	4194304	0.37	2095093	0.92	0.12	

Eficiência não apareceu no terminal:

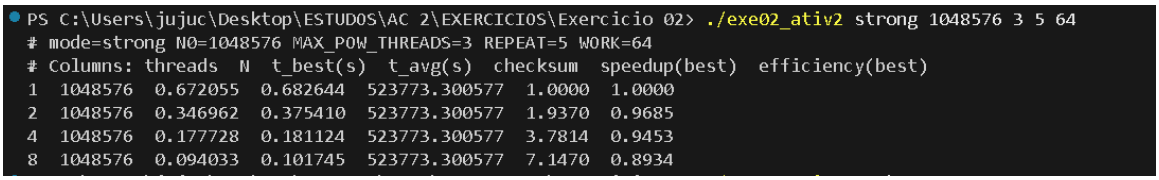
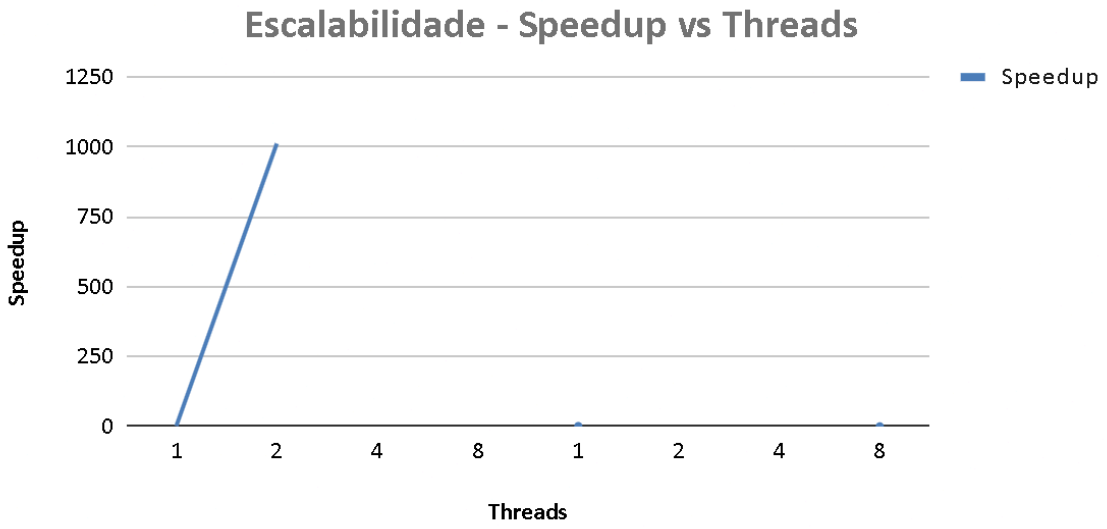


Gráfico Speedup vs Threads


Escalabilidade - Speedup vs Threads
Speedup
1250
1000
750
500
250
0
1
2
4
8
1
2
4
8
Threads
Speedup