Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Desenvolvimento de Código Otimizado 2023.2

Atividade 4: Vetorização

Gabriel da Cunha Dertoni	11795717
Matheus Ventura de Sousa	11345541
Pedro Lucas de Moliner de Castro	11795784
Vitor Caetano Brustolin	11795589

Profa. Dr. Sarita Mazzini Bruschi

São Carlos, Dezembro de 2023

1 Introdução

O objetivo da presente atividade é aprender sobre vetorização do código, seja ela feita de forma manual ou automática pelo compilador. Para isso, foi desenvolvido um código que executa múltiplos cálculos em raios de luz coloridos a cada frame para mostrar uma figura na tela.

2 Ambiente de execução

O experimento foi compilado usando `msvc` e executado em uma máquina com as especificações descritas na tabela a seguir:

Nome do modelo	Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
Arquitetura	x86_64
Modos operacionais da CPU	32-bit, 64-bit
Ordem dos bytes	Little Endian
CPUs	6
Threads por núcleo	1
Núcleos por soquete	6
Soquetes	1
Núcleos de NUMA	1
Frequência máxima	4100 MHz
Frequência mínima	800 MHz
Cache de L1d	192 KiB (6 instâncias)
Cache de L1i	192 KiB (6 instâncias)
Cache de L2	1.5 MiB (6 instâncias)
Cache de L3	9 MiB (1 instância)

3 Códigos e execução

O código que importa para essa análise está nos arquivos `src/raycast.c`, que apresenta a versão implementada tradicionalmente, e `src/raycast_simd.c` que faz os mesmos cálculos, porém usando vetorização manual no código. As instruções para compilar e executar o código, no Windows ou no Linux, com auxílio dos scripts desenvolvidos podem ser encontradas no arquivo `README.md`. Com a aplicação em execução, segure a barra de espaço para alterar para a renderização usando instruções SIMD. O tempo de cada frame será impresso no console.

3.1 Vetorização manual

O grupo identificou duas formas de fazer a vetorização do código em questão e optou pela segunda opção para essa análise por ter obtido melhores resultados nos testes iniciais:

1. Armazenar cada vetor tridimensional em um `__m128`, deixando 32 bits inutilizados e otimizando diretamente operações sobre os vetores como produto escalar.

2. Armazenar cada componente de múltiplos vetores em um `__m128`, ou seja, agrupar os vetores em grupos de 4 e alocar um vetor de operação SIMD para cada uma das componentes x, y e z do grupo. Os grupos representam blocos de 2×2 pixeis na imagem final.

Além disso, vale ressaltar que a implementação vetorizada é uma tradução quase 1 para 1 do código original, apenas substituindo os comandos tradicionais

por operações *intrinsics*. A única forma de otimização rebuscada aplicada foi a substituição de *if*s que causariam *branches* pela computação de ambos caminhos e a utilização de uma máscara para definir qual valor será utilizado.

4 Análise de Desempenho

Tempo de renderização dos frames (ms) com nível de confiança 95%

Sem flags de otimização		**Com flags de otimização**	
Sem SIMD	**Com SIMD**	**Sem SIMD**	**Com SIMD**
569	223	84	49
577	224	88	49
578	226	84	49
584	224	85	49
584	224	84	49
585	223	83	48
586	224	86	49
586	225	83	49
572	225	83	48
588	225	83	50
580.90 ± 26.37	224.30 ± 0.56	84.3 ± 1.66	48.90 ± 0.20

Primeiramente, é possível notar que a aplicação manual das instruções vetorizadas causou uma diminuição drástica no tempo de execução em ambos os casos, diminuindo o tempo por um fator 2.59 sem as flags de otimização e por um fator 1.66 no caso com as flags.

Além disso, nessa aplicação complexa, é notável que apenas aplicar manualmente as operações SIMD não causa um aumento de desempenho tão grande quanto ativar as flags de otimização do compilador. No entanto, combinar a

utilização das flags com a vetorização manual se mostrou a melhor opção em termos do desempenho final.

5 Conclusão

Apesar de uma grande perda de legibilidade no código, notou-se que a aplicação das instruções SIMD manualmente pode representar um grande aumento na performance final, principalmente quando aliado às outras otimizações feitas automaticamente pelo compilador. Portanto, conclui-se que, caso as otimizações automáticas não sejam suficientes para alcançar o desempenho desejado da aplicação, o uso manual de *intrinsics* é uma opção viável. Vale ressaltar que podem haver múltiplas formas de se vetorizar um mesmo código e, por isso, a experimentação com casos reais é muito importante.