OTIMIZAÇÕES

As otimizações de código são cruciais na Engenharia de Performance e visam contornar ou amenizar o impacto do **gargalo** (o *bottleneck*) que limita a velocidade de execução do programa. Em arquiteturas modernas, esse gargalo é frequentemente o **acesso à memória** (*von Neumann bottleneck*), e não a velocidade da Unidade Lógica e Aritmética (ULA).

A performance de um código é influenciada pela arquitetura do CPU (clock, número de *cores*, tamanho da *cache*) e pelo sistema de memória (interface, velocidade). O código pode ser limitado pela memória (*memory-bound*), limitado pela cache (*cache-bound*), ou limitado pelas operações aritméticas (*CPU-bound*). O objetivo é transformar um código *memory-bound* em *CPU-bound*.

Resumo das Otimizações de Código

1. Otimização Algorítmica e Redução de Custo

Esta otimização foca em reduzir a complexidade assintótica do algoritmo e a quantidade de operações elementares.

Otimização	Lógica
Ordem de Operações	Escolher a ordem que resulta em menor complexidade. Exemplo: MVM (Multiplicação Matriz-Vetor) é $O(N^2)$, enquanto MMM (Multiplicação Matriz-Matriz) é $O(N^3)$.
Fusão de Laços (*Loop Fusion*)	Combinação de laços adjacentes para aumentar a localidade e o reuso de dados, reduzindo a sobrecarga do laço (*loop overhead*).
Eliminação de Subexpressões	Mover cálculos constantes para fora de laços longos para que sejam executados apenas uma vez, economizando FLOPS.

2. Otimização da Hierarquia de Memória (Cache)

O foco é maximizar a **localidade espacial** e **temporal** dos dados, garantindo que os dados sejam acessados de forma contígua (*row major order* em C/C++).

Otimização	Lógica

Acesso Contínuo / Ordem de Laço	Inverter a ordem dos laços para que o índice mais interno acesse elementos adjacentes na memória, aproveitando a linha de cache (*cache line*). O acesso em coluna (*strided*) é ineficiente em C/C++.
Estrutura de Dados (SoA)	Preferir *Struct of Arrays* (SoA) em vez de *Array of Structs* (AoS) para armazenar dados quando o acesso subsequente favorece a leitura sequencial de uma coordenada por vez. Isso aumenta o **aproveitamento da linha de cache** no laço mais interno.
Redução do Volume de Dados	Reduzir o tamanho das estruturas (*structs*) se nem todos os membros são usados, diminuindo o volume de dados carregados na linha de cache, o que melhora a performance.
Blocagem de Laço (*Loop Blocking*)	Dividir matrizes grandes em blocos menores que cabem nas caches L2/L3. Isso aumenta o reuso de dados, reduzindo o tráfego de memória e *TLB misses*.

3. Otimização de Processamento (Pipeline, SIMD e Registradores)

O objetivo é manter o *pipeline* do processador cheio e utilizar recursos de paralelismo de dados (SIMD).

Otimização	Lógica
Eliminação de Desvios (*Branch Prediction*)	Mover laços condicionais (`if/else`) para fora de laços longos. Desvios repetidos impedem o preenchimento do *pipeline* e causam *branch mispredictions* caros.
Desenrolamento de Laço (*Loop Unrolling*)	Replicar o corpo do laço para expor mais **instruções independentes** ao compilador e ao *pipeline*, reduzindo a sobrecarga do laço e facilitando a vetorização (SIMD).
Unroll & Jam	Combina *unroll* com fusão de laços para otimizar operações como MVM. O vetor (ou dado) carregado no laço interno é reusado por múltiplas iterações do laço externo, reduzindo o tráfego de memória.

SIMD / Vetorização	Uso de instruções *Single Instruction Multiple Data* (SIMD, como AVX) para executar a mesma operação em múltiplos dados em paralelo, exigindo **dados independentes** e acesso contínuo/alinhado.
Pressão de Registradores	Um *unroll* excessivo ou a presença de muitos vetores em um bloco de código pode levar ao *Register Spill*, forçando o processador a escrever variáveis na memória (cache), o que prejudica a performance.

4. Otimização de Funções Matemáticas

Esta otimização trata de substituir operações de alto custo computacional.

Otimização	Lógica
Tabela de Lookup	Substituir o cálculo repetitivo de funções caras como `sin()`, `cos()`, `pow()`, ou `tanh()` por uma tabela de consulta pré-calculada, quando o domínio de entrada é pequeno. Se a tabela couber na cache, o ganho é significativo.
Substituição Aritmética	Substituir funções complexas (como `pow()`) por operações aritméticas mais simples (multiplicações) para reduzir o número de FLOPS e o custo de chamadas de função.